Implementação de módulos de kernel Linux para simulação e proveniência de Sistemas Multiagentes Embarcados

Bruno Policarpo Toledo Freitas[1], **Carlos Eduardo Pantoja**[1]

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Av. Maracanã, 229 - Maracanã – Rio de Janeiro/RJ - CEP: 20271-110

Abstract. *A Multi-agent system (MAS) is a group of autonomous agents capable of deliberating and acting upon the world. A common model to implement those systems is the Belief-Desire-Intention (BDI), based upon human reasoning. Agents can also be used to implement cognition for embedded and cyber-physical systems. Given the abstract nature of agent models, various middleware have been proposed to ease the development of these systems. However, considering embedded systems, these abstractions can negatively impact the systems and miss opportunities of using operating systems features for embedded MAS development. This work proposes a different way of developing these systems by using Linux kernel modules. In this work, we show the current development of two modules: a serial channel emulator and a provenance module for real Embedded MAS devices. The serial channel emulator presented seamlessly connects an agent's reasoning to a simulator. The provenance module aims to transparently capture data exchanged between the agent and the embedded device.*

Resumo. *Sistemas Multiagentes (SMA) são um grupo de agentes capazes de deliberar e agir sobre um ambiente de acordo de forma autônoma. Um modelo bastante comum de implementar tais sistemas é o modelo Belief-Desire-Intention (BDI), baseado no raciocínio humano. Agentes também podem ser utilizados para o desenvolvimento de softwares embarcados e ciberfísicos a fim de prover uma camada cognitiva aos mesmos. Tendo em vista a natureza abstrata do modelo de agentes, diversos middlewares tem sido propostos para fornecer as abstrações necessárias para implementação desses sistemas. Todavia, no caso de sistemas embarcados, a utilização de abstrações de mais alto nível podem impactar negativamente o sistema e perde-se oportunidades de explorar características do sistema operacional no desenvolvimento desses sistemas. Nesse sentido, este artigo apresenta o andamento do desenvolvimento de dois módulos de kernel Linux para para SMAs embarcados: um para simulação de SMAs embarcados e outro para Proveniência. Para a simulação de SMAs embarcados, o módulo criado fornece um canal de comunicação virtual e genérico entre um SMA e simuladores, sendo mostrado como o canal pode ser utilizado para tal finalidade. Já o módulo de proveniência visa capturar os fluxos de dados de um canal de comunicação serial de um SMA embarcado executado no mundo real de forma transparente.*

1. Introdução

Um sistema multiagente (SMA) é um grupo de agentes autônomos capaz de receber percepções e agir em um ambiente virtual ou computacional [Michel et al. 2009]. Agentes diferem de softwares convencionais pois são independentes, adaptáveis, pró-ativos, com cognição, sendo capazes também de colaborar entre si [Hübner et al. 2004]. O modelo cognitivo de agentes *Belief-Desire-Intention* (BDI) [Bratman 1987], baseado no raciocínio humano, costuma ser utilizado para implementar a camada de raciocínio, permitindo que eles deliberem sobre quais objetivos atingir, baseados em crenças, desejos e intenções.

Os SMAs também têm sido utilizado para adicionar uma camada cognitiva em sistemas ciberfísicos, IoT, e computação na borda [Karaduman et al. 2023]. Também têm sido usados para o desenvolvimento de sistemas robóticos [Silvestre et al. 2023], criando novas camadas de abstrações ou middleware para fazer com que os agentes consigam controlar recursos de hardware a fim de agir no mundo real [Dal Moro et al. 2022b, Dal Moro et al. 2022a]. A criação de middleware para diversos domínios de problema é recorrente, e sua definição varia conforme a área mas, de forma geral, é a ponte entre uma aplicação e um sistema operacional, facilitando o seu desenvolvimento, aumentando a escalabilidade, e facilitando sua manutenção e automação [Gazis and Katsiri 2022]. Diversos middlewares tem sido propostos na literatura para facilitar o desenvolvimento de SMAs, tais como o SPADE3 [Palanca et al. 2020], assim como formas de adaptar middlewares robóticos, tais como o Robot Operating System (ROS) [Open Robotics 2023b], para adicionar uma camada de raciocínio ao sistema [Gavigan and Esfandiari 2021].

Nesses casos, tais middlewares são construídos em cima de camadas de software a fim de prover as abstrações pretendidas. Porém, ao se olhar nessas abstrações, perde-se a oportunidade de explorar características do hardware e do sistema operacional para o desenvolvimento de SMAs. Especificamente, este trabalho irá apresentar o andamento de dois trabalhos sobre SMAs embarcados baseados em sistemas operacionais: integração de simuladores e proveniência de SMAs embarcados. O primeiro trabalho irá apresentar o estado atual de integração de SMAs com simuladores, no caso, o simulador robótico WeBOTS [Michel 1998]. Já o segundo apresenta uma proposta de solução para captura de proveniência de SMAs sistemas embarcados, de forma a permitir que a comunicação entre um SMA com microcontroladores seja salva de forma invisível e sem alteração do SMA.

Este trabalho está organizado da seguinte forma: na Seção 2 mostra o andamento do trabalho sobre o uso de simuladores para SMAs embarcados utilizando a arquitetura ARGO. A Seção 3 mostra o andamento do trabalho acerca de um módulo de proveniência para SMAs embarcados. Finalmente, A Seção 4 faz uma discussão de trabalhos apresentados.

2. Simuladores

Esta seção irá apresentar o andamento de trabalhos de integração de simuladores para o desenvolvimento de SMAs embarcados. Especificamente, será abordado o andamento atual da integração com simuladores robóticos. Primeiro, será dado um breve panorama do uso de simuladores para o desenvolvimento desses sistemas, para então apresentar o andamento atual da integração com o simulador robótico WeBOTS.

Uma solução popular para o desenvolvimento de sistemas robóticos no geral é usar a plataforma *Robot Operating System* (ROS) [Open Robotics 2023b] para o desenvolvimento do sistema robótico em conjunto com o simulador Gazebo [Open Robotics 2023a] para realizar teste e avaliação da solução. Existem trabalhos que visam realizar o desenvolvimento do software robótico por meio de um SMA descrito em Jason, em que o ROS é utilizado como uma ponte entre o agente e uma camada de aplicação, onde esta última é a responsável pela comunicação com o ambiente de simulação ou o mundo real [Gavigan and Esfandiari 2021]. Porém, tal abordagem ainda precisa da construção dessa camada de aplicação específica para cada domínio de problema, além de ser computacionalmente mais custosa por possuir uma hierarquia com três camadas de abstração de software.

Em contrapartida, pode-se utilizar o próprio kernel do sistema operacional como uma camada de ligação do raciocínio do agente com um simulador. Essa configuração é mostrada na Figura 1. Nesse caso, o raciocínio do agente não precisa ser alterado e o simulador pode, a priori, ser qualquer um. O canal de comunicação serial emulado já existe e é funcional, assim como diversos simuladores já foram usados.

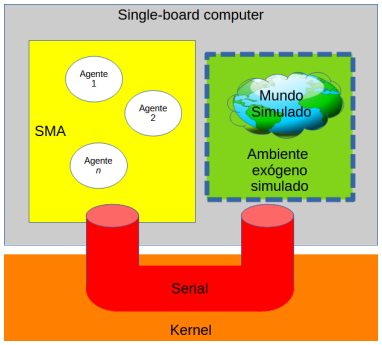

Figura 1. Canal de comunicação do agente com simulador via sistema operacional

As Listagens 1 e 2 ilustram como o canal é utilizado para conectar o raciocínio do agente ARGO com um simulador. Neste exemplo, foi utilizado o simulador WeBOTS para simular um protótipo físico 2WD. O agente ARGO precisa somente indicar como crença inicial a porta emulada de conexão, no caso *ttyEmulatedPort0*. Já no simulador é necessário mapear as ações correspondentes *goAhead*, *goBack*, *goLeft*, e *goRight* em comandos propícios do simulador e o comando *getPercepts* para retornar os valores dos sensores.

Listagem 1. ”Código do agente Jason”

```
/* Initial beliefs and rules */
serialPort(ttyACM0).

/* Initial goals */

!start.
/* Plans */
+!start: serialPort(Port) <- argo.port(Port); argo.
    percepts(open); argo.limit(750); !rollOut.

+!rollOut: dLeft(DL) & dRight(DR) & (DR >= 20) & (DL
    >= 20) & not wall
                          <- argo.act(goAhead);
    .wait(500); !rollOut.

+!rollOut: dLeft(DL) & dRight(DR) & ((DR < 20) | (DL
    <20)) & ((DR >= 10) | (DL >=10)) & not wall <-
    argo.act(stop); +wall; .wait(500); !rollOut.

+!rollOut: dLeft(DL) & dRight(DR) & ((DR < 10) | (DL
    <10)) & not wall
                          <- argo.act(goBack); +
    wall; .wait(500); !rollOut.

+!rollOut: dLeft(DL) & dRight(DR) & wall & ((DR < 30)
     | (DL <30) | (DR>100) | (DL>100))
            <- argo.act(goRight); .wait(500); !
    rollOut.

+!rollOut: dLeft(DL) & dRight(DR) & wall & ((DR >=
    30) | (DL >= 30))
                          <- -wall; !rollOut.

-!rollOut.

+dLeft(DL)  <- .print("Left Distance: ",DL).
+dRight(DR) <- .print("Right Distance: ",DR).
```

Listagem 2. ”Código do simulador 4WD”

```
if ( ! strcmp( javino_received_msg , "getPercepts"
    ) ){

  // left distance sensor value
  float d1 = wb_distance_sensor_get_value( ds[0]
      );

  // right distance sensor value
  float d2 = wb_distance_sensor_get_value( ds[1]
      );

  // Composing percepts message to send to Javino
  sprintf(percepts_msg,
    "dLeft(%.1f);dRight(%.1f);",
    d1, d2 );

  javino_send_msg( exogenous_port,
     percepts_msg);

  free( javino_received_msg );

} else if ( ! strcmp( javino_received_msg , "
    goAhead" ) ){

  left_speed = 1.0;
  right_speed = 1.0;

} else if ( ! strcmp( javino_received_msg , "
    goRight" ) ){

  left_speed = 1.0;
  right_speed = 0.0;

} else if ( ! strcmp( javino_received_msg , "
    goBack" ) ){

  left_speed = -1.0;
  right_speed = -1.0;

}
```

3. Proveniência de SMAs embarcados

Proveniência de um produto de dados informa sobre os processos e dados usados para gerá-lo [Freire et al. 2008], dessa maneira possibilitando reproduzir e validar experimentos científicos. Diferentemente do problema de simulação apresentado na Seção 2, um experimento no mundo real com SMAs embarcados gera um fluxo de dados oriundos dos sensores e as ações enviadas pelo SMA ao hardware. Nesse caso, os dados gerados são salvos apenas se houver uma instrumentação específica dentro do SMA.

De acordo com a arquitetura ARGO de SMAs embarcados, existe um ciclo bem definido de fluxos de dados onde, primeiro, o SMA recebe dados dos sensores por meio de um *getPercepts* e, de acordo com as crenças atualizadas pelos sensores, envia ações por meio de diretivas *act* do Jason. Ou seja, em termos de proveniência, o importante é capturar os dados trocados nesse ciclo. Dessa maneira, esses fluxos de dados podem ser usados posteriormente para depuração ou até mesmo para reproduzir o experimento com a ajuda de um simulador.

Para isso, está sendo desenvolvido um módulo de kernel Linux responsável por ”espionar”periodicamente o canal de comunicação serial para capturar os dados trafegando nele. Tal abordagem torna transparente para o SMA o processo de captura de proveniência - ou seja, o SMA não precisa realizar logging nem precisa ser alterado para fazer isso. Essa arquitetura é mostrada na Figura 2. O módulo captura os dados trafegando no canal por meio de interrupções periódicas, armazenando-os temporariamente em seus buffers internos. Posteriormente, um ”Processo Coletor”no espaço de usuário coleta tais dados, também de forma periódica, salvando-os então no meio de armazenamento. Por

estar dentro do kernel, o módulo pode obter dados com uma frequência superior ao do Processo Coletor e a um custo de processamento menor pois assim evitam-se o custos de trocas de contexto dos processos de usuários para o kernel e das chamadas do sistema do sistema operacional para leitura e escrita de dados.

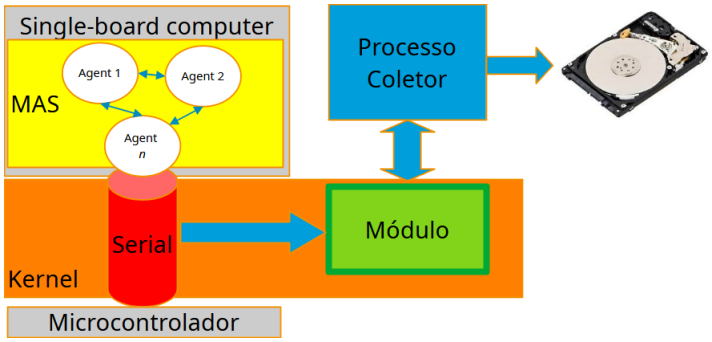

Figura 2. Módulo de proveniência proposto para SMAs embarcados

Com relação ao andamento atual deste trabalho, o módulo já consegue acessar e capturar dados do canal, e já existe uma interface kernel-espaço de usuário para que o Processo Coletor possa fazer a coleta dos dados. No estado atual, o módulo captura dados a uma frequência de 10 HZ, e o Processo Coletor os coleta com um período de 2s, mas isso é facilmente modificável. O principal desafio no momento é traduzir os dados coletados do canal nas ações Jason e dados dos sensores, devido ao fato deles estarem sendo capturados de forma bruta. Também, é ainda uma questão em aberto se a abordagem por amostragem pode resultar em perdas de dados significativas entre dois eventos de captura.

4. Discussão

A utilização de módulos de kernel é algo pouco explorado na literatura sobre SMAs. Tradicionalmente, o desenvolvimento de SMAs é feito com a ajuda de middlewares em alto nível. As vantagens de se utilizar tal abordagem é algo já bem estabelecido na literatura, e já existem diversos middlewares para o desenvolvimento de SMAs.

Este trabalho propõe fazer o movimento inverso, mostrando o andamento de diversos trabalhos de SMAs embarcados utilizando o sistema operacional como base. Foram apresentados trabalhos utilizando o sistema operacional como ponte entre um SMA e simulador e como provedor de proveniência para SMAs embarcados.

Referências

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

Dal Moro, D., Robol, M., Roveri, M., and Giorgini, P. (2022a). A demonstration of bdi-based robotic systems with ros2. In Dignum, F., Mathieu, P., Corchado, J. M., and De La Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection*, pages 473–479, Cham. Springer International Publishing.

Dal Moro, D., Robol, M., Roveri, M., and Giorgini, P. (2022b). Developing bdi-based robotic systems with ros2. In Dignum, F., Mathieu, P., Corchado, J. M., and De La Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection*, pages 100–111, Cham. Springer International Publishing.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21.

Gavigan, P. and Esfandiari, B. (2021). Agent in a box: A framework for autonomous mobile robots with beliefs, desires, and intentions. *Electronics*, 10(17).

Gazis, A. and Katsiri, E. (2022). Middleware 101. *Commun. ACM*, 65(9):38–42.

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.

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.

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`.

Michel, O. (1998). Webots: Symbiosis between virtual and real mobile robots. In Heudin, J.-C., editor, *Virtual Worlds*, pages 254–263, Berlin, Heidelberg. Springer Berlin Heidelberg.

Open Robotics (2023a). Gazebo. `https://gazebosim.org/home/`.

Open Robotics (2023b). Robot operating system. `https://www.ros.org/`.

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

Silvestre, I., de Lima, B., Dias, P. H., Buss Becker, L., Hübner, J. F., and de Brito, M. (2023). Uav swarm control and coordination using jason bdi agents on top of ros. 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 225–236, Cham. Springer Nature Switzerland.