Adição de Recursos em Tempo de Execução a Sistemas Multi-Agentes Embarcados

Nilson Mori Lazarin[1,2], Carlos Eduardo Pantoja[1,2], Vinicius Souza de Jesus[2]
Fabian Cesar Pereira Brandão Manoel[1], José Viterbo Filho[2]

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

[2]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ – Brazil

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

{souza.vdj,fabiancpbm}@gmail.com, viterbo@ic.uff.br

Abstract. *An Embedded MAS (Multi-Agent Systems) can use different communication infrastructures, exchange information, and even transport internal agents to another system. These systems can interact with a dynamic environment through their resources (sensors and actuators). However, resources are static and defined at design time. The addition of a resource, no matter how simple, forces the shutdown of the embedded system and several changes are necessary for the layers of hardware, firmware, interfacing, and reasoning. This work presents an approach that guarantees the expected adaptive capacity of a Cognitive MAS by adding resources in runtime at an Embedded SMA.*

Resumo. *Um SMA (Sistema Multi-Agente) Embarcado consegue utilizar diversas infraestruturas de comunicação, trocar informações e até mesmo transportar agentes internos para outros sistemas. Através de seus recursos (sensores e atuadores) esses sistemas podem interagir com um ambiente dinâmico. Entretanto, os recursos são estáticos e definidos em tempo de projeto. A adição de um recurso, por mais simples que seja, obriga o desligamento do sistema embarcado, pois, são necessárias diversas alterações nas camadas de hardware, firmware, interfaceamento e raciocínio. Este trabalho apresenta uma abordagem que garante a capacidade adaptativa esperada de um SMA Cognitivo, através da adição de recursos em tempo de execução para SMA Embarcados.*

1. Introdução

A Inteligência Artificial Distribuída é um campo da Ciência que permite a construção e aplicação de Sistemas Multi-Agentes (SMA), com agentes capazes de interagir para alcançar algum conjunto de objetivos ou executar algum conjunto de tarefas [Weiss 2000]. Os agentes são entidades reais ou virtuais que se comportam de forma autônoma, percebem e agem em um ambiente e se comunicam com outros agentes [Alvares and Sichman 1997]. Um SMA é formado por um grupo de agentes autônomos fracamente conectados, com objetivos comuns ou não, que atuam em um mesmo ambiente, cooperando ou competindo, compartilhando ou não conhecimento entre si [Balaji and Srinivasan 2010]. Diferente de outros paradigmas da Inteligência Artificial (IA), os SMA investigam a coletividade e não um único indivíduo [Hübner et al. 2004].

Um SMA pode ser classificado como: *reativo*, onde o comportamento inteligente da sociedade de agentes vem da interação entre um grande número agentes simples; ou *cognitivo*, onde cada agente é sofisticado e sua sociedade é formada por uma pequena quantidade de membros [Hübner et al. 2004]. Uma das arquiteturas para o desenvolvimento de agentes cognitivos é o modelo *Belief-Desire-Intention* (BDI) [Michel et al. 2009]. Este modelo está fundamentado no entendimento do raciocínio prático humano que decide, momento a momento, qual ação realizar para alcançar nossos objetivos [Wooldridge 2000]. Possibilitando, dessa forma, a implementação de atitudes mentais (*crenças, desejos e intenções*) em agentes cognitivos [Hübner et al. 2004].

Um SMA Embarcado é um sistema cognitivo baseado em: hardware, responsável pela atuação no ambiente real; e software, responsável pelo raciocínio e controle. Nesta categoria os agentes cognitivos estão fisicamente conectados ao hardware, ou seja, não há controle remoto ou processamento externo enviado via rede. Para possibilitar a atuação de um agente no mundo real, utiliza-se uma arquitetura de quatro camadas onde: a camada de hardware é formada pelo conjunto de *recursos*[1] que expressam a manifestação do agente no mundo físico; a camada de *firmware* é hospedada em um ou mais microcontroladores que manipulam a camada de hardware, conforme as deliberações do agente; a camada de interfaceamento permite a comunicação do agente com o microcontrolador; por fim, a camada de raciocínio é hospedada em um computador que executa o ciclo cognitivo do agente [Lazarin and Pantoja 2015, Pantoja et al. 2016, Souza de Castro et al. 2022].

Por fim, um SMA também pode ser aberto ou fechado. O modelo aberto permite a interação de agentes autônomos, ou ainda, a migração entre diferentes SMA. No caso de um SMA Aberto e Embarcado, os agentes podem abandonar o hardware em caso de falha, carregando consigo suas crenças, desejos e intenções para outro SMA em um ambiente físico ou virtual. Para tal, existem modelos de transferência inspirados em relações ecológicas de predação, mutualismo ou inquilinismo, onde a migração ocorre com objetivos de dominar, co-participar ou sobreviver em SMA de destino, respectivamente. Dessa forma, é possível que um agente autônomo seja adaptável, continuando a executar ações para alcançar seus objetivos em caso de insucesso, mesmo em um hardware embarcado [Jesus et al. 2018, de Jesus et al. 2019, Souza de Jesus. et al. 2021].

Do ponto de vista da embarcação, um agente cognitivo consegue interagir com um ambiente dinâmico através de seus sensores e atuadores [Lazarin and Pantoja 2015, Pantoja et al. 2016], utilizar diversas infraestruturas de comunicação [Lazarin et al. 2021, Souza de Castro et al. 2022], migrar de SMA, trocar informações, sobreviver ou até mesmo e atuar em outro hardware [de Jesus et al. 2019, Souza de Jesus. et al. 2021]. Entretanto, os *recursos* do SMA para atuação no mundo real são definidos apenas em tempo de projeto e a impossibilidade de alteração, em tempo de execução, limita a capacidade adaptativa esperada de um sistema cognitivo.

Dessa forma, este trabalho apresenta uma abordagem que visa diminuir as limitações da capacidade adaptativa do agente, através da adição de *recursos* em tempo de execução em SMA Embarcados. Portanto, um SMA embarcado que já possua recursos físicos disponíveis, poderá ter novos recursos acoplados a si, e os agentes com

[1] Atuadores e sensores digitais ou analógicos [Pantoja et al. 2019, Brandão et al. 2021]

o conhecimento necessário para controle desse novo dispositivo serão transmitidos em tempo de execução. Para isso, será a utilizada uma arquitetura de hardware com placas do tipo *System on a Chip* [Upton and Halfacree 2013] para hospedar o SMA embarcado e microcontroladores para gerenciamento de sensores e atuadores. O SMA embarcado será desenvolvido usando o Jason [Bordini et al. 2007] com agentes customizados Argo (para interfaceamento de hardware) e agentes comunicadores capacitados com protocolos bio-inspirados de comunicação (para transferência de agentes através da internet) [Pantoja et al. 2016, Souza de Jesus. et al. 2021].

Um estudo de caso será apresentado através de um cenário contendo um robô com três sensores de obstáculos e um sensor de luz, gerenciado por um SMA embarcado. Um *buzzer* será adicionado em tempo de execução. O robô se moverá toda vez que o sensor de luz for estimulado e ao perceber algum obstáculo a frente ou nas laterais, ele irá desviar. Em caso de não existir possibilidade de movimentação, o *buzzer* soará um alarme sonoro. As contribuições esperadas deste trabalho é provar ser possível a inserção de recursos físicos em tempo de execução em SMA embarcados utilizando o modelo BDI.

Este trabalho está estruturado da seguinte forma: na Seção 2 serão apresentados os trabalhos relacionados; na Seção 3 a metodologia proposta; o estudo de caso é apresentado na Seção 4; e por fim são apresentadas as considerações finais (Seção 5) e as referências utilizadas nesse trabalho.

2. Trabalhos Relacionados

A adição de recursos em tempo de execução não é uma tarefa fácil, do ponto de vista prático, considerando as diversas etapas que um sistema embarcado precisa passar para ser concebido. Quando se trata de um SMA Embarcado, além da tradicional etapa de preparo do Hardware e da programação do Firmware, é preciso se preocupar com *middleware* para interfaceamento e com a cognição do dispositivo (o SMA). Existem diversas soluções que tratam independentemente dessas etapas, ou então, permitem integrações em tempo de design. A arquitetura ARGO [Pantoja et al. 2016] adiciona uma nova categoria de agentes BDI capaz capturar e filtrar [Stabile Jr. et al. 2018] as percepções de um ambiente físico e dessa forma, processar diretamente as percepções como crenças e reduzir a carga de processamento na camada de raciocínio, permitindo o agente focar apenas nas percepções que lhe convém. Esses trabalhos possibilitam a inclusão de microcontroladores (camada de firmware) que podem gerenciar um ou mais recursos (camada de hardware) conectados ao sistema. Entretanto, em ambos, a limitação na camada de raciocínio e de interfaceamento, obriga o desligamento do SMA Embarcado, para que o ambiente simulado e o raciocínio do agente sejam ajustados.

Em outro exemplo, a Arquitetura de Gerenciamento de Recursos (*Resource Management Architecture* — RMA) [Pantoja et al. 2019] permite o gerenciamento de recursos na borda de sistemas sobre uma infraestrutura da Internet das Coisas (*Internet of Things* — IoT). Dessa forma, SMA Embarcados podem atuar para gerenciar dispositivos de forma remota e se comunicarem com outros SMA Embarcados de outros dispositivos. É possível também que artefatos físicos atuem como recursos, sem a necessidade de agentes dedicados à tarefa. As informações são repassadas a uma camada de gerenciamento que pode ser consumida no formato *Sensor as a Service*. Esses trabalhos, evoluem a relação do SMA e o gerenciamento de *recursos* físicos disponíveis em um sistema de computação

na borda. Entretanto, apesar do dinamismo da arquitetura IoT, apenas novos dispositivos podem ser adicionados a rede em tempo de execução. A modificação de dispositivos existentes só é possível em tempo de design, impossibilitando a inclusão de novos *recursos*, sem o desligamento do SMA.

A utilização de protocolos para transporte de agentes [Souza de Jesus. et al. 2021] possibilita que um SMA Embarcado em uma plataforma robótica assuma o controle de outra plataforma, através da movimentação de todos seus agentes e respectivos estados mentais. Com isso é possível preservar a integridade dos conhecimentos dos agentes da plataforma de origem. Entretanto, a plataforma de destino deve ser idêntica à plataforma de origem para o controle efetivo do hardware para não haver a necessidade de aprendizado de novas habilidades, além de não ser possível incluir recursos adicionais na plataforma de destino em tempo de execução. O SMA ainda estaria ligado a um conjunto fixo de recursos.

Este trabalho apresenta uma abordagem que explora individualmente todas as soluções acima para permitir a adição de *recursos* em tempo de execução a um SMA Embarcado. Como um SMA Embarcado utiliza uma arquitetura física com placas que permitem rodar um sistema operacional e com interfaceamento serial entre agentes e microcontroladores, é possível adicionar recursos em tempo de execução. Em seguida, como é possível a comunicação e movimentação entre agentes de um SMA para outro utilizando os protocolos bio-inspirados, é possível programar o agente em tempo de design e movê-lo em tempo de execução adotando um protocolo que não elimine o SMA de destino (o inquilinismo, por exemplo). Dessa forma, conhecendo a porta serial onde o novo dispositivo estiver conectado, e enviando o agente preparado para manipulá-la, afirmamos ser possível adicionar um recurso controlável por agentes BDI em tempo de execução em um SMA Embarcado.

3. Metodologia

Um *dispositivo* é composto por um microcontrolador integrado a uma placa do tipo *sistema em um chip*, por um *middleware* que realiza a comunicação serial entre essas partes para levar informações de recursos (sensores e atuadores) a um SMA Embarcado [Brandão et al. 2021]. O uso de *recursos* é importante para SMA Embarcados, visto que um *dispositivo* possui a capacidade de atuação e percepção no mundo real. Entretanto, os *recursos* de um SMA Embarcado são estáticos, ou seja, são incluídos em um *dispositivo* em tempo de projeto e não podem ser removidos ou incluídos em tempo de execução. O atual processo de construção desses sistemas é composto pelas seguintes etapas:

- **Definição de *recursos***: conforme o domínio de atuação, os *recursos* desejáveis devem ser escolhidos, visto que existem diversas limitações na capacidade de processamento e armazenamento no microcontrolador.
- **Construção do SMA Embarcado**: sensores e atuadores normalmente necessitam de resistores, potenciômetros ou capacitores para funcionarem corretamente, dessa forma, é necessário planejar e construir o corpo físico do SMA.
- **Programação do firmware**: muitos sensores utilizam bibliotecas específicas para funcionamento, além disso, é necessário definir os comandos a serem enviados pelo agente e aceitos pelo microcontrolador.

- **Programação do ambiente**: para atuação no mundo real, é necessário representar virtualmente o ambiente em que os agentes-BDI estão inseridos.
- **Programação do SMA**: para a deliberação do agente é necessário programar as crenças pré-definidas, os desejos e as intenções.
- **Configuração do sistema operacional**: é necessário transferir os arquivos para o dispositivo que hospedará o SMA, além de configurar tarefas e serviços a serem executados na inicialização.

Na metodologia proposta, é possível adicionar recursos em tempo de execução em um dispositivo já projetado que esteja utilizando a arquitetura descrita acima. Utilizando uma placa capaz de hospedar um sistema operacional e um extensor de portas seriais, é possível adicionar até 127 dispositivos. Contudo, recursos (sensores e atuadores) dependem de conexão a um microcontrolador. Dessa forma, todo e qualquer recurso que precise ser inserido em tempo de execução precisa estar atrelado a um microcontrolador e conectado a uma porta serial. O SMA Embarcado faz interfaceamento do hardware através de comunicação serial identificando em qual porta o microcontrolador está conectado (agentes Argo). Então, para este agente interfacear estes novos recursos, bastaria ele saber a qual porta acessar em tempo de execução. Contudo, a porta não está disponível para ele, e mesmo que estivesse, este não saberia como manipulá-la e precisaria aprender estas habilidades de outra maneira.

Em SMA Embarcados que se comunicam com redes IoT, existem agentes Comunicadores responsáveis pela identificação de todo o sistema na rede e comunicação com outros SMA Embarcados. Esses agentes conseguem enviar mensagens com performativas KQML e em casos específicos, permitir a transferências de agentes ou do sistema inteiro para outro SMA Embarcado. No caso dos protocolos bio-inspirados [Souza de Jesus. et al. 2021], três são as possibilidades: o *predatismo*, que elimina o sistema de destino quando o dispositivo de origem é incapacitado; o *inquilinismo*, que permite a co-existência de todos os agentes no destino quando o dispositivo de origem é incapacitado; e o *mutualismo*, que permite o envio de um ou mais agentes e seu retorno é previsto, não havendo incapacitação do dispositivo de origem. Considerando estes protocolos, o inquilinismo tem características que permitem a transferência de forma não destrutiva, pois não interrompe a execução os agentes no destino, e elimina o SMA da origem, não deixando rastros.

Então, ao conectar um novo microcontrolador com recursos em tempo de execução em um SMA Embarcado, é possível consultar qual porta este novo dispositivo foi inserido e, em outro ambiente de desenvolvimento, criar um SMA com um agente interfaceando esta porta com as habilidades necessárias e desejadas. Em seguida, invocar o protocolo de inquilinismo para transferir o agente para o SMA Embarcado onde os recursos foram adicionados. Uma vez que o agente chegar no destino, este conseguirá controlar os novos recursos e interagir com os demais agentes existentes naquele sistema. O SMA no ambiente de desenvolvimento ficará inutilizado e o SMA de destino não será afetado.

3.1. Abordagem Proposta

Para ser possível a adição em tempo de execução de recursos, o SMA Embarcado que terá recursos adicionados deve ser construído observando as seguintes etapas do atual processo de construção:

- **Construção do SMA Embarcado**: é necessário planejar e construir o corpo físico do dispositivo onde o SMA está embarcado com a capacidade de adição de recursos via USB.
- **Programação do SMA**: é necessário programar o SMA com um agente comunicador, habilitado para o protocolo de inquilinismo, apto a receber agentes para as tarefas a serem desempenhadas pelos novos recursos adicionados.

Em seguida, o projetista precisa se atentar para a criação dos novos recursos a serem adicionados e a programação do SMA que invocará o protocolo de inquilinismo e enviará os agentes para o SMA anterior. Este novo recurso segue as atuais etapas do processo de construção. Contudo, as seguintes etapas devem ser observadas:

- **Definição e construção do *recurso***: conforme o domínio de atuação, escolher os novos *recursos* desejáveis, que serão conectados ao dispositivo anterior.
- **Programação do SMA**: é necessário programar o SMA com um agente comunicador, habilitado para o protocolo de inquilinismo, apto a enviar os agentes criados para as tarefas a serem desempenhadas pelos novos recursos adicionados. Nesta etapa, o projetista precisa consultar em qual porta serial o novo dispositivo foi conectado para o devido interfaceamento no SMA Embarcado de destino.

A intenção prática é a criação de um dispositivo inicial que consiga receber novos recursos em tempo de execução a partir da disponibilização de portas seriais para conexão de novos microcontroladores e em seguida, a criação de recursos atrelados a microcontroladores, interfaceados por um SMA Embarcado que será transferido para o dispositivo inicial. Ao iniciar o SMA os agentes com habilidades para controlar os novos recursos, serão transferidos para o SMA Embarcado onde os recursos estarão fisicamente conectados. Um comparativo entre a abordagem atual e a proposta neste trabalho é apresentada na Figura 1.

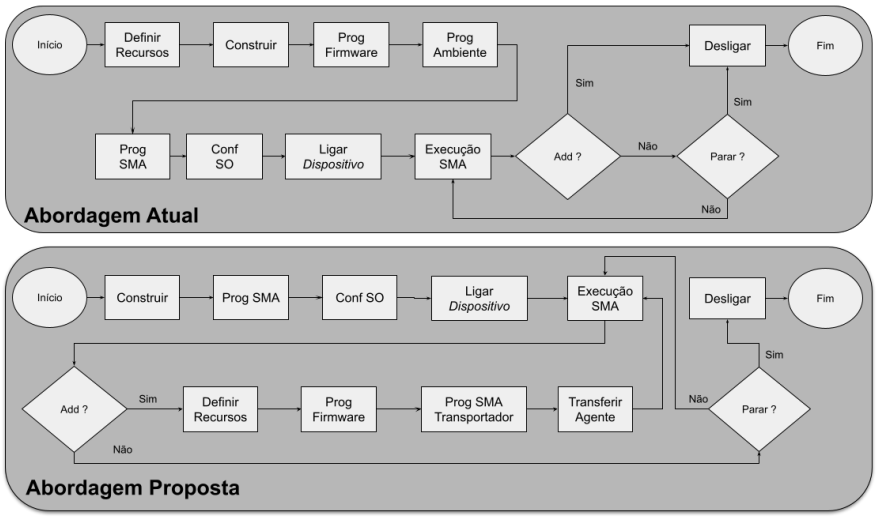


Figura 1. Comparativo entre a abordagem atual e a abordagem proposta

4. Estudo de Caso

Para exemplificar esta proposta, podemos considerar o seguinte cenário: deseja-se embarcar um SMA em um robô para evitar obstáculos. O robô deverá ser acionado através da detecção de luz e deve mudar de direção dependendo dos obstáculos a frente.

4.1. Implementação e atualização no modelo atual

Nesta seção serão descritas as etapas para a implementação e atualização no modelo atual. Para implementação são necessárias as seguintes ações:

Definição de Recursos: nesta etapa são identificados os recursos necessários para atendimento do proposto no estudo de caso. Para atender ao cenário serão utilizados os seguintes *recursos*: 01 conjunto robótico 2WD com uma ponte H; 03 sensores de obstáculo digitais; e 01 sensor de luz.

Componentes para construção: nesta etapa são listados os componentes necessários para a hospedagem do SMA Embarcado e controle dos recursos. Será necessário: 01 Arduino UNO; 01 Adaptador OTG e 01 cabo USB AM/BM; 01 Raspberry Pi Zero W e 01 Cartão MicroSD 4 GB. O esquema da construção é apresentada na Figura 2a.

Programação do Firmware: Será necessário importar o Javino[2] e programar o microcontrolador Arduino para receber os comandos enviados pelo agente.

Programação do Ambiente: Na camada de interfaceamento será necessário importar o Javino para possibilitar a comunicação com o firmware. Além disso, é necessário programar cada ação no ambiente simulado, de forma que o firmware consiga executar corretamente.

Programação do SMA: Na camada de raciocínio o agente deverá ser programado para atuar no ambiente.

Configuração do SO: Será necessário o download e instalação do Raspbian[3] no cartão de memória. Será necessário instalar o pacote: openjdk-8-jre e a biblioteca pyserial. Por fim, será necessário transferir os binários do Jason Framework para o cartão e configurar a execução automática do SMA.

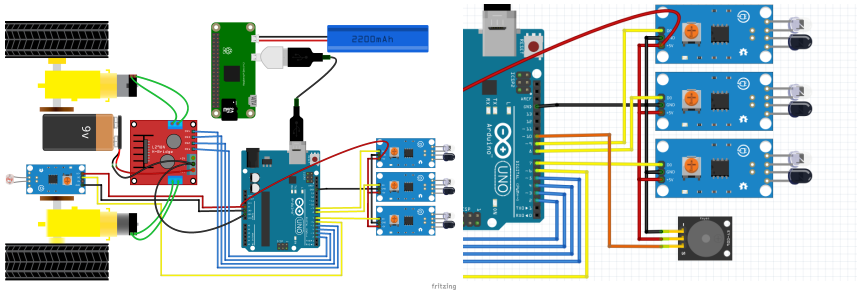
Para exemplificar a atualização, utilizando o modelo proposto vamos considerar a necessidade de inclusão de um novo *recurso* para realizar a seguinte tarefa: quando o robô estiver impossibilitado de mudar de direção, ele deverá emitir um alerta sonoro. Serão necessárias intervenções em todas as etapas. Além do fato de que o robô deverá ser desligado. Abaixo são descritas as etapas necessárias:

Definição de recursos: o primeiro passo é verificar a disponibilidade de portas no Arduino para inclusão do novo recurso; para atender ao solicitado será necessário o seguinte recurso: 01 módulo buzzer.

Construção: será necessário adicionar um jumper entre o módulo buzzer e a porta digital do Arduino, além de ser necessário realizar uma emenda aos jumpers de 5v e GND para alimentação do módulo. O desligamento se torna obrigatório para realizar os ajustes necessários. O esquemático da adição do *recurso* é apresentada na Figura 2b.

Programação do Firmware: para realizar a alteração na camada de *firmware* é necessário acesso ao código-fonte, caso contrário o Firmware deverá ser totalmente reescrito. É importante ressaltar que nesta etapa o microcontrolador deverá ser retirado do robô, visto que o *deploy* é realizado diretamente via USB. Mesmo que fosse possível adicionar o recurso na camada de hardware sem desligar o robô, esta ação torna obrigatório o desligamento.

[2] `http://javino.sf.net/`
[3] `http://www.raspbian.org/`



(a) Construção do SMA Embarcado.

(b) Recurso Buzzer adicionado ao Hardware.

Figura 2. Esquemático da implementação e atualização no modelo atual

Programação do Ambiente: será necessário alterar os arquivos que representam o ambiente simulado para o agente, para que as novas ações do SMA sejam reconhecidas. É importante ressaltar que nesta etapa o SMA deverá ser parado, mesmo que a transferência ocorra via rede sem fio, o sistema operacional do robô deverá parar a execução do processo do SMA e iniciar um novo processo com o ambiente atualizado. No pior caso, o desligamento será necessário, caso seja necessário incluir os arquivos diretamente no cartão de memória.

Programação do SMA: será necessário criar um agente que ficará responsável pelas ações do buzzer no robô.

4.2. Implementação utilizando a Abordagem proposta

Nesta seção serão apresentadas as etapas necessárias para a implementação de um SMA Embarcado, utilizando a abordagem proposta neste artigo.

Componentes para construção: nesta etapa são listados os componentes necessários para a hospedagem do SMA Embarcado e habilidade para receber recursos em tempo de execução. Será necessário: 01 Raspberry Pi Zero W e 01 Cartão MicroSD 4 GB, além de um 01 Adaptador OTG e 01 hub USB. O esquema é apresentado na Figura 3a.

Programação do SMA: será necessário um SMA e um agente comunicador, responsável pela recepção de novos agentes. O código do agente é apresentado na Figura 3c.

Configuração do SO: Será necessário o download e instalação do ChonOS[4] no cartão de memória. Por ser uma distribuição de propósito específico, a mesma já possui todos os softwares necessários para embarcar um SMA. Por fim, será necessário transferir o projeto e iniciar a execução do SMA, isso pode ser feito através do ChonOS SysConfig, uma interface web para gerenciamento do sistema.

Definição de Recursos: nesta etapa são identificados os recursos necessários para atendimento do proposto no estudo de caso. Para atender ao cenário serão utilizados os seguintes *recursos*: 01 conjunto robótico 2WD com uma ponte H; 03 sensores de obstáculo digitais; e 01 sensor de luz. Além disso, no modelo proposto será necessário adicionar 01 Arduino UNO para hospedar o firmware dos recursos do SMA. O esquemático do recurso é apresentado na Figura 3b.

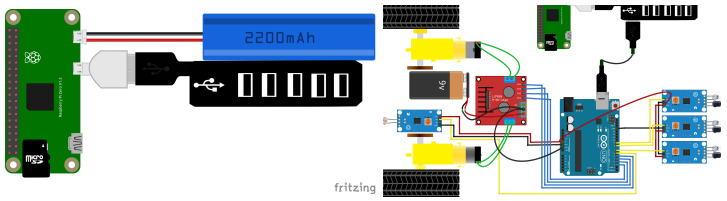
Programação do Firmware: No computador do desenvolvedor, será necessário importar o

[4] `http://chonos.sf.net/`

Javino e programar o microcontrolador Arduino para receber os comandos enviados pelo agente.

Programar o SMA Transportador: Será necessário programar e executar um SMA com o protocolo de inquilinismo no computador do desenvolvedor. Um agente será necessário para o transporte de todos os outros agentes do SMA, conforme Figura 3d. No caso apresentado, será necessário programar um agente Argo para controlar os recursos adicionados ao SMA, conforme Figura 3e.

Transferir os agentes para o SMA Embarcado: através da execução do SMA programado no computador do desenvolvedor, o agente comunicador irá transferir todos os agentes pertencentes ao SMA Transportador, para o SMA Embarcado.



(a) Dispositivo Inicial, sem recurso.

(b) Recurso adicionado

```
MAS embeddedMAS {
    infrastructure: Centralised
    agents: embeddedAgent agentArchClass Communicator;
    aslSourcePath: "asl";
}
```

```
/* Initial beliefs and rules */
embeddedAgent("4168ef73-db44-48e3-a36e-231d3ef4155a").
/* Initial goals */
!start.
/* Plans */
+!start <- .print("Iniciando SMA Embarcado"); !connect.
+!connect: embeddedAgent(UUID) <-
    .print("Iniciando conexão com a SkyNet ID=",UUID);
    .connectCN("skynet.chon.group",3273,UUID);
    .wait(60000);
    !disconnect.
+!disconnect <- .print("Encerrando conexão com a SkyNet"); .disconnectCN;
    !connect.
+waiting_for_transfer[source(X)] <-
    .print("Transfer ready for ",X);
    .sendOut(X,tell,embeddedReady);
    -waiting_for_transfer[source(X)].
```

(c) SMA Inicial.

```
MAS resourse01 {
    infrastructure: Centralised
    agents:
        agentArgo agentArchClass Argo;
        agentComm agentArchClass Communicator;
    aslSourcePath:"asl";
}
```

```
embeddedAgent("4168ef73-db44-48e3-a36e-231d3ef4155a").
commAgent01("01485bfd-683a-4e69-bd62-dd1ac32faa01").
abroad(yes).
/* Initial goals */
!start.
/* Plans */
+!start : abroad(yes) & commAgent01(UUID)<-
    .print("Connecting to ContextNet Server ID=",UUID);
    .connectCN("skynet.chon.group",3273,UUID); !send.
+!start : abroad(no)  <- .print("Nothing to do!").
+!send : abroad(yes) & embeddedReady[source(X)] <-
    .print("Transporter ready!");
    -embeddedReady[source(X)];
    +ready; !send.
+!send : abroad(yes) & ready & embeddedAgent(UUID) <-
    -ready;
    -abroad(yes); +abroad(no);
    .moveOut(UUID, inquilinism).
+!send : abroad(yes) & embeddedAgent(UUID)<-
    .print("AgentComm trying communication with EmbeddedAgent...",UUID);
    .sendOut(UUID, tell, waiting_for_transfer); .wait(2500);
    !send.
```

(d) Agente comunicador

```
/* Initial beliefs and rules */
value(30).      /*min distance*/
/* Initial goals */
!start.
/* Plans */
+!start: abroad(no)[source(agentComm)] <-
.print("I am inside - starting my task"); !conf.
+!start: abroad(yes)[source(agentComm)] <-
    .print("Waiting for transference"); .wait(5000); -abroad(yes);
    .send(agentComm, askOne, abroad(Status), Reply); -+Reply; !start.
+!start <-  .send(agentComm, askOne, abroad(Status), Reply);
            .wait(2000); -+Reply; !start.
/* Atuação quando embarcado */
+!conf: abroad(no)<- .print("Conecting with resource 01");
    .port(ttyACM0); !percebe.
+!percebe <- .percepts(open); !decide.
+!decide:lightSensor(no) & status(stopped) <-
    .print("No light - waiting"); .wait(2000); !decide.
+!decide:lightSensor(no) <- !atua.
+!decide:lightSensor(yes) <- .percepts(block); !atua.
+!decide <- .print("Error!").
```

```
+!atua: lightSensor(no)  <-
    .act(stopRightNow);
    .print("Stopping light off");
    !percebe.
+!atua: lightSensor(yes) <- !andar.
+!andar: obstFront(F) & obstRight(D) & obstLeft(E) & value(Y) & F > Y <-
    .act(goAhead);
    .print("(goAhead} Esquerda=",E," Frente=",F," Direita=",D); !percebe.
+!andar: obstFront(F) & obstRight(D) & obstLeft(E) & value(Y) & F <= Y <-
    .act(stopRightNow);
    .print("(stopRightNow) Esquerda=",E," Frente=",F," Direita=",D); !rota.
/* Decisão de rota */
+!rota: obstFront(F) & obstRight(D) & obstLeft(E) & value(Y) &  D > Y <-
    .act(turnRight);
    .print("(turnRight) Esquerda=",E," Frente=",F," Direita=",D); !percebe.
+!rota: obstFront(F) & obstRight(D) & obstLeft(E) & value(Y) & E > Y <-
    .act(turnLeft);
    .print("(turnLeft) Esquerda=",E," Frente=",F," Direita=",D); !percebe.
+!rota: obstFront(F) & obstRight(D) & obstLeft(E) & value(Y) <-
    .act(stopRightNow);
    .print("(Sem saída) Esquerda=",E," Frente=",F," Direita=",D); .wait(5000);
    !percebe.
```

(e) Agente atuador

Figura 3. Transferência de Agentes para o SMA

4.3. Atualização do SMA no modelo proposto

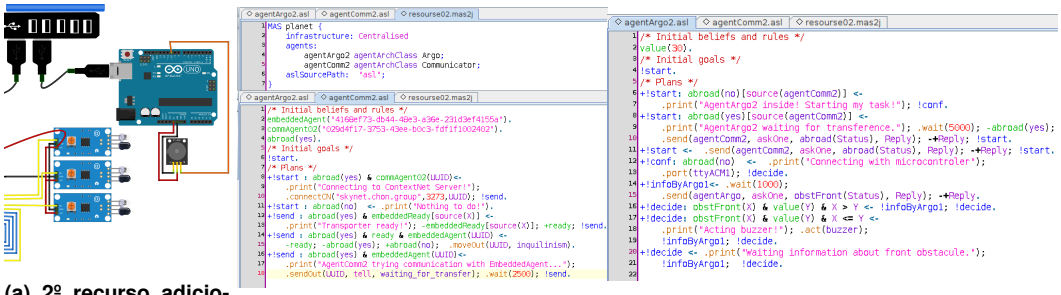
Nesta seção serão apresentadas as etapas necessários para a atualização do SMA, conforme o modelo proposto.

Definição de Recursos: nesta etapa são identificados os recursos necessários para atendimento da necessidade de atualização do SMA Embarcado. Para tal, será necessário: 01 módulo buzzer e 01 Arduino UNO para hospedar o firmware do recurso. O esquemático dos recursos é apresentado na Figura 4a.

Programação do Firmware: No computador do desenvolvedor será necessário importar o Javino e programar o microcontrolador Arduino para receber os comandos enviados pelo agente.

Programar o SMA Transportador: Será necessário programar e executar um SMA com o protocolo de inquilinismo no computador do desenvolvedor. Um agente será necessário para o transporte de todos os outros agentes do SMA, conforme Figura 4b. No caso apresentado, será necessário programar um agente Argo para controlar o novo recurso adicionado ao SMA, conforme Figura 4c.

Transferir os agentes para o SMA Embarcado: através da execução do SMA programado no computador do desenvolvedor, o agente comunicador irá transferir o novo agente necessário para controlar o novo recurso.



(a) 2º recurso adicionado

(b) Agente comunicador 2º recurso

(c) Agente atuador do 2º recurso

Figura 4. Transferência de Agentes para o SMA

5. Considerações Finais

Esse trabalho apresentou uma metodologia para adição de recursos em tempo de execução em SMA Embarcados utilizando o Jason e arquiteturas de agentes customizáveis capazes de interfacear hardware e mover agentes através de uma rede IoT. A adição de recursos permite que um SMA Embarcado possa ser atualizado e aperfeiçoado em tempo de execução, sem a necessidade de para-lo. A parada de um SMA pode acarretar algumas situações indesejadas como, por exemplo, em um domínio missão crítica. Além disso, ao adicionar um novo recurso, modificações necessárias na estrutura física do dispositivo seriam necessárias, oferecendo alguns riscos de continuidade e disponibilidade do serviço que o dispositivo estiver executando. Atualmente, uma adição de recurso acaba por obrigar o desligamento do dispositivo, limitando a capacidade de adaptação inerente a um SMA Cognitivo.

Essa discussão pode ser também direcionada para as ações de substituição e remoção de recursos em tempo de execução. Em sistemas embarcados, não é raro componentes danificados ao interagir com o mundo real, dado sua imprevisibilidade. A substituição poderia ser realizada sem prejuízo ao SMA Embarcado, se o recurso danificado for resposto por um outro de uma mesma estrutura lógica conectada a mesma

porta serial. No caso da remoção de um recurso, seja esse danificado ou intencionalmente removido, o SMA precisaria se readaptar para não perseguir intenções e objetivos que não mais podem ser alcançados dado a ausência de interfaceamento. Nesse caso, mecanismos para remoção de intenções e objetivos, ou até mesmo planos, se fazem necessários.

Sobre a composição de agentes de um SMA Embarcado para a adição de recursos, é obrigatório a existência de um agente Comunicador no SMA de origem e um outro no de destino, pois a comunicação entre SMA distintos se dá através de uma rede IoT das quais apenas esses agentes são clientes. Estes agentes são responsáveis por invocar os protocolos bio-inspirados. Neste trabalho usamos o Inquilinismo por sua atuação não destrutiva para o destino quanto para a origem. Contudo, a existência do SMA de origem após o envio dos agentes novos não é necessária. Portanto, o uso do protocolo de Mutualismo pode apresentar uma vantagem, visto que os agentes a serem enviados podem ser escolhidos. Uma outra questão, é que os agentes estão atualmente especializados por opção, ou seja, cada agente tem uma habilidade (comunicadores, comunicam e agentes Argo interfaceiam hardware). Nada impede a existência de uma arquitetura mista. Contudo, tal saída poderia sobrecarregar o agente dado o fluxo de mensagens e percepções.

A adição de recursos em tempo de execução ainda requer um esforço multidisciplinar do projetista, uma vez que esse tem que conhecer diversas áreas (eletrônica, sistemas operacionais, programação orientada a objetos e a agentes). Como trabalhos futuros, é necessário um mecanismo para o gerenciamento dinâmico de recursos em SMA Embarcados, de tal forma que, ao adicionar um novo recurso, o SMA automaticamente reconheceria o dispositivo e suas funcionalidades sem a necessidade de transferência de agentes de outros sistemas. Por exemplo, caso o robô, apresentado no estudo de caso, estivesse utilizando algum mecanismo de adição dinâmica de recursos no seu SMA Embarcado, bastaria conectá-lo ao robô e toda a habilidade necessária seria automaticamente carregada para o SMA

Referências

Alvares, L. O. and Sichman, J. S. (1997). Introdução aos sistemas multiagentes. In *Anais da XVI Jornada de Atualização em Informática (JAI) do XVII Congresso da Sociedade Brasileira de Computação*, Brasília. SBC.

Balaji, P. G. and Srinivasan, D. (2010). An Introduction to Multi-Agent Systems. In Kacprzyk, J., Srinivasan, D., and Jain, L. C., editors, *Innovations in Multi-Agent Systems and Applications - 1*, volume 310, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Studies in Computational Intelligence.

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).

de Jesus, V. S., Manoel, F. C. P., and Pantoja, C. E. (2019). Protocolo de interação entre sma embarcados bio-inspirado na relação de predatismo. In *Anais do XIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações — WESAAC 2019*, pages 95–106, Florianópolis.

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.

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

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 the 9th Software Agents, Environments and Applications School (WESAAC)*, pages 13–20, Niterói.

Lazarin, N. M., Pantoja, C. E., and Jesus, V. S. d. (2021). Um Protocolo para Comunicação entre Sistemas Multi-Agentes Embarcados. In *Proceedings of the 15th Workshop-School on Agents, Environments, and Applications (WESAAC 2021)*, pages 166–177, Rio de Janeiro.

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.

Pantoja, C., Junior, M., Lazarin, N. M., and Sichman, J. (2016). ARGO: A Customized Jason Architecture for Programming Embedded Robotic Agents. In *Fourth International Workshop on Engineering Multi Agent Systems (EMAS 2016)*, Singapore.

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.

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. *Revista de Informática Teórica e Aplicada*, 29(1):81–90.

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 Jr., 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.

Upton, E. and Halfacree, G. (2013). *Raspberry Pi User Guide*. Wiley.

Weiss, G. (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1st edition.

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