Proposta de uma IDE para desenvolvimento de SMA Embarcados

Vinicius Souza de Jesus[1], Nilson Mori Lazarin[12], Carlos Eduardo Pantoja[12], Fabian Cesar Pereira Brandão Manoel[2], Gleifer Vaz Alves[3], Gabriel Ramos[2], Jose Viterbo Filho[1]

[1]Universidade Federal Fluminense (UFF)
Niterói, RJ – Brazil

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

[3]Universidade Tecnológica Federal do Paraná (UTFPR)
Ponta Grossa, PR – Brazil

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

fabiancpbm@gmail.com, gleifer@utfpr.edu.br

gabrielttnik@gmail.com, viterbo@ic.uff.br

Abstract. *Developing an Embedded MAS is a task that requires knowledge in different areas; therefore, the architecture used in this work is divided into four layers: reasoning, serial, firmware, and hardware. Different knowledge is required at each project layer. This work seeks to raise the main solutions available and others still needed to simplify the development of these systems. Through the extension of the JaCa Framework, the centralization of the development of the firmware and reasoning layers, and finally, the use of standardized prototype models, this work presents the initial functionalities of a specialized IDE for the development of Embedded MAS.*

Resumo. *Desenvolver um SMA Embarcado é uma tarefa que exige conhecimento em diferentes áreas, com isso, a arquitetura utilizada neste trabalho é dividida em quatro camadas: reasoning, serial, firmware e hardware. Em cada camada do projeto são necessários conhecimentos distintos. São apresentadas as principais soluções disponíveis e outras que ainda são necessárias para a simplificação do desenvolvimento desses sistemas. Através da extensão do Framework JaCa, da centralização do desenvolvimento das camadas de firmware e de raciocínio, e por fim, do uso de modelos padronizados de protótipos, este trabalho apresenta as funcionalidades iniciais de uma IDE especializada para o desenvolvimento de SMA Embarcados.*

1. Introdução

O desenvolvimento de SMA embarcados envolve conceitos de diferentes áreas de conhecimento, dos quais destacamos, primeiramente, Sistemas Embarcados que são sistemas aplicados em dispositivos físicos, esses dispositivos são compostos por sensores e atuadores para interação com o ambiente físico. Esses sensores e atuadores são controlados por microcontroladores que, por sua vez, interagem com o Sistema Embarcado [Heath 2002].

Sistemas Multiagentes (SMA) são sistemas compostos por múltiplos agentes autônomos e pró-ativos, com capacidade de tomada de decisão e interação com outros agentes. Esses agentes são denominados cognitivos, visto que, possuem um ciclo de raciocínio capaz de analisar as informações percebidas no ambiente em que estão inseridos, além dos conhecimentos adquiridos com a comunicação com outros agentes. Sendo assim, esses agentes podem interagir coletivamente para a resolução de um objetivo comum ao SMA [Wooldridge 2000]. O *Belief-Desire-Intention* (BDI) é um modelo de desenvolvimento que possibilita o uso de atitudes mentais (*crenças, desejos e intenções*) em agentes cognitivos. Uma *crença* é uma informação, ou seja, aquilo que o agente sabe sobre o ambiente, si mesmo ou outros agentes. Um *desejo* é um propósito que o agente busca tornar realidade. E por fim, uma *intenção* é um compromisso que o agente assume para alcançar um desejo [Bratman 1987, Hübner et al. 2004].

Um SMA pode ser aplicado em ambientes simulados, ou em ambientes reais. Um SMA implementado em um dispositivo físico que interage, com microcontroladores para manipulação de atuadores e sensores, é denominado SMA Embarcado. A atuação de agentes cognitivos no mundo real pode ser baseada em uma arquitetura de quatro camadas: a camada superior (camada de raciocínio) implementada em um *hardware* capaz de executar o ciclo de raciocínio dos agentes; a próxima camada (camada serial) conecta a linguagem de alto nível do agente com a linguagem de baixo nível dos microcontroladores; a camada de *firmware* recebe as mensagens do agente e controla os sensores e atuadores; por fim, a camada de *hardware*, conecta os sensores e atuadores do agente com o ambiente físico [Pantoja et al. 2016, Souza de Castro et al. 2020]. Sendo assim, para o desenvolvimento de SMA Embarcado é necessário o domínio de conceitos de diversas áreas, tais como: eletrônica; programação de baixo nível; programação orientada a objetos; programação orientada a agentes.

Existem trabalhos na literatura que buscam auxiliar na embarcação de SMA como a arquitetura de agentes customizada denominada ARGO [Pantoja et al. 2016] que permite a comunicação com microcontroladores possibilitando esses agentes a controlar os sensores e atuadores de um dispositivo físico. Já o *Lego Agent* [Jensen 2010] são agentes capazes de enviar comandos para plataforma *Lego Mindstorm NXT* via *Bluetooth*. Todavia, ao optar por essas soluções, o projetista de SMA embarcado necessita utilizar diferentes IDE em conjunto para o desenvolvimento do SMA embarcado.

Portanto, o objetivo deste trabalho é apresentar as soluções já realizadas para a auxiliar no processo de simplificação do desenvolvimento de SMA Embarcados, além de levantar as soluções que ainda se fazem necessárias. Adicionalmente, é apresentada a proposta de desenvolvimento de um *Integrated Development Environment* (IDE) *web*, de forma que, o desenvolvedor de SMA Embarcado não tenha que utilizar diferentes IDE como é realizado atualmente, sendo uma IDE para programar o microcontrolador (camada de *firmware*), outra para o *middleware* (camada serial) e uma terceira para o SMA (camada de raciocínio). Com a IDE proposta, o desenvolvimento do SMA Embarcado é centralizado em uma única plataforma.

Para isso, é utilizado o *Framework Jason* [Bordini et al. 2007] possui um interpretador da linguagem orientada a agente *AgentSpeak* [Rao 1996] que é baseada no modelo BDI e permite o desenvolvimento de agentes BDI. Este *Framework* é utilizado neste trabalho por ser uma plataforma bem explorada e consolidada na literatura.

Além disso, é utilizado o *CArtAgO (Common ARTifact Infrastructure for AGents Open Environments)* [Ricci et al. 2006] é um *framework* que possibilita programar e executar ambientes virtuais para SMA baseado no meta-modelo *Agents & Artifacts* (A&A). Esses dois *Frameworks* foram estendidos para permitir a programação de agentes BDI e arterfatos capazes de interagir com dispositivos físicos (agentes ARGO e Artefatos Físicos [Manoel et al. 2020]) para a criação do *JaCa Embedded* que a IDE proposta utiliza para provê as funcionalidades de desenvolvimento da camada de raciocínio do SMA Embarcado.

Este trabalho está organizado da seguinte forma: na Seção 2 são apresentados os conceitos necessários para auxiliar no entendimento deste trabalho; na Seção 3 são apresentados os trabalhos relacionados, suas limitações e um comparativo com o proposto neste trabalho; na Seção 4 é apresentada a proposta de plataforma para desenvolvimento de SMA Embarcados; na Seção 5 são apresentados os avanços realizados no desenvolvimento da plataforma apresentada; por fim, na Seção 6 são apresentadas as considerações finais e a indicação de trabalhos futuros.

2. Referencial Teórico

Embarcar um SMA para atuar no ambiente físico que exige conhecimentos de diversas áreas, visto que, o projetista (ou a equipe) deve possuir um arcabouço de conhecimentos sobre eletrônica básica, funcionamento de sensores e atuadores, programação de baixo nível, comunicação serial, sistema operacional, programação orientada a objeto para representação do ambiente simulado e orientada a agente para implementar o raciocínio BDI. Durante a construção, diversos problemas podem ocorrer em qualquer camada da arquitetura:

- seja na camada de *hardware*, responsável por interligar os sensores e atuadores a um microcontrolador, onde um simples *jumper* desconectado ou uma solda mal executada pode atrapalhar o funcionamento;
- seja na camada de *firmware*, responsável por transformar sinais analógicos ou digitais em informação, onde erros lógicos na implementação ou problemas no versionamento de bibliotecas podem influenciar o funcionamento;
- seja na camada de interfaceamento, responsável pela representação virtual do ambiente e pela comunicação das deliberações do agente (alto nível) com o *firmware* (baixo nível), onde podem ocorrer ruídos na comunicação, excesso ou falta de percepções ou ainda perda de dados podem influenciar o funcionamento;
- por fim, seja na camada de raciocínio, responsável pela deliberação e execução de planos, onde agentes concorrentes podem gerar comportamento instáveis do protótipo.

Além dos desafios inerentes da construção de um SMA Embarcado, é necessário destacar algumas decisões de projetos ou etapas de construção do protótipo do dispositivo físico que o projetista deve se atentar:

- *Escolha dos sensores e atuadores*: Existem limitações na capacidade de processamento e armazenamento nos hardwares de baixo nível, nesta etapa é necessário o conhecimento sobre sensores digitais, sensores analógicos, atuadores e microcontroladores.

- *Planejamento e construção do protótipo*: É necessário experiência e conhecimento de eletrônica básica. Dado que sensores e atuadores normalmente necessitam de resistores, potenciômetros ou capacitores para funcionarem corretamente;
- *Deploy do firmware*: É necessário experiência com linguagens de programação de microcontroladores, pois, muitos sensores utilizam bibliotecas específicas para funcionamento. Além disso, nesta etapa é necessário definir os comandos recebidos da camada superior que serão aceitos pelo *firmware*;
- *Programação do ambiente*: É necessário experiência com linguagens de programação de alto nível, pois, é necessário programar os Artefatos Físicos do *CArtAgO* representando virtualmente o ambiente em que os agentes BDI estão inseridos;
- *Programação do agente*: É necessário experiência com linguagens de programação de agentes para definir as crenças iniciais, desejos e intenções dos agentes.
- *Configuração do Sistema Operacional*: É necessário experiência com o sistema operacional que roda no hardware onde o SMA será embarcado, pois, ele deve ser executado na inicialização do sistema.

3. Trabalhos relacionados

Diferentes trabalhos têm sido propostos na tentativa de auxiliar na simplificação no desenvolvimento de SMA Embarcados. Sendo assim, esta seção explora trabalhos na literatura de SMA Embarcados evidenciando suas respectivas limitações e fazendo um comparativo com as contribuições apresentadas por este trabalho.

O *Lego Agent* [Jensen 2010] é uma extensão da arquitetura de agentes padrão do *Jason* e apresenta ações internas do raciocínio do agente cognitivo, capazes de enviar comandos para uma plataforma *Lego Mindstorm NXT*. Este trabalho apresenta uma solução que atua na camada de *firmware*, via *Bluetooth* integrado com o *LeJOS JVM*. Entretanto, a baixa taxa de transferência para recebimento de percepções e envio de ações ao sistema, afetam a percepção do agente, tornando o SMA lento. Além disso, o raciocínio do agente não fica acoplado ao dispositivo físico, não caracterizando uma embarcação, mas sim um controle remoto. Ou seja, para esses *Lego Agent* serem considerados uma solução embarcada, esses agentes deveriam ter a possibilidade de estar situados na plataforma *Lego Mindstorm NXT* a qual estão controlando.

O *Middleware Javino* [Lazarin and Pantoja 2015], uma biblioteca de interfaceamento que possibilita a construção de agentes robóticos, apresenta um protocolo de comunicação para agentes-BDI *Jason*, permitindo o envio de comandos aos atuadores e requisição de dados aos sensores conectados à camada de *firmware* do robô. O *Javino* atua na camada serial e resolve a complexidade do interfaceamento entre o microcontrolador e o SMA. Entretanto, ainda é necessário utilizar a *Arduino IDE* para programar o microcontrolador (C++) e a Eclipse IDE para programar o ambiente simulado (Java).

O agente *ARGO* [Pantoja et al. 2016], é uma extensão da arquitetura *Jason*, que adiciona uma nova categoria de agente-BDI, capaz de filtrar as percepções do ambiente físico e dessa forma reduzir a carga de processamento na camada de raciocínio. Este agente consegue acessar diretamente a porta de comunicação serial do computador, para enviar mensagens ao microcontrolador. Este trabalho atua na camada serial, resolvendo

problemas de desempenho do ciclo de raciocínio do agente. Além disso, o trabalho amplia as opções de microcontroladores na camada de *firmware*, possibilitando a comunicação do *Arduino* (via *Javino*) ou *PIC* (via *Javic* [Guinelli and Pantoja 2016]). Entretanto, ainda é necessário utilizar a *Arduino IDE* ou o *Proteus* para programar a camada de *firmware*.

É possível perceber que todas as iniciativas aqui destacadas atuam em camadas específicas, os agentes *ARGO* e os agentes *Lego* atuam unicamente na camada de raciocínio e o *Javino* atua na camada serial. Assim, para o desenvolvimento de um SMA Embarcado, o projetista ainda deve necessariamente utilizar diferentes IDE para cobrir as demais camadas do SMA Embarcado. Neste artigo, busca-se demonstrar a viabilidade em centralizar o desenvolvimento de todas as camadas de SMA Embarcado em uma IDE *web*.

4. Proposta de Plataforma de desenvolvimento para SMA Embarcado

Nesta seção é apresentada uma proposta IDE que busca facilitar o desenvolvimento de SMA Embarcados, através da unificação da codificação das camadas de *firmware* e raciocínio em uma única plataforma de desenvolvimento *web*. Além disso, é apresentada uma padronização de modelos de protótipos para embarcação (camada física). Na Figura 1 é apresentada uma ilustração da integração do projetista com a IDE e exemplos de protótipos que podem ser construídos utilizando o modelo de padronização de prototipagem proposto.

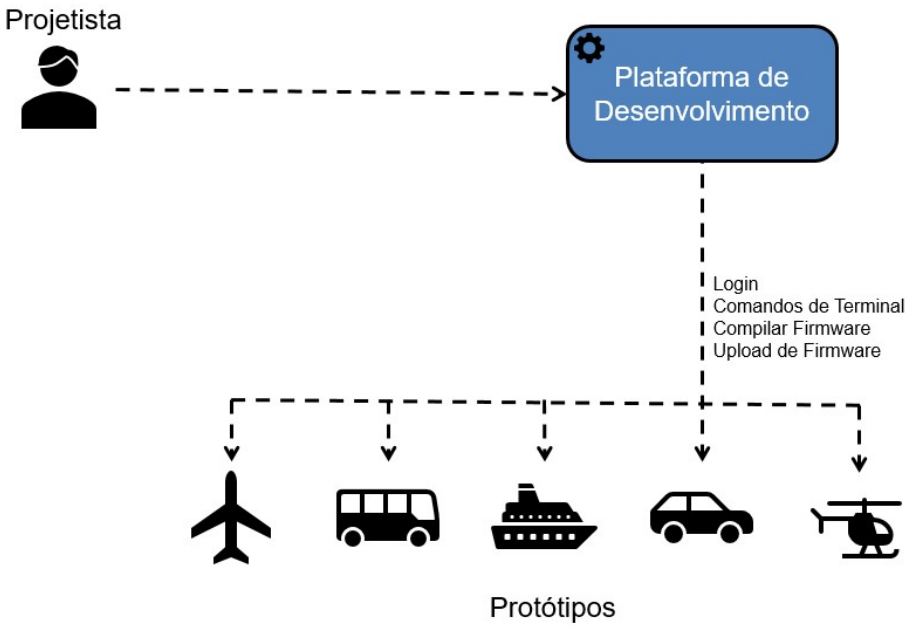


Figura 1. Ilustração da proposta de IDE e suas interações.

A plataforma de desenvolvimento para SMA Embarcado proposta é uma aplicação *web* dividida em dois módulos: um módulo de desenvolvimento e um módulo de gerenciamento.

- **Módulo de Desenvolvimento**: a aplicação deverá possibilitar a codificação de SMA e *firmware*, além de fornecer funcionalidades comuns de desenvolvimento.
- **Módulo de Gerenciamento**: a aplicação deverá se conectar diretamente com sistema operacional em execução no protótipo físico, dessa forma, o projetista poderá: enviar comandos de terminal; configurar a conectividade; executar *upload* e *deploy* de *firmware* do microcontrolador; além de realizar *upload*, iniciar ou parar um SMA.

Por fim, é necessária uma padronização dos protótipos para integração com a plataforma. Cada protótipo deve executar um sistema operacional, especializado para SMA, em um computador de placa única, para possibilitar a integração com o módulo de gerenciamento da plataforma. Além disso, a camada de *Hardware* (sensores e atuadores) de cada protótipo deve seguir uma padronização, disponibilizada pela plataforma ou criada pelo desenvolvedor.

Dessa forma, acredita-se que o desenvolvimento de SMA Embarcados será facilitado, visto que a construção de protótipos é genérica o suficiente para permitir a construção de diversos tipos de protótipos, diminuindo a complexidade de planejamento e construção de protótipos na camada de *Hardware*, visto que o projetista possui o conjunto de sensores e atuadores que podem ser utilizados para prevenir as possíveis dificuldades levantadas anteriormente na seção 2. Além disso, a utilização de um sistema operacional especializado, diminuirá a complexidade da programação do ambiente, compilação e *deploy* de *firmware* (camadas de interfaceamento e de *firmware*) e da configuração do SO, umas vez que o SO possui serviços especializados para qualquer outra interação de forma remota via protocolo de rede sem a necessidade de conexão cabeada. Ou seja, o projetista poderá dedicar-se ao desenvolvimento na camada de raciocínio do SMA Embarcado.

5. Avanços Realizados

A primeira solução é um protótipo de veículo utilizando a padronização de modelos de protótipos composto de sensores, atuadores, microcontrolador e computador de placa única pronto para hospedar um SMA, visando eliminar os principais problemas na etapa de construção do protótipo como, por exemplo, mal contato de *jumper* que interligam os sensores e atuadores com os microcontroladores, confusões no mapeamento das portas onde os sensores e atuadores devem ser conectados e entre outros. A segunda solução é a integração de todas as soluções para SMA Embarcados utilizando o *Framework JaCaMo* e criando o *JaCaMo Embedded*. Por fim, é apresentada a IDE e funcionalidades já implementadas.

5.1. Protótipo 4WD

O protótipo de veículo 4WD, apresentado na Figura 2 é composto de: um sensor de luminosidade denominado *Light Dependent Resistor* LDR; um Sensor ultrassônico HC-SR04; um conjunto de 5 *Light-Emitting Diode* (LED) para fazer a parte de iluminação do veículo; um *Buzzer*; um conjunto de motores DC com uma ponte H L298N; um microcontrolador *Arduino UNO*; um *Raspberry Pi* com placa de rede *Wi-Fi*.

O sistema operacional executando na *Raspberry Pi* é o ChonOS[1], uma distribuição GNU/Linux sem ambiente gráfico com uma interface de comunicação *Secure Shell (SSH)* com serviços especializados para a interação com o dispositivo físico. Esses serviços permitem o monitoramento do *status* da conexão de rede, possibilitam a compilação e o *deploy* do *firmware* via *arduino-cli* e execução de comandos no terminal. É importante destacar que o protótipo dispensa a conexão de monitor, teclado ou mouse. Todo o gerenciamento é feito via rede sem fio com o SSH.

Figura 2. Veículo 4WD utilizando a padronização de modelos de protótipos.

5.2. JaCa Embedded

O *JaCa Embedded* é uma extensão dos *Frameworks Jason e CArtAgO* que unifica as implementações da arquitetura customizada de agentes *ARGO* e os Artefatos Físicos já apresentados na literatura. O *JaCa Embedded* é utilizado no módulo de Desenvolvimento da IDE proposta para permitir o projetista fazer a implementação referente a camada de raciocínio do SMA Embarcado.

Tendo em vista a incorporação do *ARGO* ao *JaCa Embedded*, algumas dificuldades foram destacadas: o *ARGO* foi desenvolvido na versão 1.4.1 do Jason e utilizava a versão do 1.1 do *Middleware Javino*, nesta versão, o *Javino* utilizava o *Python* 2.7 juntamente com o PySerial 2.7 para a comunicação serial com o microcontrolador. Sendo assim, foi necessário analisar e adaptar o código desenvolvido no *Framework* Jason para a

[1]http://chonos.sf.net

versão 2.4, atualizar a versão do *Python* do *Middleware Javino* para a versão 3.9, adaptar o código da biblioteca de alto nível do *Javino* para suportar o Python 3.9 e gerar a nova versão do *Javino* que foi a versão 1.3.

Já a parte dos Artefatos Físicos utiliza o *JaCa* na versão 0.8 (uma versão recente), então não foi necessário nenhuma atualização de código nesta parte. Porém, os Artefatos Físicos também utilizam o *Middleware* Javino, então todas as alterações feitas para o *ARGO* puderam ser aproveitadas incluindo as atualizações do *Framework* Jason. A decisão de atualizar a versão do Jason na implementação dos agentes *ARGO* para a versão 2.4 foi devido ao fato dos Artefatos Físicos utilizarem essa versão e facilitar a incorporação dessas soluções. Portando, o *JaCa Embedded* atualmente utiliza a versão 0.8 do *Framework JaCa* e a versão 2.4 do *Framework* Jason.

5.2.1. Desenvolvimento da IDE

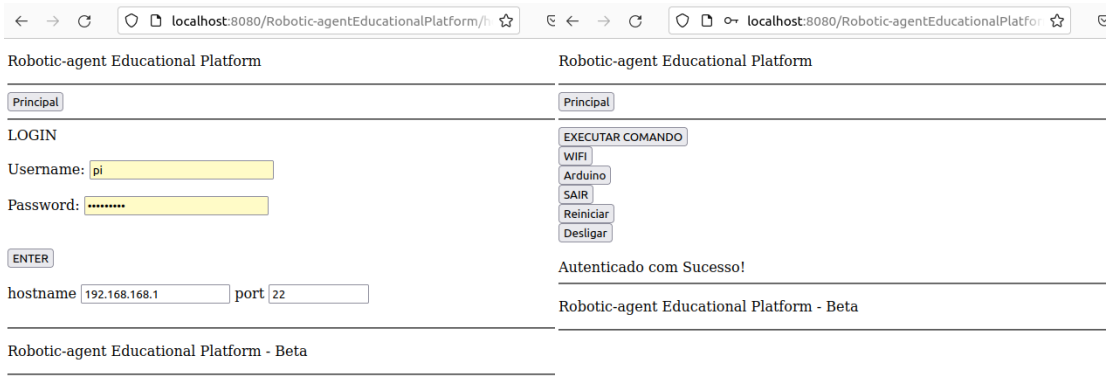
Atualmente as primeiras funcionalidades da IDE estão parcialmente implementadas, que são: fazer *login* e executar comandos no terminal do dispositivo físico; compilar, editar e fazer *deploy* do *firmware*; compilar, iniciar, parar a execução de um SMA, além de adicionar, remover e editar o código fonte dos agentes. Apenas o módulo de interação com o dispositivo físico foi finalizado. A primeira interface construída para esse módulo é a de login, nela o projetista pode inserir um *username* e uma senha de usuário válido no SO do dispositivo físico Em sequência, deve-se informar o *hostname* ou o endereço IP do dispositivo físico na rede e a porta de comunicação.

O protótipo proposto possui algumas funcionalidades que auxiliam o projetista a se conectar com o dispositivo, a primeira delas é o modo *Access Point* que o próprio dispositivo levanta uma rede *Wi-Fi* própria quando o mesmo não se encontra conectado em nenhuma rede ou caso todas as suas redes de conexão *Wi-Fi* cadastradas estejam fora de alcance. Desta maneira, o projetista pode se conectar com dispositivo utilizando o IP 192.168.168.1 e porta 22 (que são as informações padrões do dispositivo físico na rede *myRobot* levantada) e continuar o desenvolvimento do SMA Embarcado sem a necessidade de uma conexão cabeada com o dispositivo físico. Na Figura 3a mostra a interface de login em dispositivos físicos da IDE proposta.

Após se conectar no dispositivo físico, o projetista acessa o menu de funcionalidades de interação com o dispositivo. Nesta interface, estão disponíveis cada uma das funcionalidade de interação com o dispositivo desenvolvidas até o presente momento. A primeira funcionalidade diz respeito a execução de comandos no terminal; a segunda é gerenciamento de redes *Wi-Fi*, a terceira é para a interação direta com o microcontrolador, a quarta é para se desconectar do dispositivo físico atualmente conectado, a quinta é para reiniciar o SO do dispositivo físico e a sexta e última é para desligá-lo. Na Figura 3b é mostrada a interface de menu principal com as funcionalidades de interação com o dispositivo descritas.

A funcionalidade que permite executar comandos no terminal do dispositivo físico é bem simples, permite ao projetista inserir comandos para serem enviados para executar no dispositivo físico. Além disso, permite visualizar o retorno da execução do comando e, por fim, possui um mecanismo para voltar para o menu principal onde estão todas as

funcionalidades da IDE. Na Figura 4a é evidenciada a interface para execução de comandos no terminal do dispositivo com um exemple de um comando real executado e seu respectivo retorno.



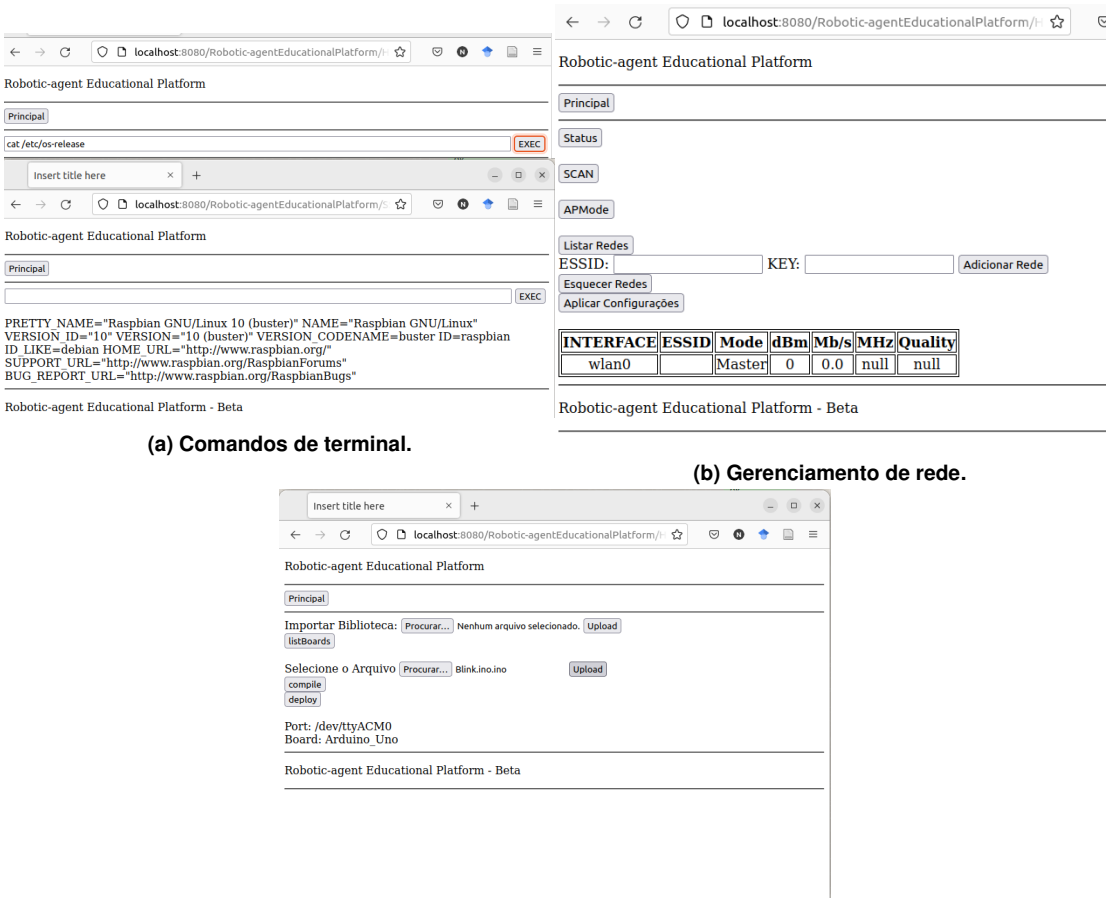
(a) Tela de login. **(b) Tela principal.**

Figura 3. Telas iniciais da IDE.

Já a interface de controle de conexões de redes *Wi-Fi*, disponibiliza ao projetista uma série de funcionalidades. A primeira é ver o estado da rede conectada atualmente, ao acionar esta funcionalidade é apresentada uma tabela com as principais informações da rede (Interface, ESSID, Modo, dBm, Mb/s, MHz, Qualidade). Caso o dispositivo se encontre em modo *Access Point*, a coluna de ESSID estará em branco e a coluna de Modo estará com o valor *Master* preenchido e as demais colunas com o valor zero ou nulo como pode ser observado na Figura 4b.

Além disso, esta interface permite escanear novas redes de *Wi-Fi* no alcance do dispositivo físico, se conectar a elas, listar as redes que o dispositivo possui gravadas e esquecer as informações de conexões. Essa interface permite também forçar que o dispositivo físico fique em modo de *Access Point* mesmo que ele esteja conectado em uma rede, ou seja, o dispositivo físico irá se desconectar da rede atual e levantará a rede *myRobot* com as configurações padrões. Por fim, após o projetista finalizar as configurações de rede desejadas, o mesmo deve confirmar as alterações para que sejam efetivamente executadas no dispositivo físico. Ne Figura 4b é mostrada a interface de controle de conexões de redes *Wi-Fi* do dispositivo.

Por fim, a funcionalidade de gerenciamento das interações com o microcontrolador do dispositivo foi desenvolvida levando em consideração inicialmente que o protótipo utilizará o microcontrolador *ATMEGA* do *Arduino*. Com isso, essa interface permite fazer a importação de bibliotecas externas, *upload* de um novo código-fonte de *firmware* bem como a compilação e o *deploy* no microcontrolador. A interface conta também com as informações da porta de comunicação do microcontrolador, o modelo da placa microcontrolada e caso o projetista faça um *deploy* é apresentado um retorno se o *deploy* foi bem-sucedido ou não. Na Figura 4c é apresentado a interface de gerenciamento do microcontrolador do dispositivo físico.



(a) Comandos de terminal.

(b) Gerenciamento de rede.

(c) Gerenciamento do *firmware*.

Figura 4. Telas da IDE

6. Considerações Finais e Trabalhos futuros

Neste trabalho foi utilizada uma arquitetura de desenvolvimento de SMA Embarcado dividida em quatro camadas: *Reasoning* camada dos SMA; *Serial* camada de interface entre o SMA e o dispositivo físico; *Firware* camada do programa do comportamento do microcontrolador; *Hardware* camada dos sensores e atuadores. Com isso, é possível observar que uma das primeiras dificuldades na embarcação de SMA é que exige o domínio de conhecimento em diferentes áreas (Eletrônica, Programação de baixo nível, Programação orientada a objetos e orientada a agentes).

Tendo em vista essa dificuldade e outras como a necessidade de utilização de diferentes IDE (normalmente uma para cada camada) e confusões no mapeamento de portas dos sensores e atuadores, este trabalho propôs soluções com o intuito de auxiliar na simplificação do desenvolvimento de SMA Embarcados. A primeira delas é uma padronização de modelos de protótipos onde é apresentado um protótipo de veiculo 4WD como exemplificação da utilidade dessa padronização. O protótipo é possui sensores, atuadores, microcontrolador, minicomputador e funcionalidades que permitem a conexão e *upload* de *firmware* remotamente. A segunda solução é o *JaCa Embedded* que é uma extensão dos *Frameworks Jason e CArtAgO* com uma adaptação e junção do agente *ARGO* e Artefatos Físicos. Por fim, é apresentada uma proposta de IDE *web* com as interfaces e

funcionalidades desenvolvidas até o momento como: Login remoto em dispositivo físico; execução de comandos no terminal do SO do dispositivo; gerenciamento das redes *Wi-Fi* e gerenciamento do *firmware* do microcontrolador (importação de bibliotecas, compilação e *deploy*).

Como continuidade imediata do trabalho, almeja-se inicialmente incluir uma tela que permita ao projetista desenvolver os agentes com o *JaCa Embedded*. Depois, disponibilizar uma tela para o desenvolvimento do código fonte do *firmware* do microcontrolador. Adicionar nessas telas de desenvolvimento de código, um verificador em tempo real de sintaxe para o auxílio no desenvolvimento. Além disso, deseja-se estilizar toda a interface gráfica da IDE. Por fim, criar uma interface para os projetistas de SMA Embarcados enviar sugestões de melhoria e avaliar o quanto a IDE está simplificando o desenvolvimento de SMA Embarcados.

Uma vez que, toda a parte de agentes leva em consideração somente o *Framework JaCaMo*, deseja-se também implementar uma extensão para o desenvolvimento de SMA Embarcados utilizando uma outra linguagem de programação de agentes, a linguagem GWENDOLEN [Dennis and Müller 2008]. GWENDOLEN é uma linguagem para programação de agentes BDI e que ainda oferece a capacidade de verificação formal dos agentes usando a ferramenta de model checking *AJPF*. Com isso, GWENDOLEN torna-se uma alternativa adequado ao *JaCaMo*, porém não possui nenhum mecanismo para a embarcação de SMA atualmente.

Referências

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

Bratman, M. E. (1987). *Intention, Plans and Practical Reasoning*. Cambridge Press.

Dennis, L. and Müller, B. (2008). Gwendolen : A BDI Language for Verifiable Agents. In *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning*, volume 9, pages 16–23. University of Aberdeen.

Guinelli, J. V. and Pantoja, C. (2016). A Middleware for Using PIC Microcontrollers and Jason Framework for Programming Multi-Agent Systems. In *Anais do Workshop de Pesquisa em Computação dos Campos Gerais WPCCG*, volume 1, Ponta Grossa.

Heath, S. (2002). *Embedded systems design*. Elsevier.

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.

Jensen, A. S. (2010). Implementing Lego Agents Using Jason.

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.

Manoel, F., Pantoja, C., Samyn, L., and Souza de Jesus, V. (2020). Physical artifacts for agents in a cyber-physical system: A case study in oil & gas scenario (eeas). In *The Thirty Second International Conference on Software Engineering and Knowledge Engineering (SEKE 2020)*, pages 55–60.

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.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W. V. and Perram, J. W., editors, *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world (MAAMAW'96)*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55, USA. Springer-Verlag.

Ricci, A., Viroli, M., and Omicini, A. (2006). Cartago: A framework for prototyping artifact-based environments in mas. In *International Workshop on Environments for Multi-Agent Systems*, pages 67–86. Springer.

Souza de Castro, L., Manoel, F., Souza de Jesus, V., Pantoja, C., Borges, A., and Vaz Alves, G. (2020). Integrando sistemas multi-agentes embarcados, simulação urbana e aplicações de iot. In *XIV Workshop Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC 2020)*.

Wooldridge, M. J. (2000). *Reasoning about rational agents*. MIT press.