

Automação em Sistemas Multiagentes: Estudo de Caso com Controle de LED Remoto

Jean A. Fagundes¹, Marcelo S. Santos²

¹Departamento de Sistemas de Informação – Centro Federal de Educação Tecnológica
Celso Suckow da Fonseca (CEFET-RJ) – Campus Maria da Graça

{jean.assis,marcelo.sousa}@aluno.cefet-rj.br

Abstract. *This article, developed in the Sistemas Especialistas course of the Bachelor's Degree in Information Systems at CEFET-RJ, explores the implementation of a multi-agent system focused on the automation and control of a simple hardware device - a LED. Through the presented codes, we demonstrate that each agent plays a crucial role, from the physical interface with the device to the execution of automated and programmed tasks. The detailed analysis of each agent provides a deep understanding of how complex tasks can be managed and executed through collaboration and communication among various agents in an integrated system.*

Resumo. *O presente artigo, elaborado na de Sistemas Especialistas do Bacharelado em Sistemas da Informação do CEFET-RJ, explora a implementação de um sistema multiagentes focado na automação e controle de um dispositivo de hardware simples - um LED. Através dos códigos apresentados, demonstramos que cada agente desempenha um papel crucial, desde a interface física com o dispositivo até a execução de tarefas automatizadas e programadas. A análise detalhada de cada agente fornece uma compreensão aprofundada de como as tarefas complexas podem ser gerenciadas e executadas através da colaboração e comunicação entre agentes diversos em um sistema integrado.*

1. Introdução

Este artigo explora a aplicação prática e teórica de sistemas multiagentes, utilizando o controle de LEDs por forma remota, como um estudo de caso focado. A escolha de um dispositivo simples, como um LED, permite a exploração detalhada das capacidades de automação em um contexto controlado e compreensível. O controle de dispositivos a distância, através de sistemas multiagentes, é uma área que apresenta significativas oportunidades de inovação e transformação no modo como interagimos com dispositivos e sistemas do cotidiano. A análise detalhada de cada agente, suas funções, e a interação dentro do sistema, revela os desafios enfrentados em sistemas de automação distribuída. A automação remota serve não apenas como um exemplo prático, mas também como um modelo para compreender questões mais amplas relacionadas à sincronização, comunicação em rede e integração de sistemas em ambientes de Internet das Coisas (IoT). Buscamos oferecer a compreensão sobre o papel dos sistemas multiagentes na automação, ilustrando como a colaboração entre agentes pode ser efetivamente empregada para controlar dispositivos remotos.

2. Metodologia

O procedimento envolve inicialmente a condução de uma análise detalhada dos requisitos do projeto. Segue-se a escolha da plataforma Arduino UNO e Javino como os principais recursos tecnológicos. Posteriormente, há a configuração do ambiente de desenvolvimento, seguida pelo desenvolvimento de código para a integração de multiagentes. O passo subsequente abrange a realização de testes, com o objetivo de verificar e aprimorar a eficácia do sistema. Por fim, uma análise criteriosa dos resultados obtidos é efetuada para discernir as funcionalidades, limitações e possíveis aplicações futuras do sistema desenvolvido.

3. Discussão

3.1. O que é um agente

Segundo [Lazarin and Pantoja 2022] um agente, no contexto computacional, é definido como uma entidade capaz de exercer autonomia decisória, fundamentando-se em princípios pré-determinados que incluem intenções, planos e crenças. Distingue-se de sistemas computacionais tradicionais em diversos aspectos. Primeiramente, caracteriza-se pela independência operacional, isto é, sua funcionalidade ou existência não é condicionada pela presença ou ação de outro agente. Ademais, é proativo, agindo no ambiente por iniciativa própria para cumprir seus objetivos. Em termos de racionalidade, o agente elabora estratégias de ação visando atingir metas específicas. Por fim, destaca-se pela capacidade de adaptação, na qual, diante de contratempos ou falhas, busca implementar planos alternativos para superar obstáculos..

3.2. Sistema Multiagente

O ciclo de vida de um Sistema Multiagente (SMA) é composto por duas fases distintas: concepção e resolução[Sichman and Demazeau 1995]. Na fase de concepção, estabelece-se de forma ampla as estratégias para a organização de um conjunto de agentes visando um objetivo específico. Por outro lado, na etapa de resolução, com um objetivo claramente delineado, determina-se a metodologia mais eficaz para que o grupo de agentes possa abordar e resolver o problema em questão.

3.3. Agentes cognitivos em BDI

O modelo *Belief, Desire e Intention* (BDI), com sua origem no pensamento filosófico de Michael Bratman sobre a teoria da ação racional humana, é uma arquitetura cognitiva influente no campo do raciocínio prático. Inspirado no raciocínio humano, o BDI é estruturado em três componentes principais. O aspecto filosófico, que constitui a base teórica do modelo, é complementado por uma arquitetura de software flexível, que não prescreve uma implementação específica e permite diversas abordagens. Além disso, o modelo incorpora um componente lógico, fornecendo ferramentas essenciais para orientar as decisões dos agentes. Juntos, esses elementos capacitam os agentes a operar eficazmente em ambientes complexos e imprevisíveis, seja de forma isolada ou em interação com outros agentes.[Wooldridge 2003]

3.4. AgentSpeak

AgentSpeak(L) é uma linguagem de programação que representa uma extensão da programação em lógica, oferecendo um framework para a programação de agentes seguindo o modelo BDI, permitindo a abstração necessária para a programação de agentes cognitivos. Sendo o agente é caracterizado por um conjunto de crenças, que constituem a base inicial de crenças do agente. Estas crenças são compostas por fórmulas atômicas básicas de primeira ordem. Adicionalmente, o agente possui um conjunto de planos que compõem sua biblioteca de planos, permitindo-lhe a execução de ações e reações baseadas em suas crenças e objetivos [Bordini and Hübner 2006].

3.5. Jason

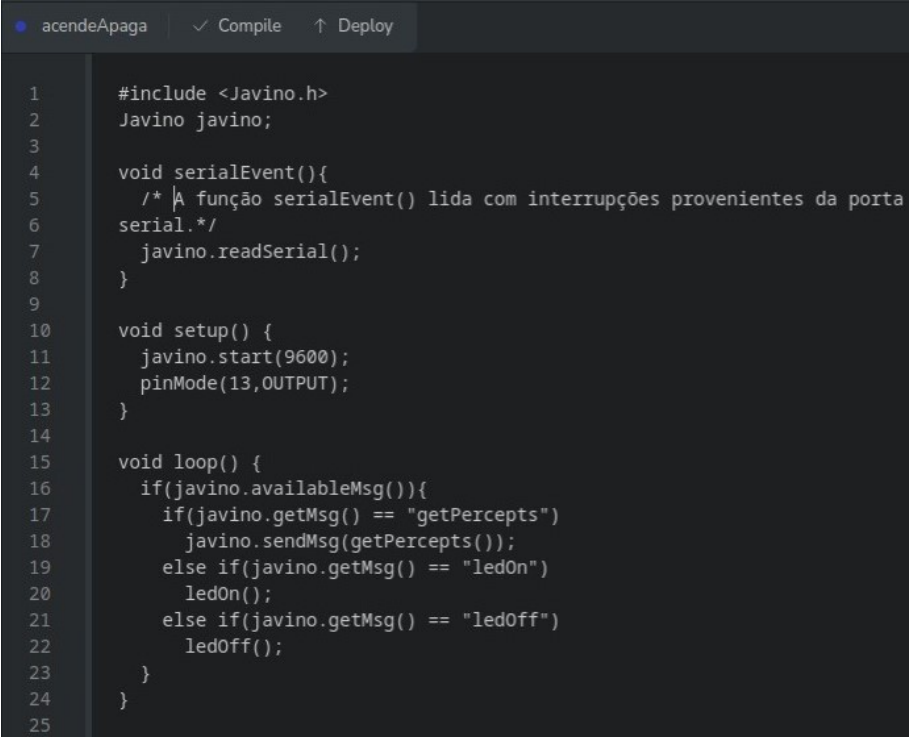
O framework Jason, conforme descrito por [Pantoja et al. 2016], é uma linguagem de programação voltada para agentes, integrando um interpretador em Java para AgentSpeak, focado no desenvolvimento de agentes inteligentes cognitivos baseados no modelo BDI. Esses agentes são projetados com um ciclo de raciocínio distinto, capaz de perceber o ambiente simulado, atualizar suas crenças, selecionar e implementar planos conforme as condições vigentes, e interpretar informações provenientes de outros agentes. No entanto, para a efetiva interação com as variáveis de ambientes reais, é essencial a integração com a arquitetura ARGO. Isto se deve ao fato de que a ferramenta, em sua forma original, não possui a capacidade de empregar sensores e atuadores para tomar decisões em tempo real dentro de seu ciclo de raciocínio.

4. Implementação do código

4.1. Introdução

Este capítulo examina a implementação e interação de um sistema multiagentes, utilizando uma série de exemplos de código para ilustrar seu funcionamento. Cada agente desempenha um papel único em um cenário de controle e comunicação de dispositivos, especificamente focado no controle de um LED via comandos serial e de rede.

4.2. Código principal

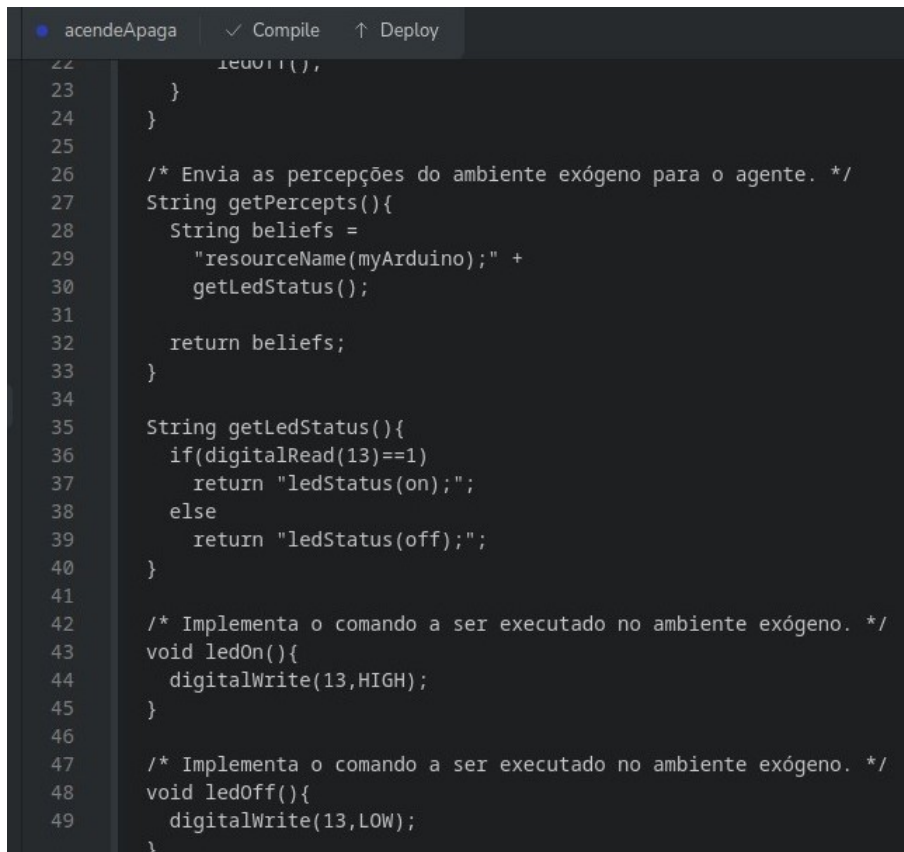
The image shows a screenshot of an IDE window titled 'acendeApaga'. The window has tabs for 'Compile' and 'Deploy'. The code is written in C++ and is as follows:

```
1  #include <Javino.h>
2  Javino javino;
3
4  void serialEvent(){
5      /* A função serialEvent() lida com interrupções provenientes da porta
6      serial.*/
7      javino.readSerial();
8  }
9
10 void setup() {
11     javino.start(9600);
12     pinMode(13,OUTPUT);
13 }
14
15 void loop() {
16     if(javino.availableMsg()){
17         if(javino.getMsg() == "getPercepts")
18             javino.sendMsg(getPercepts());
19         else if(javino.getMsg() == "ledOn")
20             ledOn();
21         else if(javino.getMsg() == "ledOff")
22             ledOff();
23     }
24 }
25
```

Figure 1. Código principal

Basicamente se inicia o código com a introdução da biblioteca "Javin" que facilita a comunicação entre o Arduino e os demais dispositivos. Através das funções "serialEvent" e "setup", inicia a comunicação serial através da leitura e processamento dos dados recebidos. A função "Loop" é a principal do programa, sendo executado repetidamente. Ao verificar se há uma mensagem disponível via Javino, é definida uma das três possibilidades ("getPercepts", "ledOn", "ledOff") e executa a ação correspondente.

A função "getPercepts" retorna uma string representando as "percepções" do Arduino, incluindo o status atual do LED verificado pela função "getLedStatus". Finalizando com as funções que controlam o estado do LED. Sendo que "ledOn()" liga o LED (definindo o pino 13 para HIGH), e "ledOff()" desliga o LED (definindo o pino 13 para LOW).



```

22     ledOff(),
23     }
24 }
25
26 /* Envia as percepções do ambiente exógeno para o agente. */
27 String getPercepts(){
28     String beliefs =
29         "resourceName(myArduino);" +
30         getLedStatus();
31
32     return beliefs;
33 }
34
35 String getLedStatus(){
36     if(digitalRead(13)==1)
37         return "ledStatus(on)";
38     else
39         return "ledStatus(off)";
40 }
41
42 /* Implementa o comando a ser executado no ambiente exógeno. */
43 void ledOn(){
44     digitalWrite(13,HIGH);
45 }
46
47 /* Implementa o comando a ser executado no ambiente exógeno. */
48 void ledOff(){
49     digitalWrite(13,LOW);
50 }

```

Figure 2. Código principal (continuação)

4.3. Agente Argo: Interface com o Hardware

O agente Argo serve como um intermediário entre o mundo físico (um LED conectado a uma placa Arduino) e o ambiente de agentes. Utilizando a biblioteca Javino, ele interpreta comandos recebidos através da porta serial para controlar o estado do LED. O código do Argo demonstra a capacidade de um agente inteligente de interagir diretamente com hardware e responder a comandos externos.

No código, apresentado na figura 3, é definida a porta serial usada para comunicação. Logo, se define intenção inicial "start", que configura e inicia a comunicação pela porta serial. Ela imprime uma mensagem, define a porta serial, define limite (500), e se abre para percepção.

Já "acendeLed" é adotada imprime uma mensagem, executa a ação "ledOn", e posteriormente remove a intenção "acendeLed" da lista de intenções do agente.

De forma similar o plano "apagaLed" imprime uma mensagem, executa a ação "ledOff" para desligar o LED, e remove a intenção "apagaLed".

```

1      /* Configuração da porta serial do arduino */
2      serialPort(ttyUSB0).
3
4      /* Cria intenção para começar o programa. */
5      !start.
6
7      /* Iniciando o plano para perceber o ambiente */
8      +!start:
9      serialPort(Port) <-
10         .print("Conectando...");
11         .port(Port);
12         .limit(500);
13         .percepts(open).
14
15      /* Plano para acender o Led */
16      +!acendeLed <-
17         .print("Ligando o Led do Arduino...");
18         .act(ledOn);
19         -acendeLed.
20
21      /* Plano para acender o Led */
22      +!apagaLed <-
23         .print("Desligando o Led do Arduino...");
24         .act(ledOff);
25         -apagaLed.
26
27      /* Ao fim dos planos coloca-se para que o agente esqueça a crença recebida.*/
28      /* Dessa forma não haverá atrito ao receber uma nova crença.*/
29

```

Figure 3. Argo

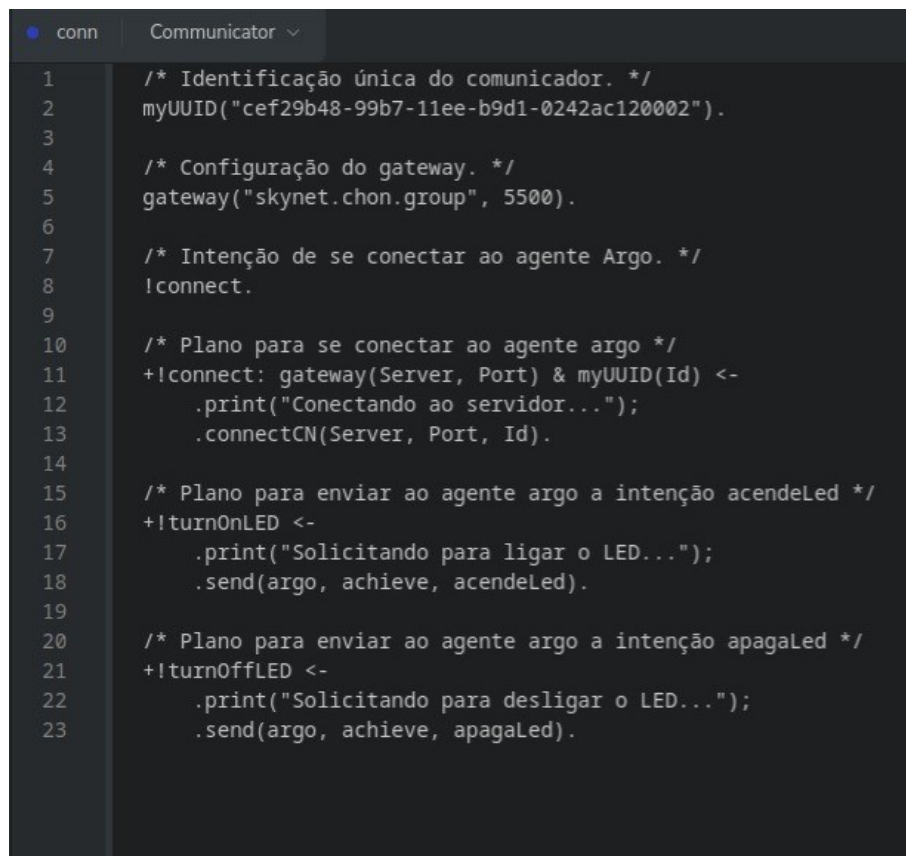
4.4. Agente Conn: Estabelecimento de Conexão

O agente Conn atua como um ponto de conexão em um sistema distribuído. Ele é responsável por estabelecer uma conexão com o agente Argo e enviar comandos para controlar o LED. Este agente exemplifica a comunicação em rede e a transmissão de intenções específicas em um sistema de agentes.

No código, apresentado na figura 4, se define um identificador único (UUID) para o agente "conn". Define o endereço do gateway (servidor) e a porta para a comunicação. O agente "conn" usará esses detalhes para se conectar ao servidor.

Através do plano que é ativado quando a intenção !connect é adotada, se utiliza as informações de gateway e UUID para se conectar a um servidor.

Por meio do plano ativado pela intenção "!turnOnLED", se envia uma mensagem para o agente "argo" com a intenção de ativar a ação acendeLed (ligando o LED).

A screenshot of a code editor window. The title bar shows a blue dot, the name 'conn', and a dropdown menu labeled 'Communicator'. The code is written in Prolog and is numbered from 1 to 23. It includes comments in Portuguese and Prolog code for setting a UUID, configuring a gateway, connecting to an agent, and sending commands to turn an LED on and off.

```
1      /* Identificação única do comunicador. */
2      myUUID("cef29b48-99b7-11ee-b9d1-0242ac120002").
3
4      /* Configuração do gateway. */
5      gateway("skynet.chon.group", 5500).
6
7      /* Intenção de se conectar ao agente Argo. */
8      !connect.
9
10     /* Plano para se conectar ao agente argo */
11     +!connect: gateway(Server, Port) & myUUID(Id) <-
12         .print("Conectando ao servidor...");
13         .connectCN(Server, Port, Id).
14
15     /* Plano para enviar ao agente argo a intenção acendeLed */
16     +!turnOnLED <-
17         .print("Solicitando para ligar o LED...");
18         .send(argo, achieve, acendeLed).
19
20     /* Plano para enviar ao agente argo a intenção apagaLed */
21     +!turnOffLED <-
22         .print("Solicitando para desligar o LED...");
23         .send(argo, achieve, apagaLed).
```

Figure 4. Comunicador

4.5. Agente Externo: Controle Remoto e Automação

O agente externo demonstra a automação em um nível superior. Ele periodicamente envia comandos para o agente Conn para alternar o estado do LED, criando um loop contínuo de controle. Este agente ilustra como os agentes podem ser programados para realizar tarefas repetitivas e interagir com outros agentes em um sistema distribuído.

No código, apresentado na figura 5, se define dois UUIDs: um para o próprio agente ("myUUID") e outro para o agente "conn" ("connUUID"). Esses identificadores são usados para estabelecer comunicações únicas entre agentes.

São definidos o endereço do servidor (gateway) e a porta para comunicação, semelhante aos exemplos anteriores.

O plano ativado pela intenção "!connect" imprime uma mensagem indicando a tentativa de conexão, estabelece a conexão com o servidor usando as informações de gateway e UUID, e então cria a intenção "!enviaraoconn".

Já o plano ativado pela intenção "!enviaraoconn" envia um comando para ligar o LED (turnOnLED) para o agente "conn", espera 3 segundos (3000 milissegundos), envia um comando para desligar o LED (turnOffLED), espera mais 3 segundos, e então reinicia o plano criando novamente a intenção "!enviaraoconn". Isso cria um loop contínuo de alternar o estado do LED.

```

1  /* Identificação única do comunicador e do comunicador externo. */
2  myUUID ("80d9c5b3-5327-4836-b722-7481061affef").
3  connUUID("cef29b48-99b7-11ee-b9d1-0242ac120002").
4
5  /* Configuração do gateway. */
6  gateway("skynet.chon.group", 5500).
7
8  /* Cria intenção de se conectar. */
9  /*!connect.*/
10
11 /* Plano para conectar, como já existe a intenção, o plano é iniciado. */
12 +!connect: gateway(Server, Port) & myUUID(Id) <-
13     .print("Conectando ao servidor ", Server, ":", Port);
14     .connectCN(Server, Port, Id);
15     !enviaraoconn.
16
17 /* No plano anterior é estabelecido a intenção. Portanto, o plano é iniciado */
18 /* Como o plano termina com a intenção novamente, é criado um loop. */
19 +!enviaraoconn: connUUID(uuid) <-
20     .print("Enviando um On para o conn!");
21     .sendOut(uuid, achieve, turnOnLED);
22     .wait(3000);
23     .print("Enviando um Off para o conn!");
24     .sendOut(uuid, achieve, turnOffLED);
25     .wait(3000);
26     !enviaraoconn.
27

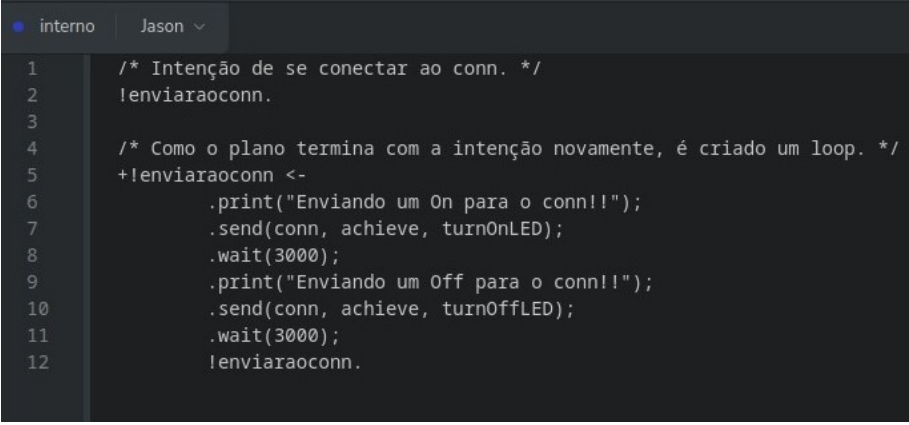
```

Figure 5. Agente externo

4.6. Agente Interno: Implementação de Loop de Feedback

Semelhante ao agente externo, o agente interno é projetado para enviar comandos periódicos ao agente Conn. A diferença chave é que este agente opera dentro do mesmo ambiente ou framework dos outros agentes, destacando a flexibilidade e a modularidade de um sistema de agentes inteligentes.

Como visto na figura 6 sua ativação se dá pela intenção ”!enviaraoconn”. Que imprime uma mensagem indicando que está enviando um comando para ligar o LED ao agente ”conn”. Envia um comando turnOnLED ao agente ”conn” com a ação achieve, indicando que deseja que o agente ”conn” realize a ação de ligar o LED. Espera por 3000 milissegundos (3 segundos). Imprime uma mensagem indicando que está enviando um comando para desligar o LED ao agente ”conn”. Envia um comando turnOffLED ao agente ”conn” com a ação achieve, para que o agente ”conn” desligue o LED. Espera novamente por 3000 milissegundos. Reinicia o plano ao criar novamente a intenção !enviaraoconn. Este ciclo cria um loop contínuo onde o agente ”interno” alterna entre enviar comandos para ligar e desligar o LED ao agente ”conn”. O uso do .wait(3000) garante que haja um intervalo de 3 segundos entre cada ação.

A screenshot of a code editor with a dark theme. At the top, there are two tabs: 'interno' (selected) and 'Jason'. The code is written in Prolog and consists of 12 lines. Line 1 is a comment: '/* Intenção de se conectar ao conn. */'. Line 2 is a goal: '!enviaraoconn.'. Line 3 is an empty line. Line 4 is a comment: '/* Como o plano termina com a intenção novamente, é criado um loop. */'. Line 5 is a goal with a trailing dash: '+!enviaraoconn <-'. Lines 6 through 12 form a loop body. Line 6: '.print("Enviando um On para o conn!!");'. Line 7: '.send(conn, achieve, turnOnLED);'. Line 8: '.wait(3000);'. Line 9: '.print("Enviando um Off para o conn!!");'. Line 10: '.send(conn, achieve, turnOffLED);'. Line 11: '.wait(3000);'. Line 12: '!enviaraoconn.'. The code is indented for the loop body.

```
1  /* Intenção de se conectar ao conn. */
2  !enviaraoconn.
3
4  /* Como o plano termina com a intenção novamente, é criado um loop. */
5  +!enviaraoconn <-
6      .print("Enviando um On para o conn!!");
7      .send(conn, achieve, turnOnLED);
8      .wait(3000);
9      .print("Enviando um Off para o conn!!");
10     .send(conn, achieve, turnOffLED);
11     .wait(3000);
12     !enviaraoconn.
```

Figure 6. Agente interno

4.7. Síntese

A interação entre esses agentes destaca a versatilidade dos sistemas multiagentes em ambientes distribuídos. Cada agente desempenha um papel específico, desde a interface direta com o hardware até a execução de tarefas automatizadas e programadas. Esta configuração demonstra a eficácia dos agentes inteligentes em realizar tarefas complexas através da colaboração e comunicação.

5. Conclusão

A interação entre os agentes em sistemas multiagentes destaca sua versatilidade em ambientes distribuídos. Cada agente, desempenhando um papel específico, contribui para a eficiência e eficácia do sistema, variando da interface direta com o hardware à execução de tarefas automatizadas e programadas. Esta configuração evidencia a habilidade dos agentes inteligentes em realizar tarefas complexas por meio de colaboração e comunicação. A análise dos códigos fornecidos revelou como eles integram o sistema, realçando o potencial dos multiagentes na automação e controle remoto.

Nas figuras 7 e 8 podemos observar o resultado prático do projeto.

As implicações deste estudo vão além do controle simples de um LED, abrangendo desde a automação industrial até sistemas domésticos inteligentes e aplicações em Internet das Coisas (IoT). A autonomia dos agentes inteligentes, sua capacidade de adaptação a novas informações e colaboração mútua são essenciais para o desenvolvimento de sistemas mais complexos e eficientes.

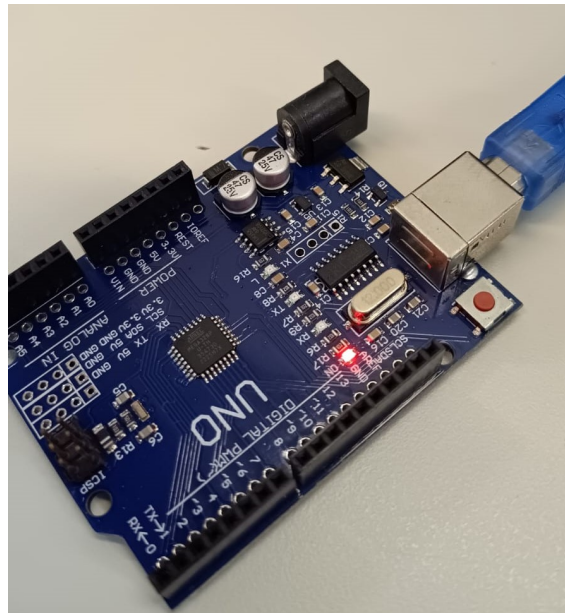


Figure 7. Implementação: LED desligado

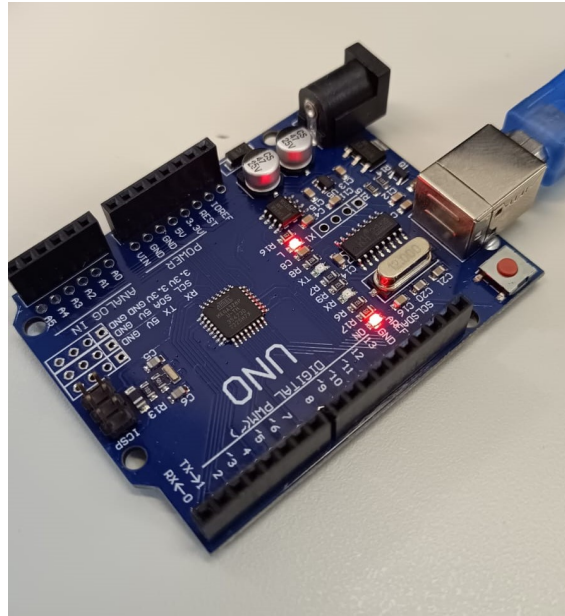


Figure 8. Implementação: LED ligado

6. References

References

- [Bordini and Hübner 2006] Bordini, R. H. and Hübner, J. F. (2006). Bdi agent programming in agentspeak using jason. In Toni, F. and Torroni, P., editors, *Computational Logic in Multi-Agent Systems*, pages 143–164, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Lazarin and Pantoja 2022] Lazarin, N. M. and Pantoja, C. (2022). Desenvolvimento de sistemas multiagentes embarcados.
- [Pantoja et al. 2016] Pantoja, C. E., de Jesus, V. S., and Filho, J. V. (2016). Aplicando sistemas multi-agentes ubíquos em um modelo de smart home usando o framework jason. In *II Workshop de Pesquisa e Desenvolvimento em Inteligência Artificial, Inteligência Coletiva e Ciência de Dados*. [S. l.: sn].
- [Sichman and Demazeau 1995] Sichman, J. S. and Demazeau, Y. (1995). Exploiting social reasoning to enhance adaptation in open multi-agent systems. pages 253–263.
- [Wooldridge 2003] Wooldridge, M. (2003). *Reasoning about rational agents*. MIT press.