

Introdução a Programação de Agentes Robóticos Usando Jason e ARGO

Carlos Eduardo Pantoja ^{1,2}

- 1. Centro Federal de Educação Tecnológica (CEFET/RJ), Brasil
- 2. Universidade Federal Fluminense (UFF), Brasil

IV Semana de Informática e Segurança do Trabalho (SIST) do IFC Fraiburgo
6 Maio 2016



INSTITUTO FEDERAL
CATARINENSE
Câmpus Fraiburgo



OUTLINE

1. Introdução
2. A Arquitetura Robótica para SMA
3. Jason Framework
4. ARGO for Jason
5. Conclusão
6. Referências Bibliográficas

OUTLINE

1. Introdução

2. A Arquitetura Robótica para SMA

3. Jason Framework

4. ARGO for Jason

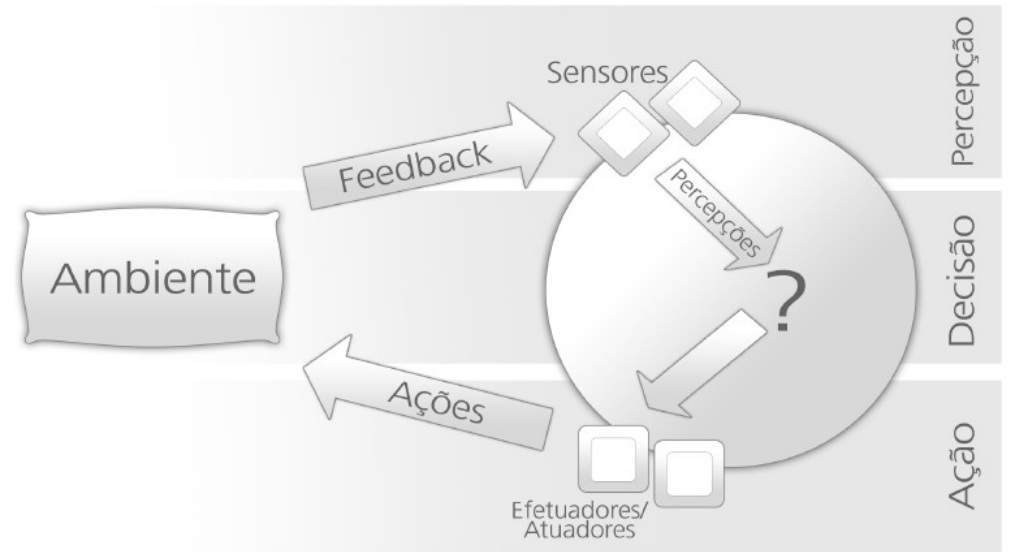
5. Conclusão

6. Referências Bibliográficas

1. INTRODUÇÃO: AGENTE

Agente

Conforme [Wooldridge, 2000], agentes são componentes **autônomos** e **cognitivos**, originados da *inteligência artificial*, situados em um **ambiente** e possuem uma biblioteca de **planos** com possíveis **ações** em resposta aos estímulos **percebidos**, com a finalidade de atingir seus **objetivos** de projeto e modificar o ambiente em que estão inseridos.



1. INTRODUÇÃO: SMA

Sistemas Multi-Agentes (SMA)

Um SMA contém um quantitativo de agentes que se **comunicam** entre si e podem **agir** em determinado **ambiente**. Diferentes agentes possuem esferas de influência onde terão controle sobre o que será **percebido** do ambiente e que podem coincidir em alguns casos.

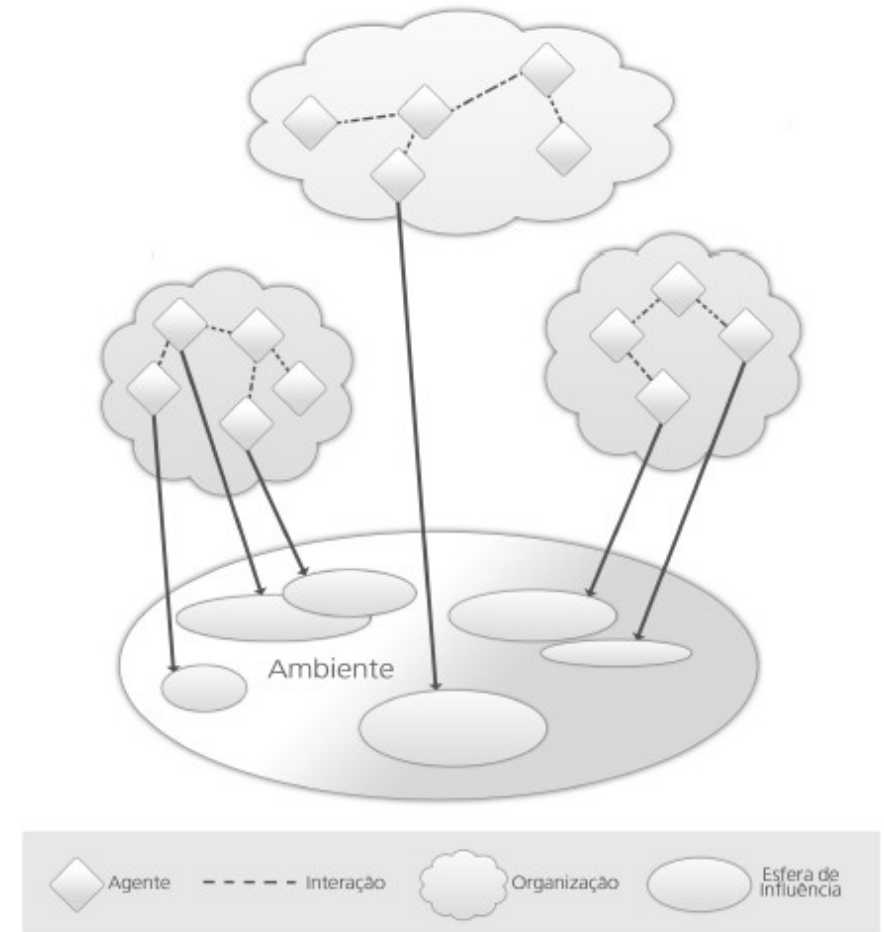
Os agentes ainda podem estar agrupados em **organizações** com a finalidade de atingir **objetivos** e metas comuns. [Wooldridge, 2009].

1. INTRODUÇÃO: SMA

Visão Tradicional de um SMA

Conforme [Wooldridge, 2009], a abordagem SMA permite a modelagem desde sistemas simples a **complexos** e são usados em uma variedade de aplicações como indústria:

1. Gestão da Informação
2. Internet
3. Transportes
4. Telecomunicações
5. Medicina
- 6. Robótica**
7. Entretenimento

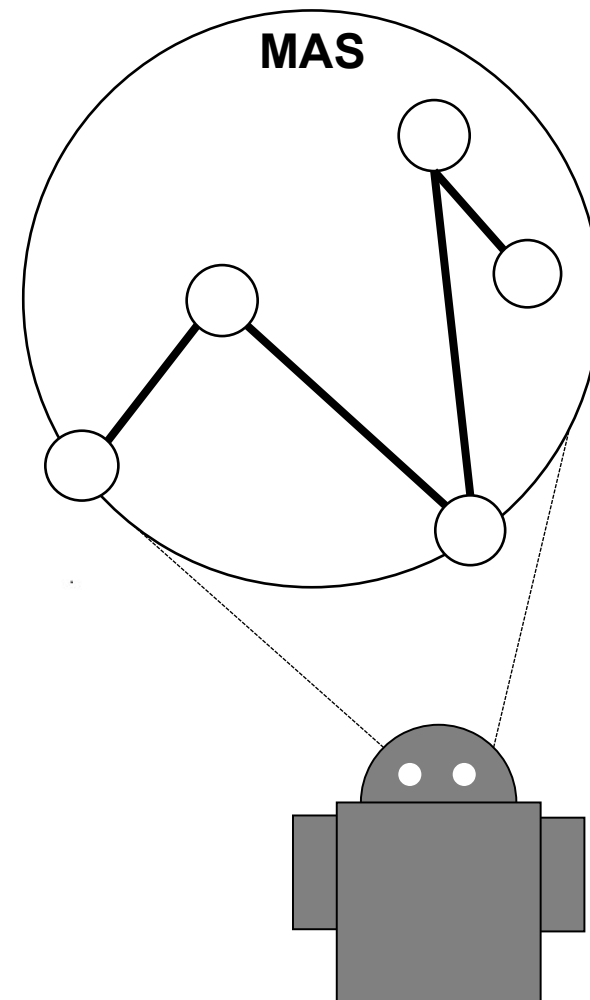


1. INTRODUÇÃO: AGENTE ROBÓTICO

Robô

É um agente **físico** que possui [Mataric, 2007]:

1. Hardware
2. Sensores e Atuadores
3. Software (raciocínio)
4. Middleware



1. INTRODUÇÃO: BDI

Modelo Belief-Desire-Intention (BDI)

O BDI se refere ao uso de programas de computadores com analogias a **crenças** (beliefs), **desejos** (desires) e **intenções** (intentions). A definição de cada uma é descrita como se segue [Bordini et al., 2007]:

- Crenças são informações que o agente tem sobre o mundo.
- Desejos são todas as possibilidades de estados de negócio que o agente deve querer atingir.
 - ✓ Porém, ter um desejo não significa que o agente irá atuar sobre ele, mas este é uma potencial influência nas ações do agente.
- Intenções são todos os estados de negócios em que o agente decidiu trabalhar.

1. INTRODUÇÃO: LINGUAGENS

Existem diversas linguagens e plataformas que implementam o conceito de **BDI**:

1. PRS [Bratman, 1987]
2. JAM [Huber, 1999]
3. dMARS [D'Inverno et al., 1998]
4. JACK [Winikoff, 2005]
5. **JASON [Bordini et al., 2007]**
6. JADE/JADEX [Bellifemine et al., 2007]

1. INTRODUÇÃO: ARGO

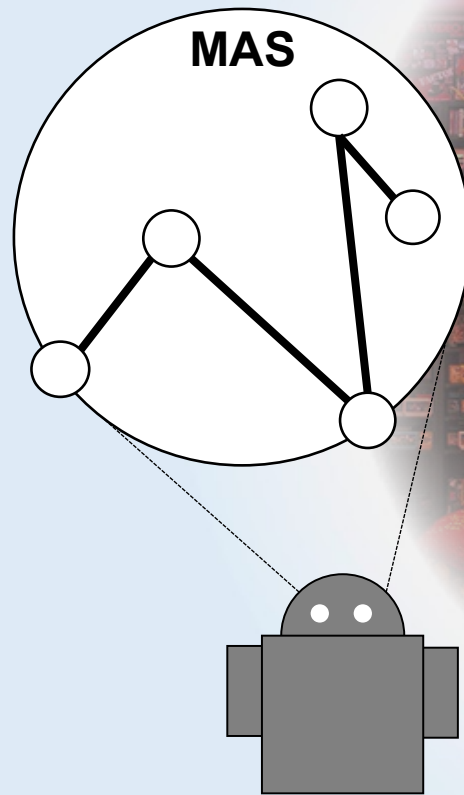
ARGO [Pantoja et al., 2016]: Uma arquitetura customizada do Jason para programação de **Agentes Robóticos** usando placas microcontroladas (**Arduino**):

- **Javino** [Lazarin e Pantoja, 2015]
 - ✓ middleware para comunicação entre controladores e software de alto nível com detecção de erro.
- **Filtros de Percepções** [Stabile Jr e Sichman, 2015]
 - ✓ Filtros de percepção reduzem a quantidade de informação percebida pelo agente em tempo de execução.

1. INTRODUÇÃO: FILTROS



1. INTRODUÇÃO: FILTROS



1. INTRODUÇÃO: OBJETIVOS

Objetivo Principal

- Criar sistemas multi-agentes utilizando uma **plataforma cognitiva** baseada em Java e **AgentSpeak**: Jason Framework.
- Criar agentes robóticos utilizando a arquitetura customizada **ARGO**

Objetivo Secundário

- Firmar uma parceria extensionista institucional entre o Projeto Turing do **CEFET/RJ** e o **IFC Fraiburgo**.
- permitir à equipe do IFC desenvolver projetos científicos para participar da **Mostra Nacional de Robótica de 2016**.

OUTLINE

1. Introdução

2. A Arquitetura Robótica para SMA

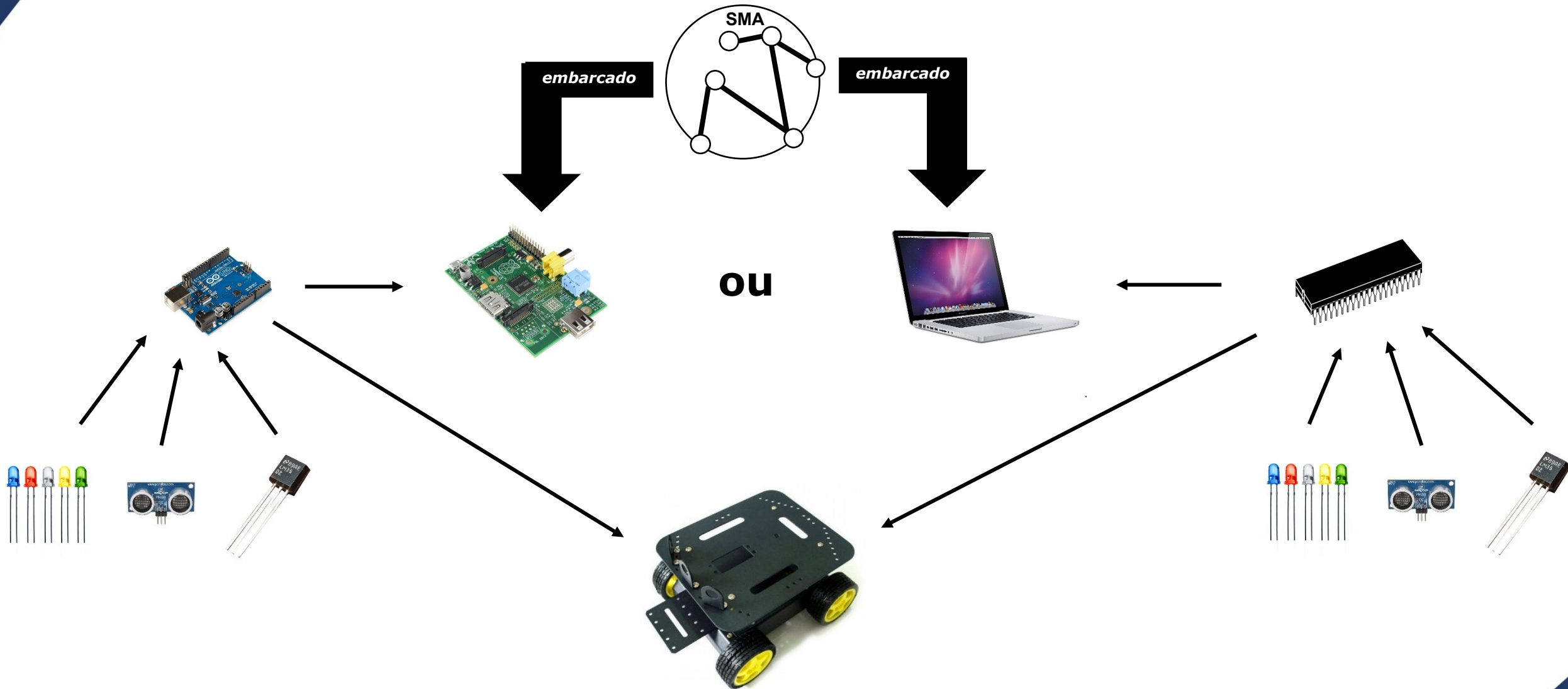
3. Jason Framework

4. ARGO for Jason

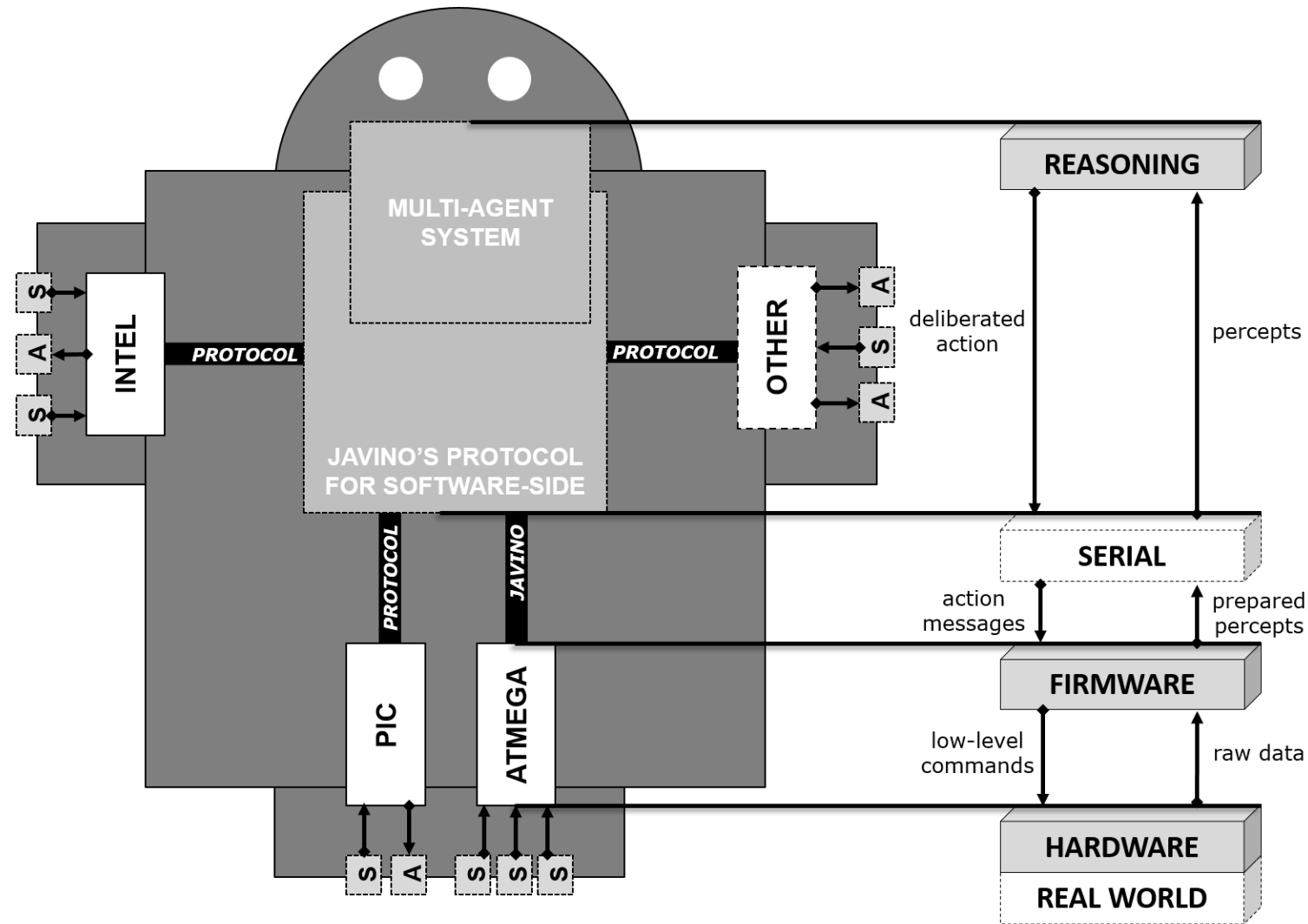
5. Conclusão

6. Referências Bibliográficas

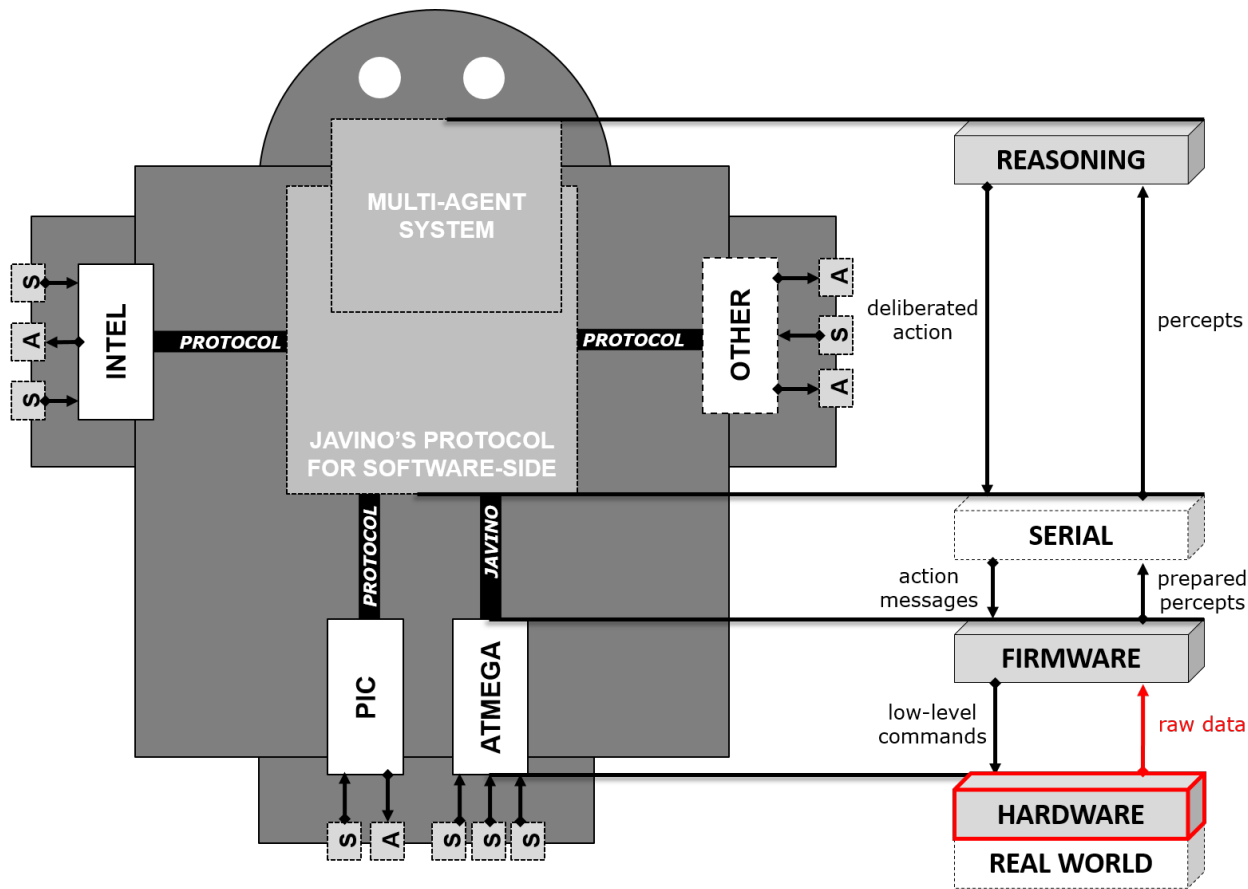
2. ARQUITETURA



2. ARQUITETURA: FLUXO DA MENSAGEMEM

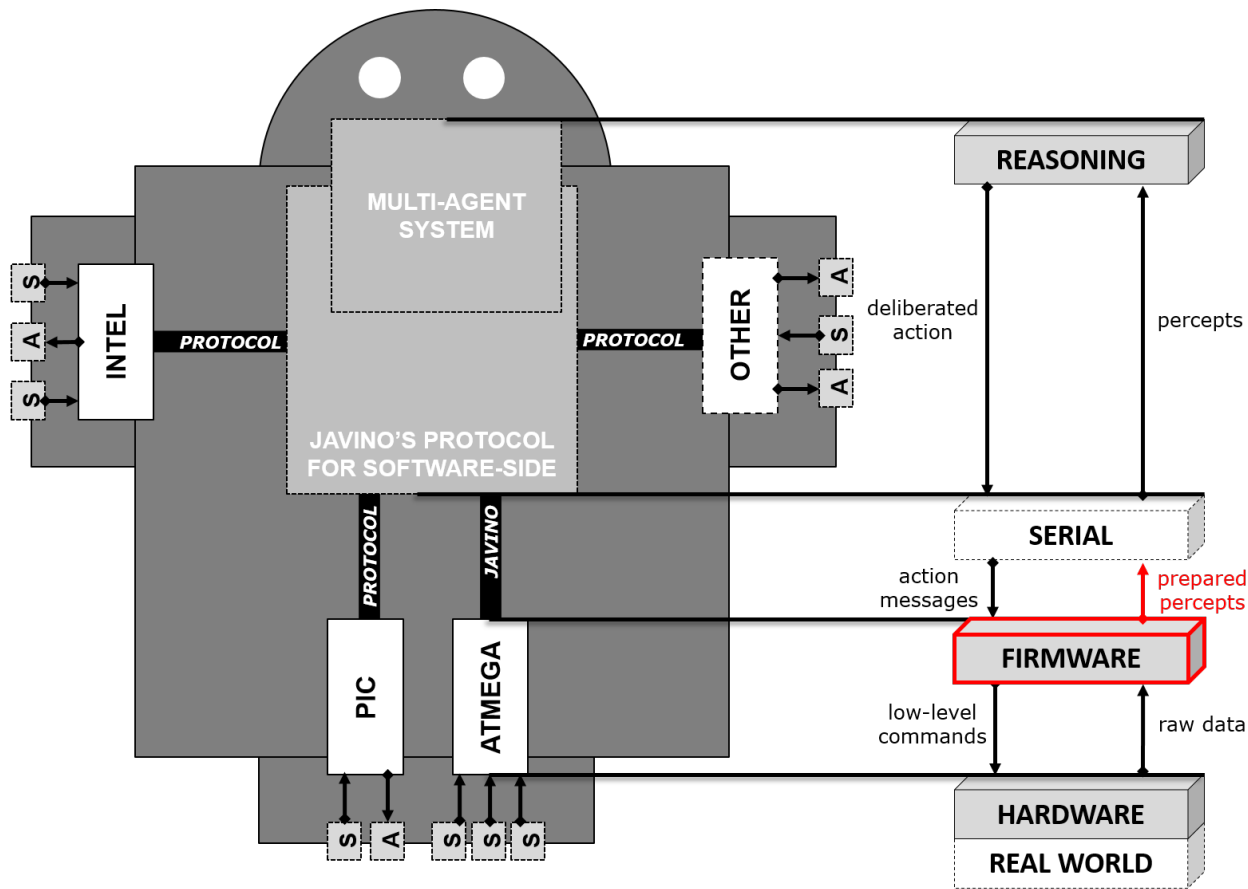


2. ARQUITETURA: FLUXO DA MENSAGEM



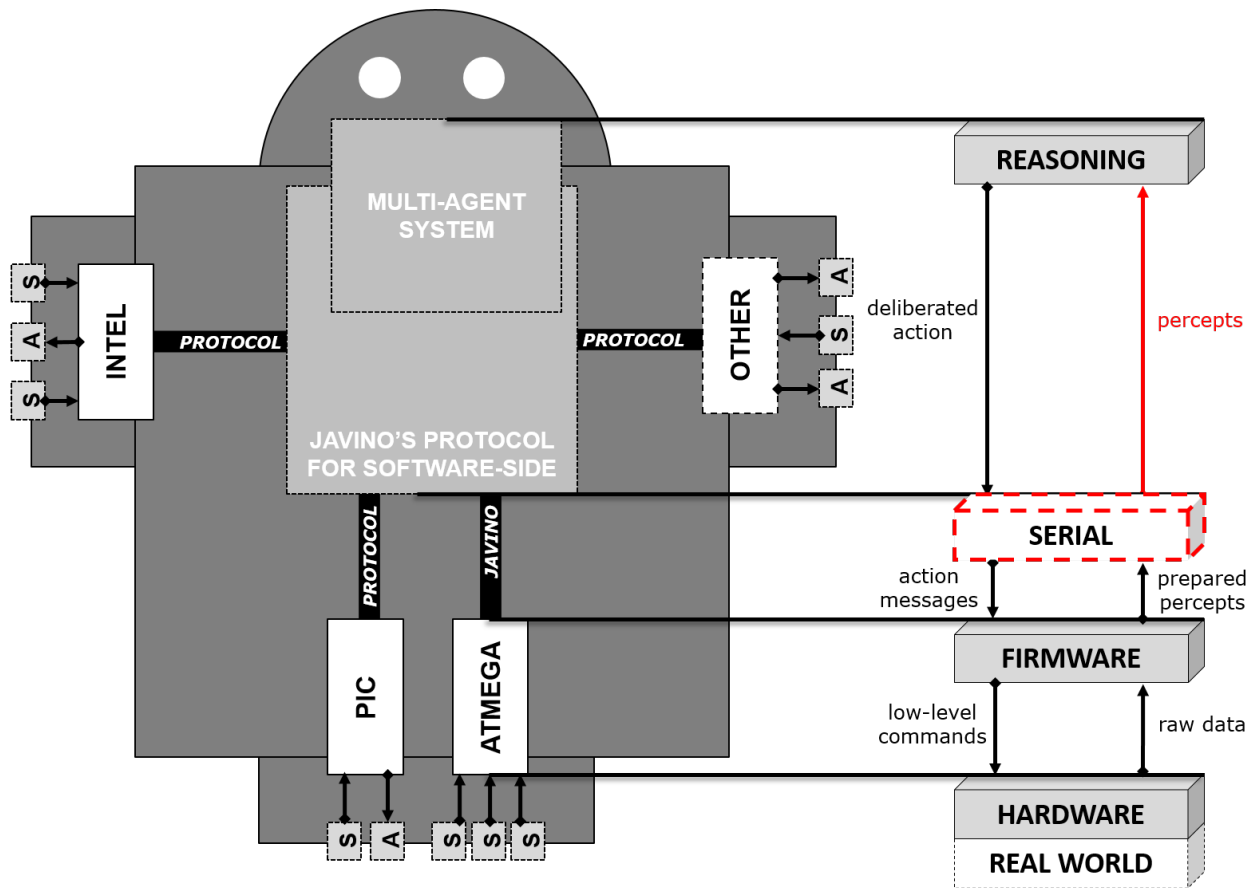
Sensores capturam dados brutos do mundo real e os envia para um dos microcontroladores escolhido para o projeto.

2. ARQUITETURA: FLUXO DA MENSAGEM



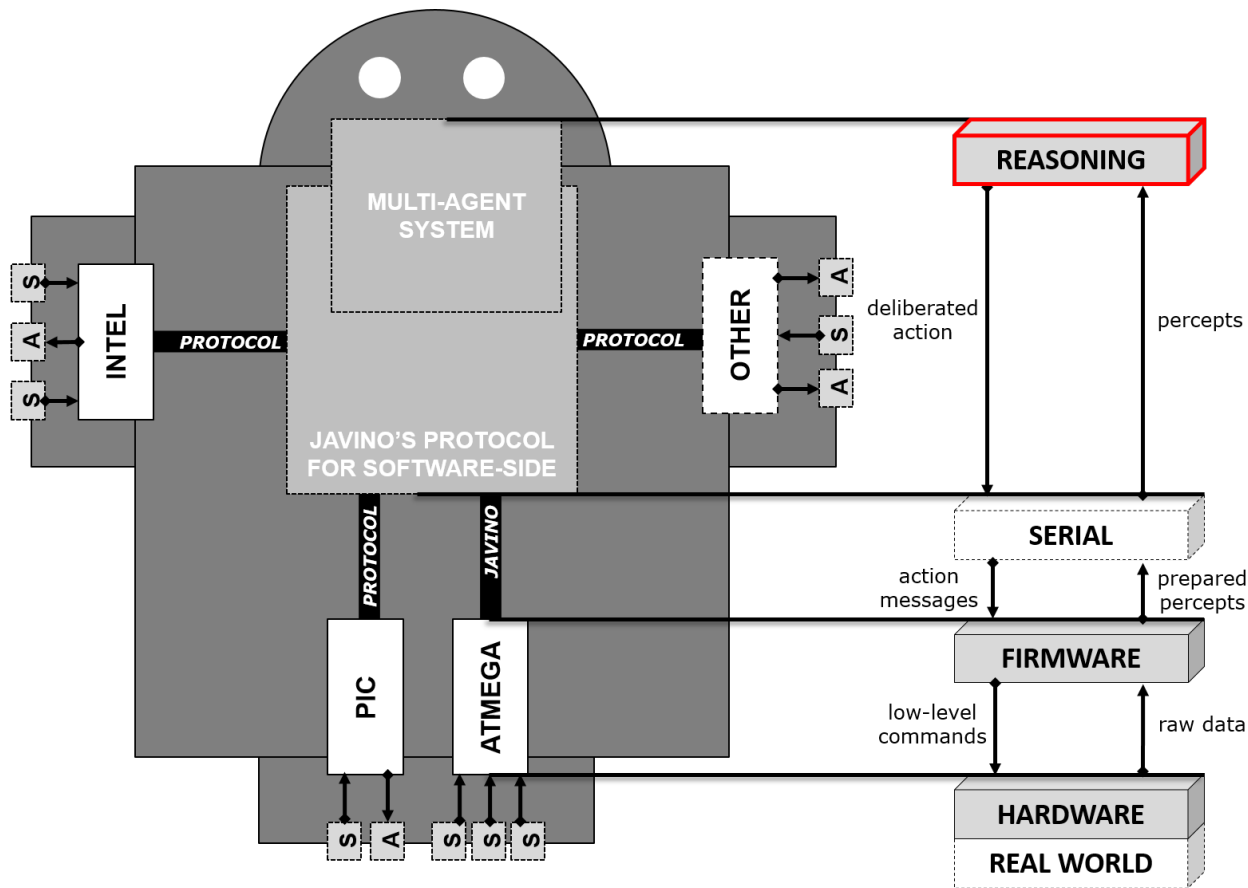
Na programação do microcontrolador, os dados brutos são transformados em percepções baseado na linguagem de programação orientada a agentes escolhida.

2. ARQUITETURA: FLUXO DA MENSAGEM



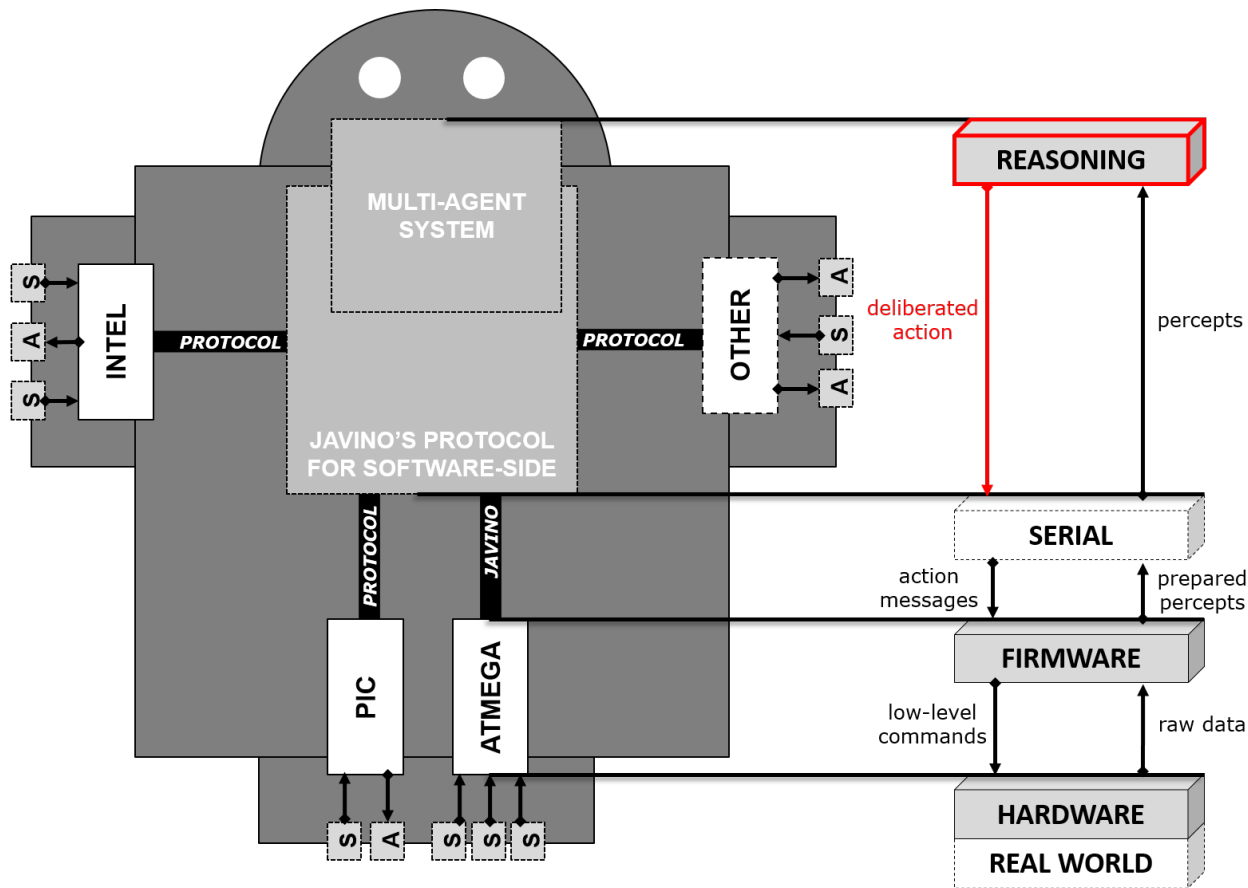
O Javino é responsável por enviar as percepções para a camada de raciocínio usando a comunicação serial.

2. ARQUITETURA: FLUXO DA MENSAGEM



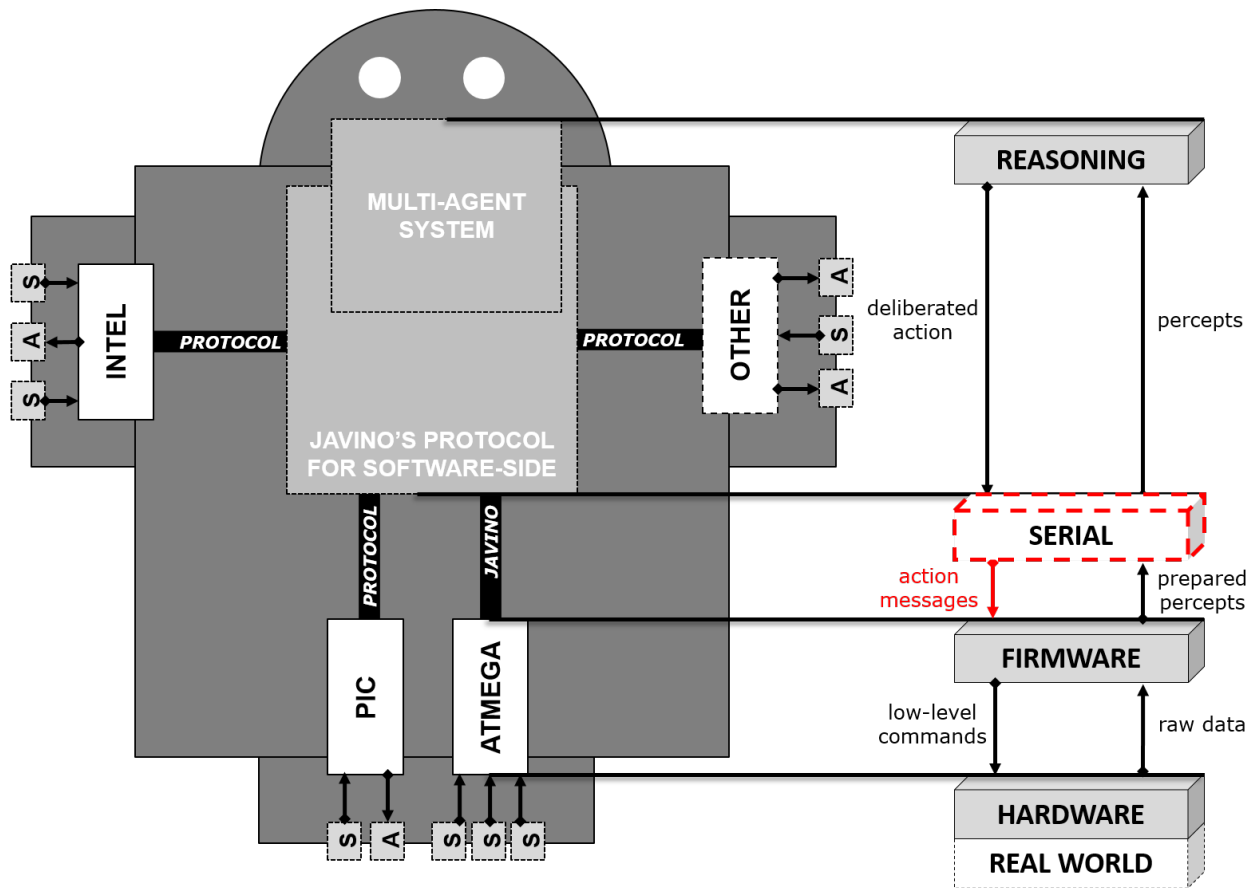
O agente é capaz de raciocinar com as percepções que vem diretamente do mundo real. Além disso, o SMA pode ser embarcado em placas computadorizadas como a Raspberry Pi ou computadores com interface USB.

2. ARQUITETURA: FLUXO DA MENSAGEM



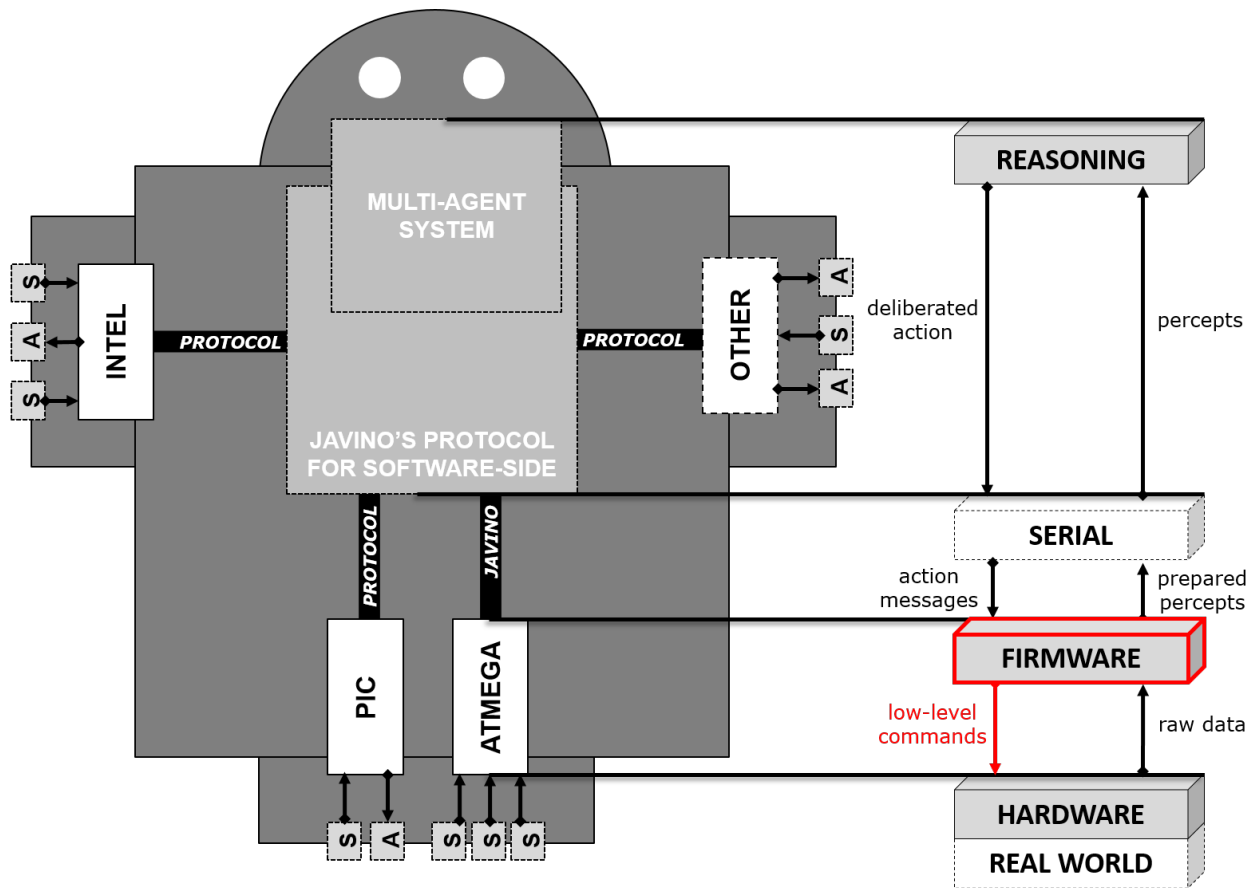
Então, o agente delibera e se alguma ação precisar ser executada, uma mensagem é enviada a camada de baixo utilizando o Javino.

2. ARQUITETURA: FLUXO DA MENSAGEM



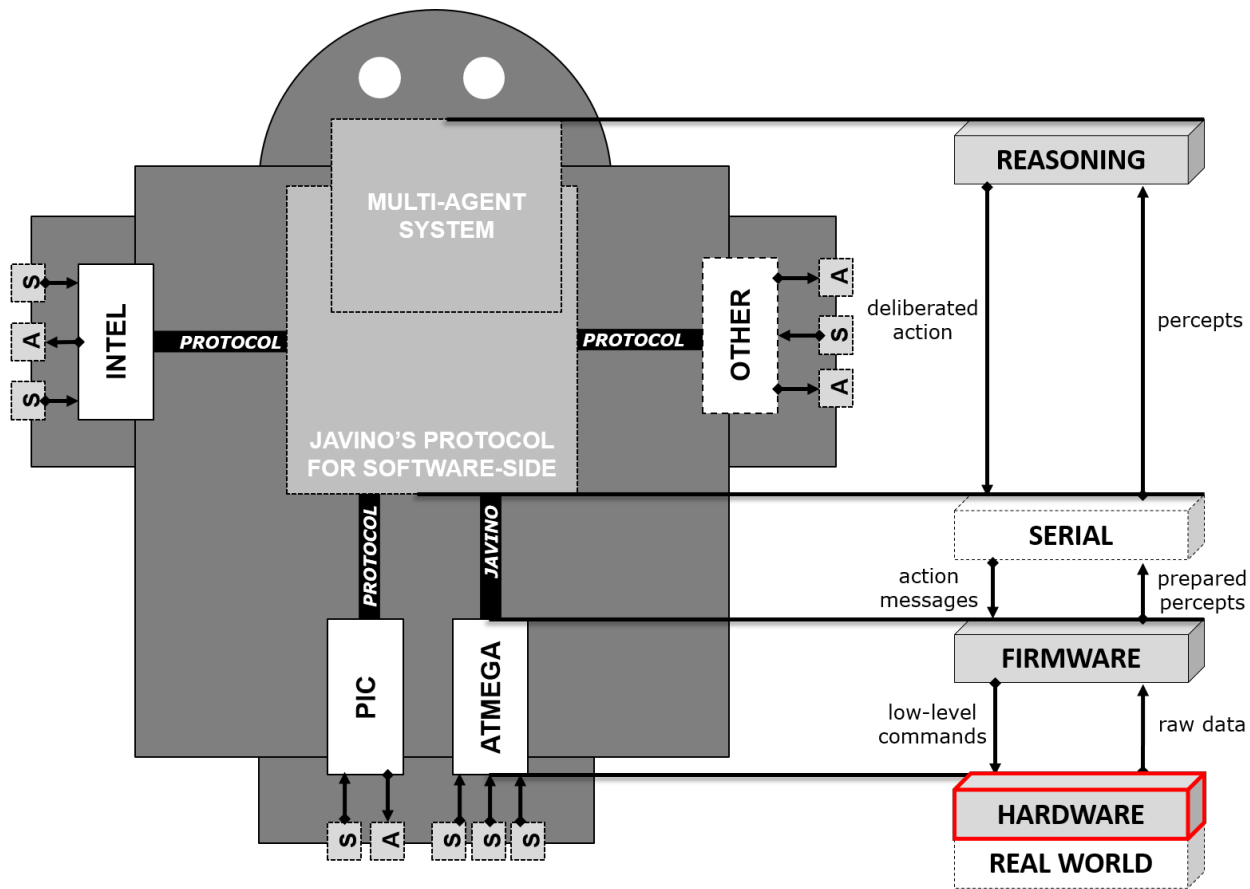
O Javino envia mensagens de ações para o microcontrolador que está conectado na porta USB identificado na mensagem.

2. ARQUITETURA: FLUXO DA MENSAGEM



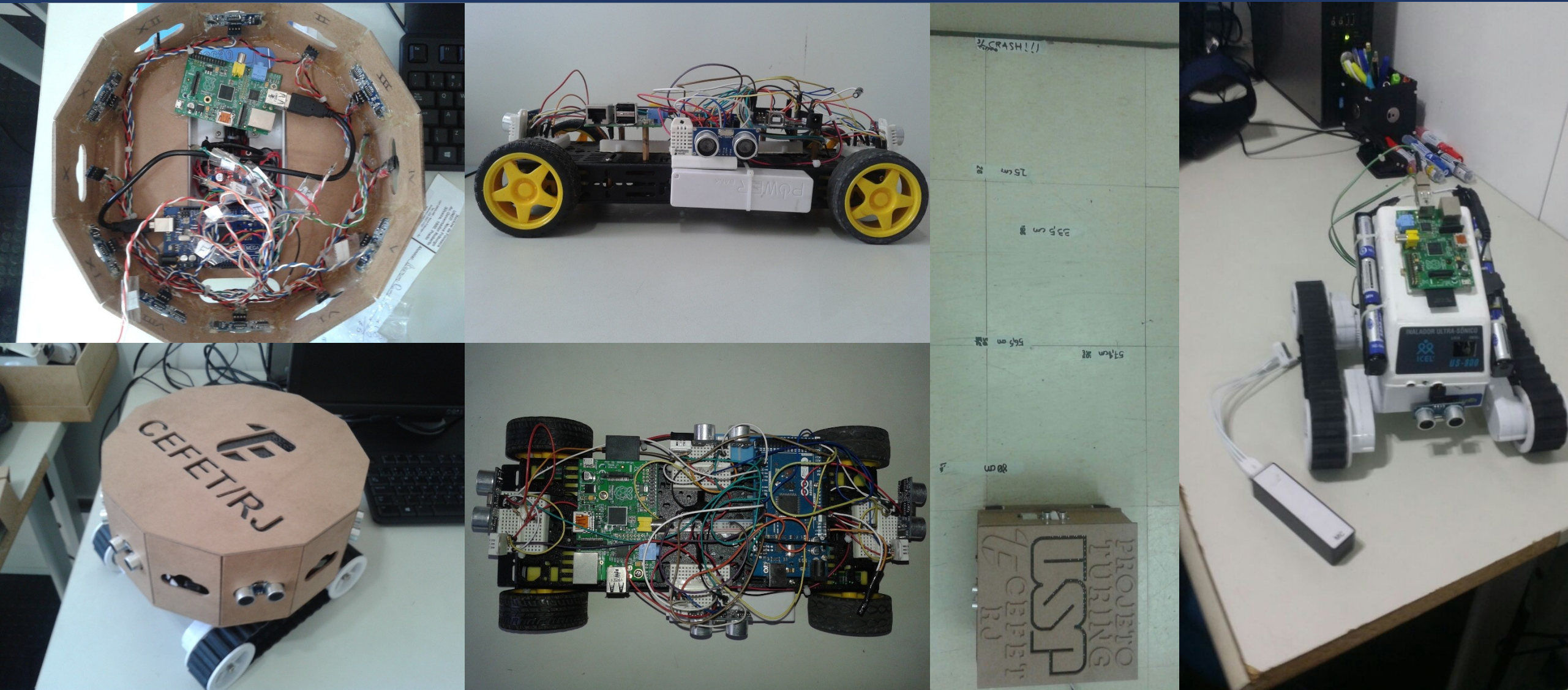
Todas as funções possíveis dos atuadores são programadas para serem executadas em resposta a mensagens da porta seriais vinda do Javino.

2. ARQUITETURA: FLUXO DA MENSAGEM



O efetuator é ativado.

2. ARQUITETURA: PROTÓTIPOS



OUTLINE

1. Introdução

2. A Arquitetura Robótica para SMA

3. Jason Framework

4. ARGO for Jason

5. Conclusão

6. Referências Bibliográficas

3. JASON FRAMEWORK

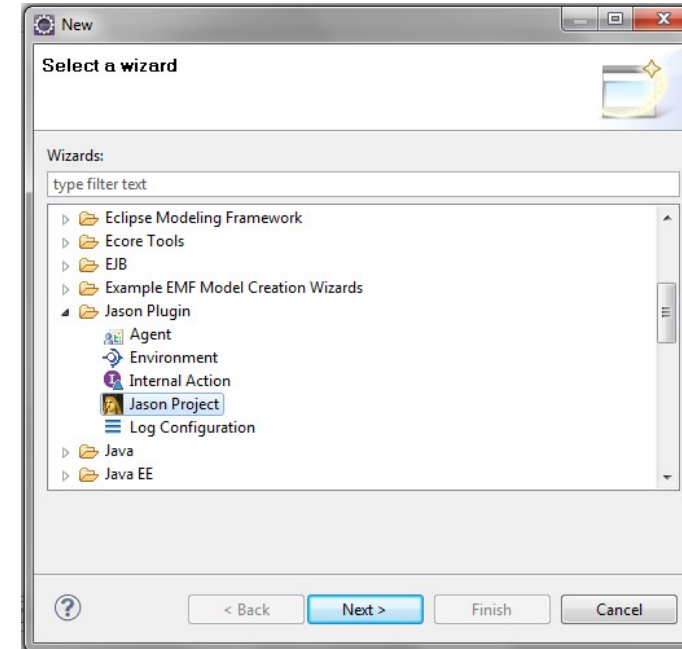
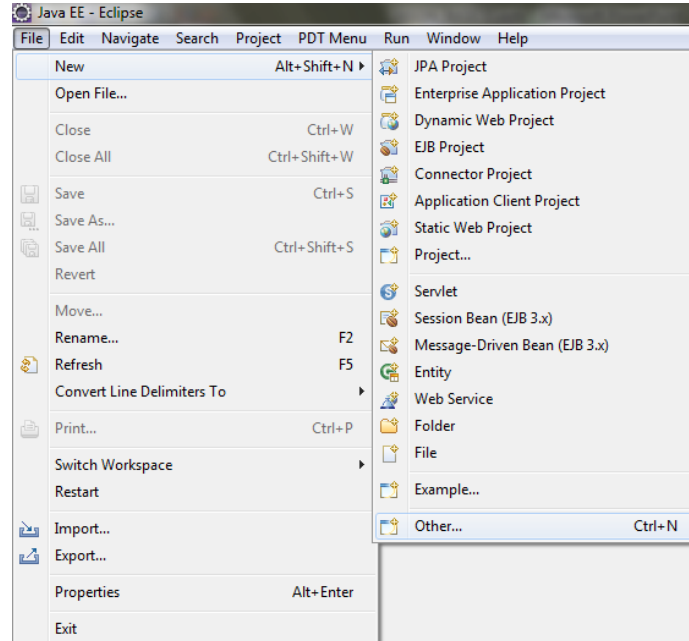
O JASON é um framework baseado em AgentSpeak e Java que utiliza as principais características do PRS. Em JASON um agente é composto de **crenças**, **metas**, **planos** e **ações** e é programado utilizando o **AgentSpeak**.

Os agentes em JASON estão inseridos em um ambiente onde as **percepções** e **reações a estímulos** do ambiente podem ser programadas em Java [Bordini et al., 2007].

Contudo, ao usar o **ARGO**, as **percepções** são provenientes do mundo real (**sensores**) e o agente as recebe automaticamente, sem intervenção do programador [Pantoja et al., 2016].

3. JASON FRAMEWORK

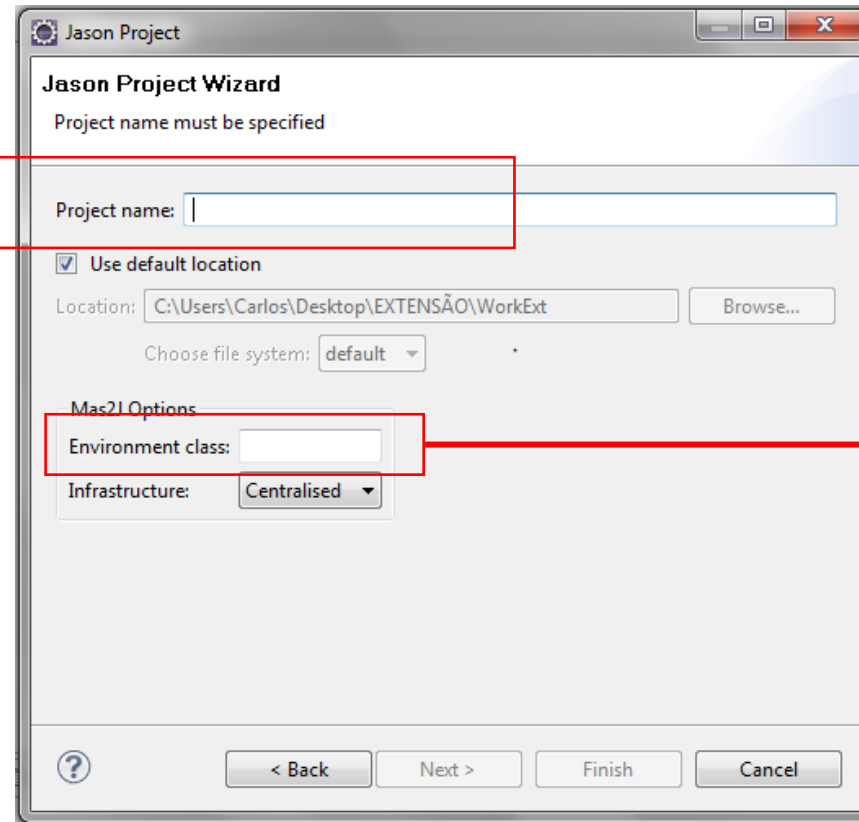
- **Criando um Novo Projeto Jason**
 - File>New>Other>Jason Project



3. JASON FRAMEWORK

- **Criando um Novo Projeto Jason**

Colocar o
nome do
projeto



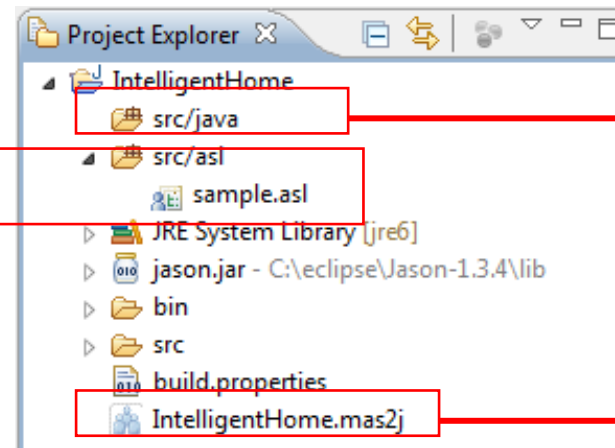
The screenshot shows the 'Jason Project Wizard' dialog box. It has a title bar 'Jason Project' and a subtitle 'Jason Project Wizard'. Below the subtitle is the text 'Project name must be specified'. There is a text input field for 'Project name:'. Below this is a checked checkbox 'Use default location' and a text field for 'Location: C:\Users\Carlos\Desktop\EXTENSÃO\WorkExt' with a 'Browse...' button. Below that is a 'Choose file system:' dropdown menu set to 'default'. Under the 'Mas21 Options' section, there is a text input field for 'Environment class:' and a dropdown menu for 'Infrastructure:' set to 'Centralised'. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

Começar o
projeto
com um
ambiente
definido

3. JASON FRAMEWORK

- **Criando um Novo Projeto Jason**

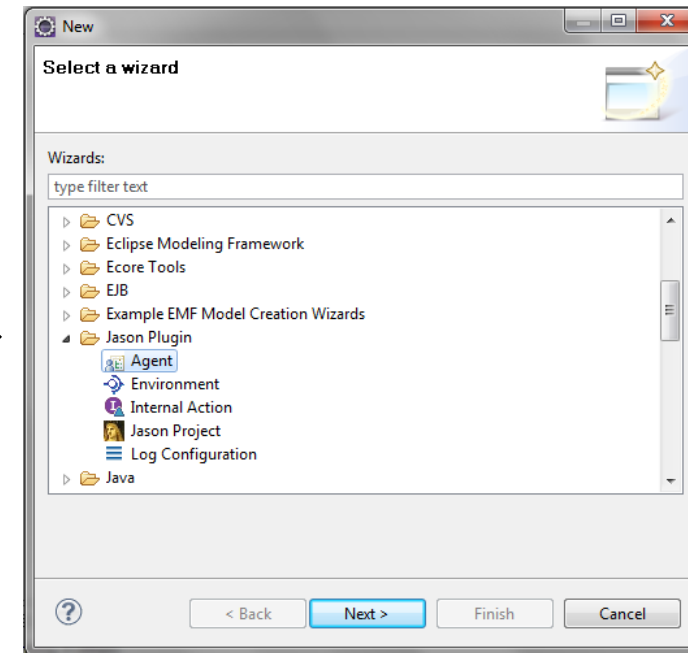
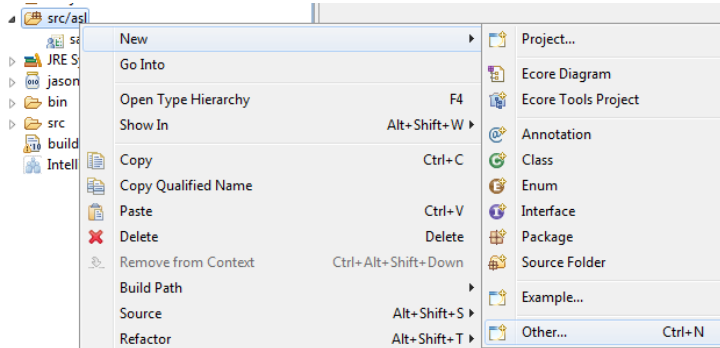
Pasta com
os arquivos
dos
agentes
em
AgentSpea
k



Pasta com
os arquivos
do
ambiente
em Java
Arquivo de
configuraç
ão do SMA

3. JASON FRAMEWORK

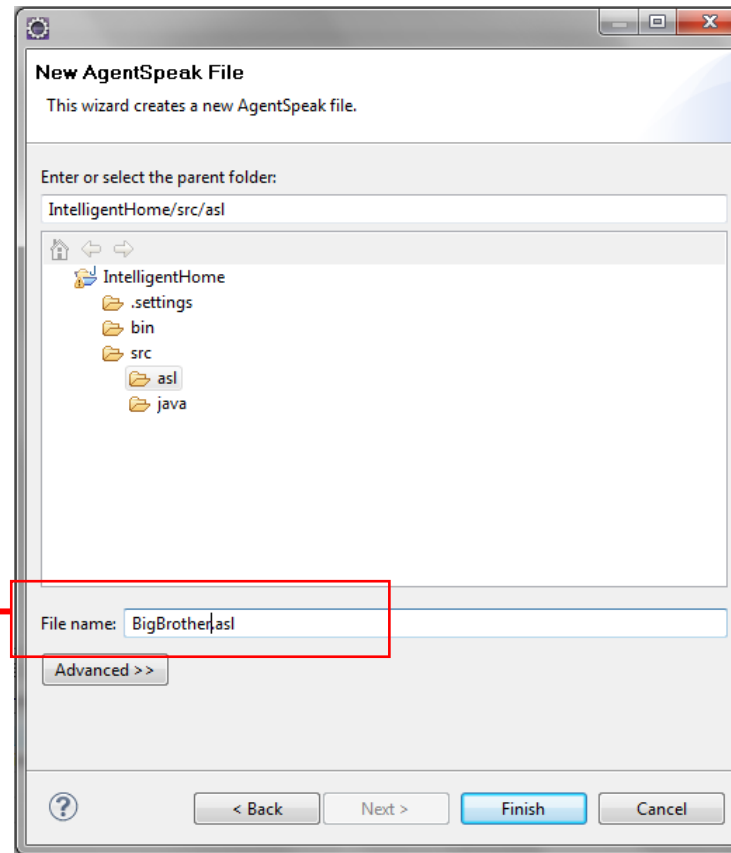
- **Inserindo um Novo Agente**
 - File>New>Other>Agent



3. JASON FRAMEWORK

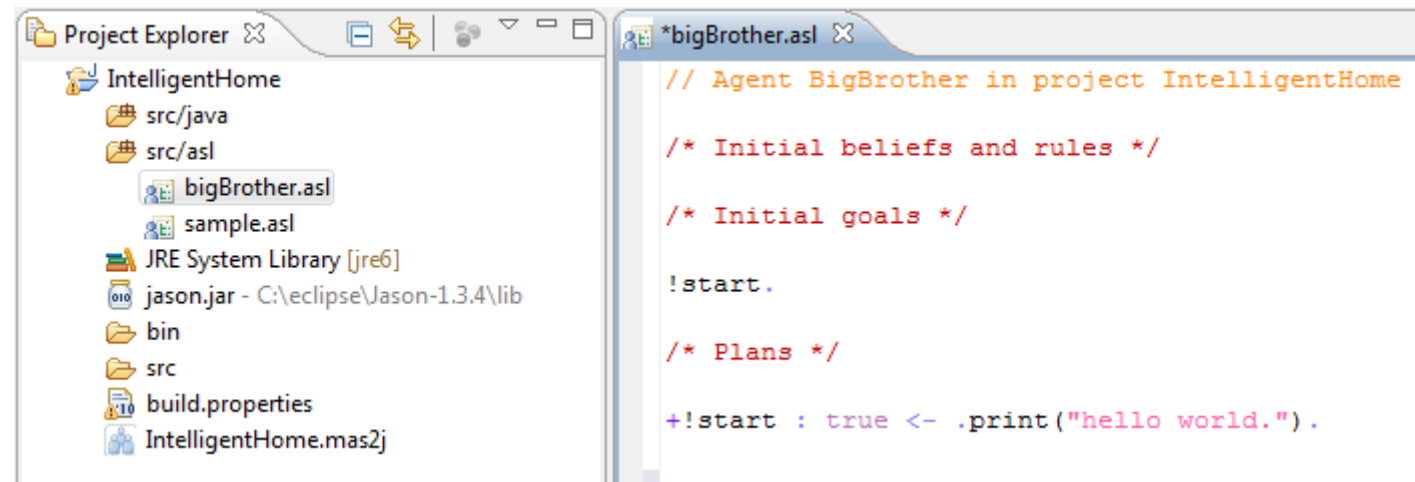
- **Inserindo um Novo Agente**

Digitar o
nome do
Agente. A
primeira
letra deve
ser
MINÚSCUL
A.



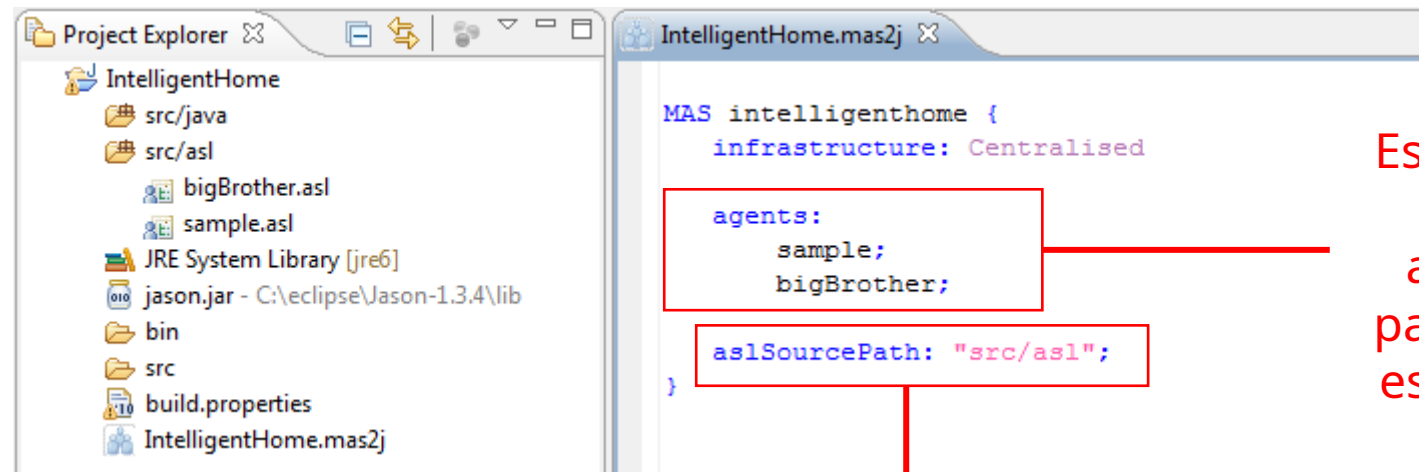
3. JASON FRAMEWORK

- **Inserindo um Novo Agente**



3. JASON FRAMEWORK

- **Configurando o SMA**

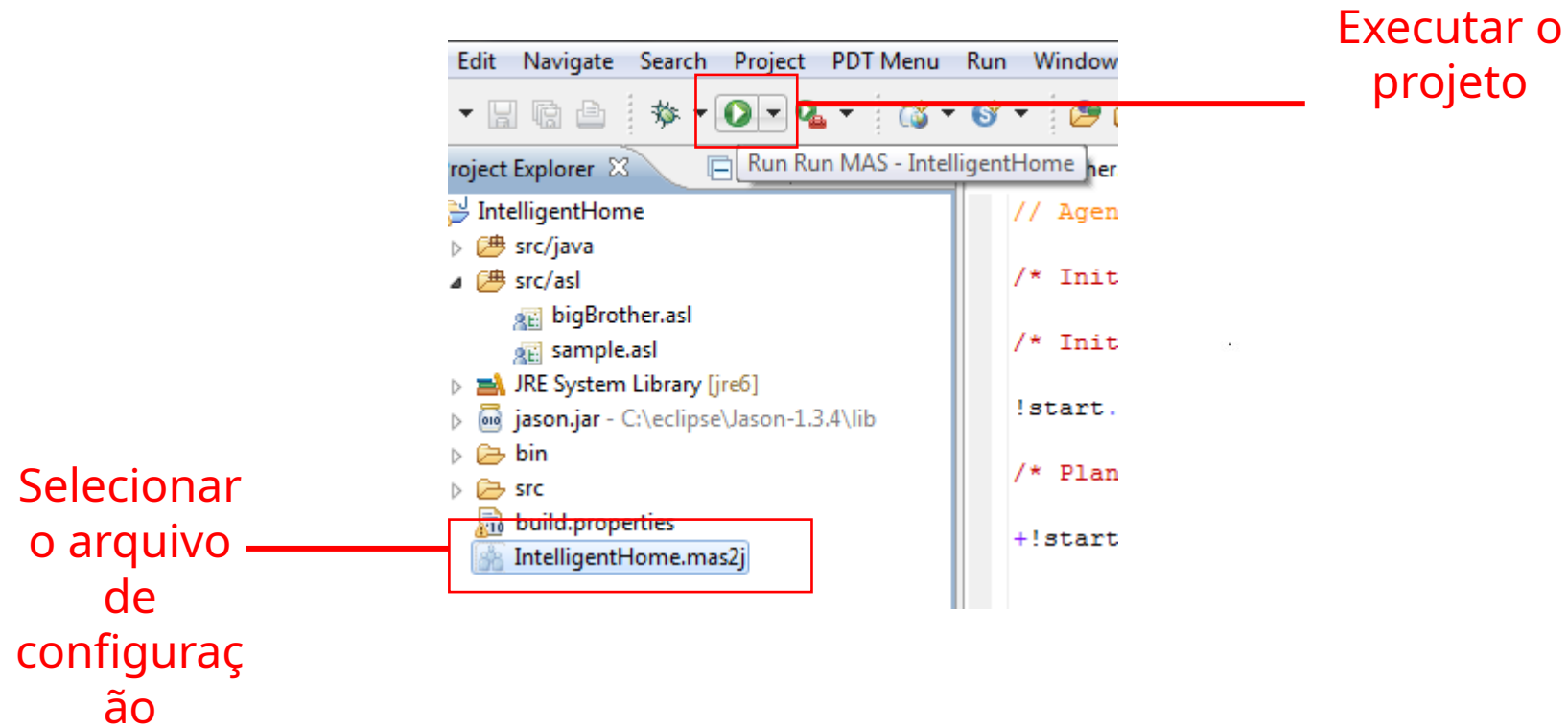


Especificação dos agentes participantes do SMA

Especificação do endereço da pasta onde estão localizados os agentes

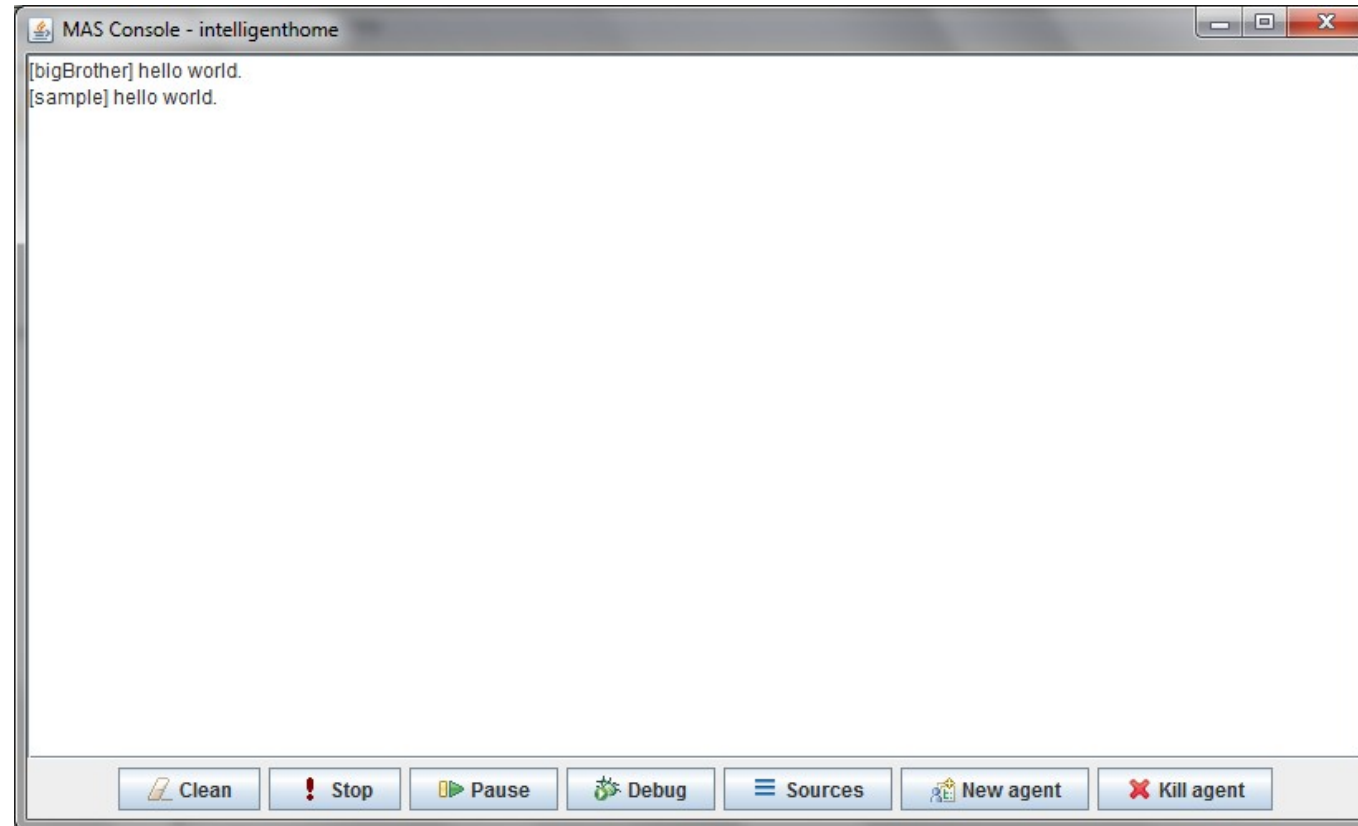
3. JASON FRAMEWORK

- **Executando o SMA**



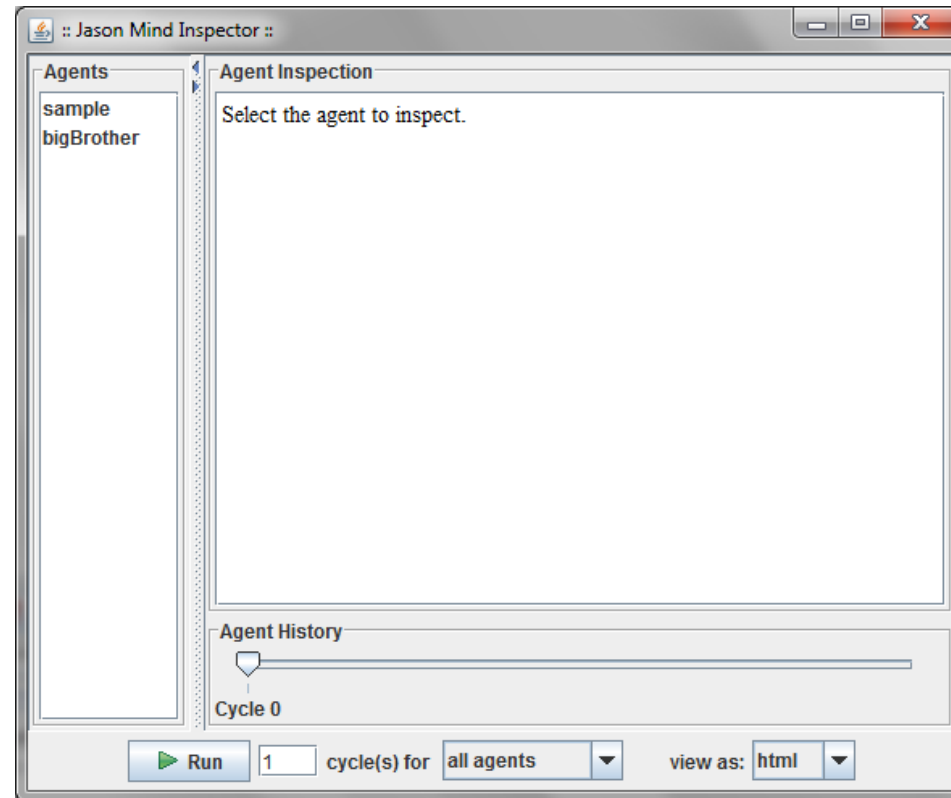
3. JASON FRAMEWORK

- **Executando o SMA**



3. JASON FRAMEWORK

- **Debug do SMA**



3. JASON FRAMEWORK: CRENÇAS

- **Beliefs**

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

As crenças são representadas como predicados da **lógica tradicional**. Os **predicados** representam propriedades particulares.

3. JASON FRAMEWORK: CRENÇAS

- **Tipos**

- 1. Percepções do Ambiente (Percepts)**

Informações coletadas pelo agente que são relativas ao sensoramento constante do ambiente.

- 2. Notas Mentais (Mental Notes)**

- Informações adicionadas na base de crenças pelo próprio agente.
 - ✓ coisas que aconteceram no passado;
 - ✓ promessas;
 - ✓ execução de um plano;
 - ✓ Constante do ambiente.

- 3. Comunicação**

Informações obtidas pelo agente através da interação com outros agentes.

3. JASON FRAMEWORK: CRENÇAS

- **Exemplos: Crenças Iniciais**

salario(5000).

paulo(alto)

missionStarted.

carro(ferrari, kadu).

OBS. 1: Toda **crença inicial** em Jason deve terminar com .

OBS. 2: Toda **crença** deve começar com letra **MINÚSCULA**.

3. JASON FRAMEWORK: CRENÇAS

- **Exemplos: Strong Negation**

~missionStarted.

~alto(carlos).

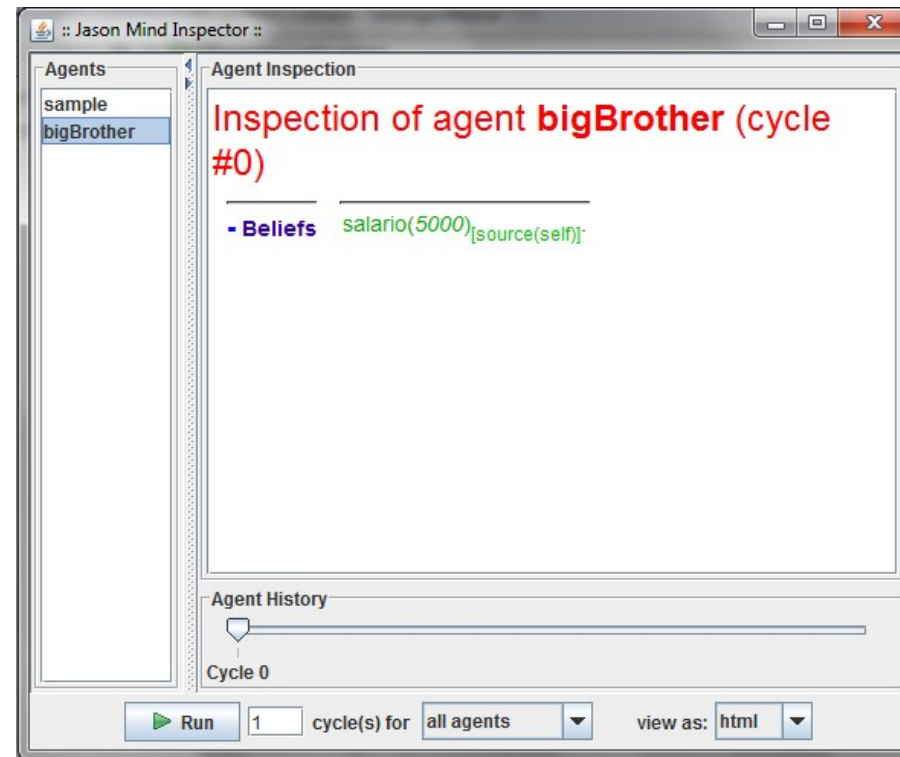
~dia.

OBS.: Toda **strong negation** em Jason
deve começar com **~**

3. JASON FRAMEWORK: CRENÇAS

- **Exemplos: Crenças Iniciais**

salario(5000).



3. JASON FRAMEWORK: GOALS

- **Goals**

Em Jason, os **goals** (objetivos) representam os estados do mundo em que o agente deseja atingir.

- **Tipos**

1. **Achievement Goals (!)**

- ✓ É um objetivo para atingir determinado estado desejado pelo agente.

2. **Test Goals (?)**

- ✓ É um objetivo que tem basicamente a finalidade de resgatar informações da base de crenças do agente.

3. JASON FRAMEWORK: GOALS

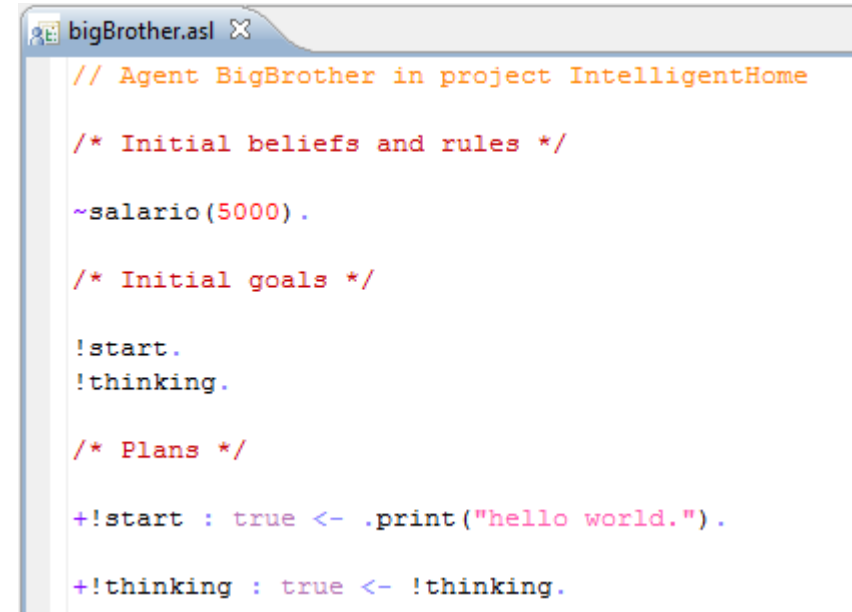
- **Exemplos: Goals Iniciais**

!start.

!thinking.

OBS. 1: Toda **goal inicial** em Jason deve ser um Achievement Goal; começar com **!**; e terminar com **.**

OBS. 2: Todo **goal** deve começar com letra **MINÚSCULA**.



```
bigBrother.asl
// Agent BigBrother in project IntelligentHome

/* Initial beliefs and rules */

~salario(5000).

/* Initial goals */

!start.
!thinking.

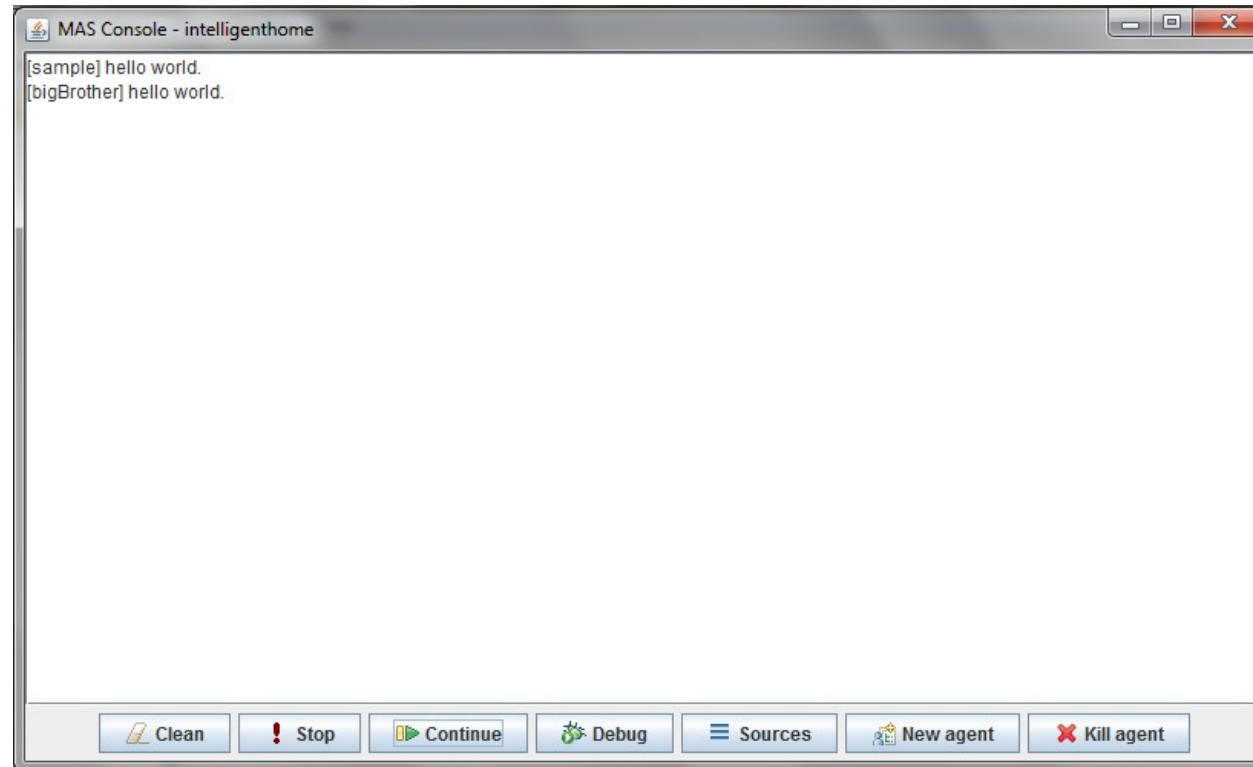
/* Plans */

+!start : true <- .print("hello world.").

+!thinking : true <- !thinking.
```


3. JASON FRAMEWORK: GOALS

- **Exemplos: Goals Iniciais**



3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Plans & Actions**

Em Jason, um plano é composto por três partes:

Triggering_event : **context** <- **body**.

```
+!order(Product,Qtd)[source(Ag)] : true <-  
  ?last_order_id(N);  
  OrderId = N + 1;  
  -+last_order_id(OrderId);  
  deliver(Product,Qtd);  
  .send(Ag, tell, delivered(Product,Qtd,OrderId)).
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Formato de um Plano**

- 1. Triggering Event**

Um agente pode ter diversos objetivos. Os planos são ativados baseados nos eventos que podem ser ativados em determinado momento.

- 2. Context**

São as condições para a ativação de um plano dentro vários eventos.

- 3. Body**

É o corpo do plano. Uma sequência de ações a ser executada pelo agente.

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Tipos de Triggering Events**

1. **Addition**

São ativados quando um plano é transformado de um desejo para uma **intenção** na mente do agente.

```
!bark.  
  
+!bark : true <-  
  .print("Au Au Au!").
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Tipos de Triggering Events**

2. Deletion

Funciona como um **"tratamento de erros"** para planos que não possuem ativação.

```
!bark.
```

```
+!bark : dog(unknow) <-  
    .print("Au Au Au!");
```

```
-!bark <-  
    .print("sniff sniff!");  
!bark.
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Tipos de Planos**

1. **Achievement Goal**

São **objetivos** que os agentes se comprometem em **atingir**.

```
!bark.
```

```
+!bark : true <-  
    .print("Au Au Au!").
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Tipos de Planos**

2. Test Goal

São planos que são ativados quando se **recuperam informações** da base de crenças.

```
!bark.  
  
+!bark : dog(unknow) <-  
    .print("Au Au Au!");  
  
-!bark <-  
    .print("sniff sniff!");  
    !sniff.  
  
+!sniff <-  
    .print("Is it bob?");  
    ?dog(X);  
    .print(X).
```

```
+?dog(X) <-  
    X = bob;  
    +dog(X).
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Tipos de Planos**

3. Belief

São planos ativados quando o agente **adiciona** ou **remove** uma **crença** da sua base de crenças

```
!sniff.
```

```
+!sniff <-  
  .print("Is it bob?");  
  +dog(bob).
```

```
+dog(bob) <-  
  .print("sniff sniff!").
```


3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Ações de um Plano**

1. Achievement e Test Goals

São as chamadas para execução de um plano.

```
!bark.  
  
+!bark <-  
    .print("sniff!");  
    !sniff;  
    .print("sniff!");  
  
+!sniff <-  
    .print("Is it bob?");  
    ?dog(X);  
    .print(X).  
  
+?dog(X) <-  
    X = bob;  
    +dog(X);  
    .print("I found X").
```

```
!bark.  
  
+!bark <-  
    .print("sniff!");  
    !!sniff;  
    .print("sniff!");  
  
+!sniff <-  
    .print("Is it bob?");  
    ?dog(X);  
    .print(X).  
  
+?dog(X) <-  
    X = bob;  
    +dog(X);  
    .print("I found X").
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Ações de um Plano**

2. Mental Notes

São ações que **adicionam**, **removem** ou **atualizam** uma **crença** na base de crença do agente.

```
!sniff.  
  
+!sniff <-  
  .print("Is it bob?");  
  +dog(bob).  
  
+dog(bob) <-  
  .print("sniff sniff!").
```

```
hungry.  
food(100).  
stomach(0).  
  
!eat.  
  
+!eat: hungry & food(F) & stomach(S) & S<=50 <-  
  .print("Eating...");  
  +-food(F-1);  
  +-stomach(S+1);  
  .print(F);  
  !eat.  
  
+!eat: stomach(S) & S>50 <-  
  .print("I'm Satisfied.");  
  -hungry.
```

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Ações de um Plano**

3. Internal Action

São ações pré-definidas **executadas** internamente no agente.

<code>.print</code>	<code>.max</code>	<code>.create_agent</code>
<code>.send</code>	<code>.nth</code>	<code>.date</code>
<code>.broadcast</code>	<code>.sort</code>	<code>.wait</code>
<code>.drop_all_desires</code>	<code>.substring</code>	<code>.random</code>
<code>s</code>	<code>.drop_all_events</code>	<code>.kill_agent</code>
<code>.my_name</code>	<code>.abolish</code>	<code>.time</code>
<code>.concat</code>	<code>.string</code>	<code>.perceive</code>
<code>.length</code>	<code>.count</code>	<code>.stopMAS</code>
<code>.min</code>		

3. JASON FRAMEWORK: PLANOS E AÇÕES

- **Ações de um Plano**

4. Expressões

```
!bark.  
  
+!bark <-  
    .print("sniff!");  
    !sniff;  
    .print("sniff!");  
  
+!sniff <-  
    .print("Is it bob?");  
    ?dog(X);  
    .print(X).  
  
+?dog(X) <-  
    X = bob;  
    +dog(X);  
    .print("I found X").
```

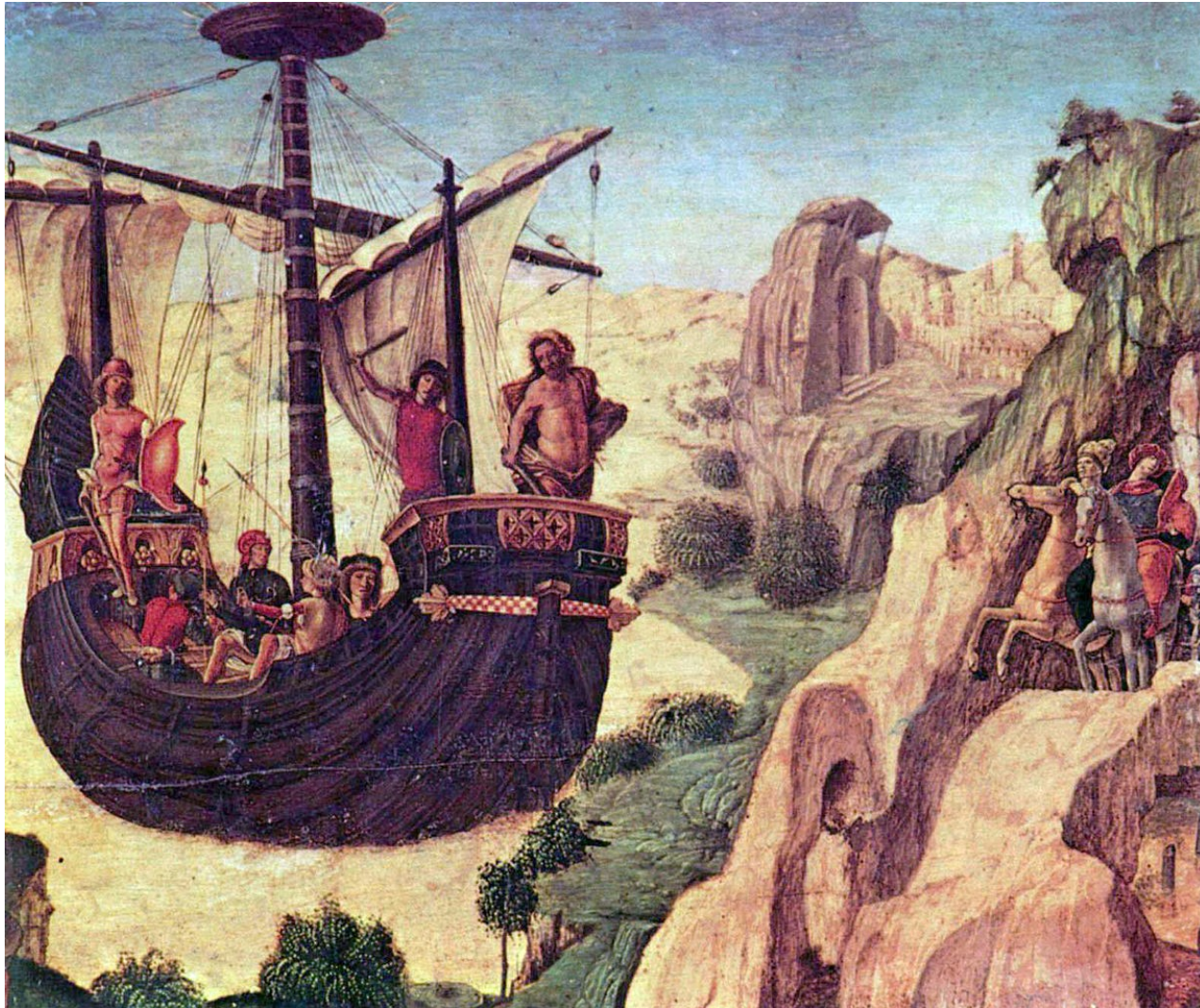
3. JASON FRAMEWORK: EXERCÍCIO

- 1) Criar um agente com as crenças iniciais com seu **name** e **age**.
- 2) Adicionar nesse agente com um objetivo inicial **introduce** e implemente um plano que informe o seu nome e sua idade na tela.
- 3) Adicionar nesse agente um outro plano com o mesmo nome **introduce** que seja responsável por informar caso o agente já tenha se apresentado.
- 4) Adicione um plano para imprimir sua idade caso ele acredite que o dia é segunda-feira. Adicione um plano de contenção caso não exista a crença de que dia é hoje na base de crenças do agente.

OUTLINE

1. Introdução
2. A Arquitetura Robótica para SMA
3. Jason Framework
4. ARGO for Jason
5. Conclusão
6. Referências Bibliográficas

2. ARGO FOR JASON



Argo foi o barco que Jasão (**Jason**) e os Argonautas navegaram na busca pelo **velocino de ouro** na mitologia grega.

The Argo
by Lorenzo Costa

2. ARGO FOR JASON

O **ARGO** é uma arquitetura customizada que emprega o **middleware Javino** [Lazarin e Pantoja, 2015], que provê uma **ponte** entre o agente inteligente e os sensores e atuadores do robô.

Além disso, o **ARGO** possui um mecanismo de **filtragem de percepções** [Stabile Jr e Sichman, 2015] em tempo de execução.

O **ARGO** tem como objetivo ser uma arquitetura prática para a **programação de agentes robóticos embarcados** usando agentes BDI e placas microcontroladas como o **Arduino**.

2. ARGO FOR JASON: FUNCIONALIDADES

O ARGO permite:

1. Controlar diretamente os atuadores em tempo de execução;
2. Receber percepções dos sensores automaticamente dentro de um período de tempo pré-definido;
3. Mudar os filtros de percepção em tempo de execução;
4. Alterar quais os dispositivos que estão sendo acessados em tempo de execução;
5. Se comunicar com outros agentes em Jason;
6. Decidir quando perceber ou não o mundo real em tempo de execução.

2. ARGO FOR JASON: AÇÕES INTERNAS

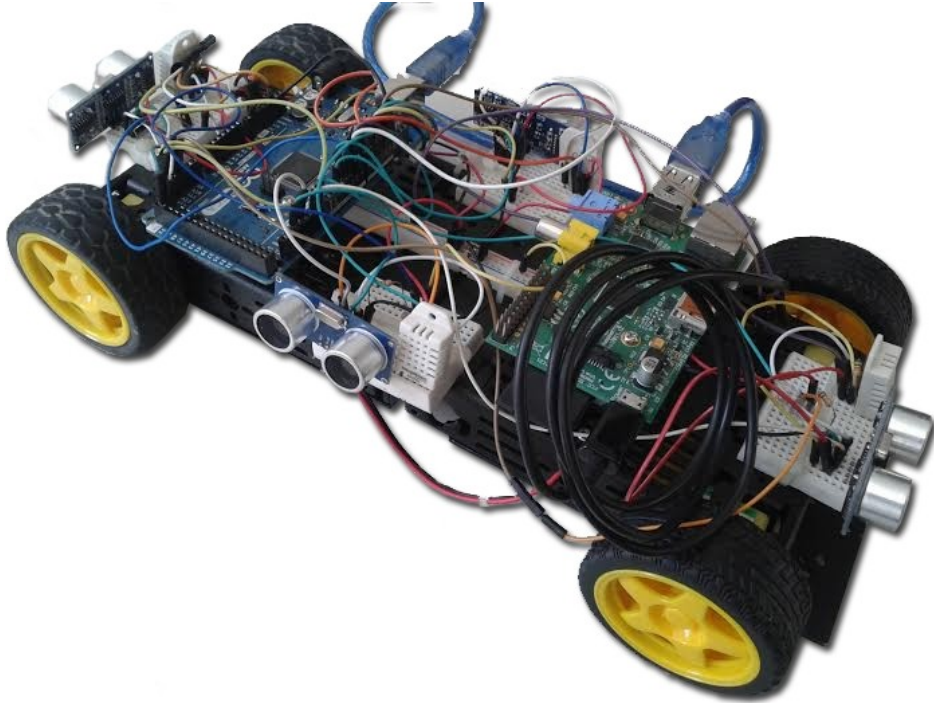
- ARGO Internal Actions:
 - `.limit(x)`
 - Define um intervalo de tempo para perceber o ambiente
 - `.port(y)`
 - Define qual porta serial deve ser utilizada pelo agente
 - `.percepts(open|block)`
 - Decide quando perceber ou não o mundo real
 - `.act(w)`
 - Envia ao microcontrolador uma ação para ser executada por um efetuator
 - `.change_filter(filterName)`
 - Define um filtro de percepção para restringir percepções em tempo real

2. ARGO FOR JASON: LIMITAÇÕES

Limitações:

- Limite de 127 portas seriais
 - O limite da USB
- Uma porta de cada vez
 - Sem competição de porta para evitar conflitos [Guinelli et al., 2016].
 - As portas podem ser mudadas em tempo de execução.
- Só agentes ARGO podem controlar dispositivos
 - Agentes em Jason não possuem as funcionalidades do ARGO.
- Só pode existir uma instância para cada arquivo do agente
 - Se mais de um agente com o mesmo código for instanciado, conflitos acontecem

2. ARGO FOR JASON: EXEMPLO



- O robô [Pantoja et al., 2016]:
 - quatro sensores de distância
 - quatro sensores de luminosidade
 - quatro sensores de temperatura
 - uma placa Arduino
 - um chassis 4WD
- dois metros de distância de um muro
- velocidade constante
- o robô deve para após perceber uma distância especificada

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.

+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).

-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.

+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```



Seta a porta serial
COM8. Um
dispositivo Arduino.

```
-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.

+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

Seta um intervalo de 25ms para perceber o mundo real.

```
-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.

+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.

+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

Abre a porta serial
para começar a
receber as
percepções.

```
-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```


2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).  
!config.
```

```
+!config : true <-  
  .port(COM8);  
  .limit(25);  
  .percepts(open);  
  .change_filter(frontSide);  
!start.
```

```
+!start : true <-  
  .act(moveFront);  
  +status(front);  
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-  
  .print("distance: ", X);  
  .act(moveFront);  
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-  
  .act(stop);  
  ?timeDistance(A,B);  
  -status(front);  
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

Ativa o filtro.

```
-!moving <-  
  !!moving.
```

```
+light(X,Y) : Y>100 <-  
  .print("ledLightOff:",X," ",Y);  
  .act(ledLightOff).
```

```
+light(X,Y) : Y<=100 <-  
  .print("ledLightOn:",X," ",Y);  
  .act(ledLightOn).
```

```
+temp(X,Y) : Y>25 <-  
  .print("ledTempOff:",X," ",Y);  
  .act(ledTempOff).
```

```
+temp(X,Y) : Y<=25 <-  
  .print("ledLightOn:",X," ",Y);  
  .act(ledTempOn).
```

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).  
!config.
```

```
+!config : true <-  
  .port(COM8);  
  .limit(25);  
  .percepts(open);  
  .change_filter(frontSide);  
  !start.
```

```
+!start : true <-  
  .act(moveFront);  
  +status(front);  
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-  
  .print("distance: ", X);  
  .act(moveFront);  
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-  
  .act(stop);  
  ?timeDistance(A,B);  
  -status(front);  
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

Envia uma
ação para ser
executado pelo
microcontrolad
or

```
-!moving <-  
  !!moving.
```

```
+light(X,Y) : Y>100 <-  
  .print("ledLightOff:",X," ",Y);  
  .act(ledLightOff).
```

```
+light(X,Y) : Y<=100 <-  
  .print("ledLightOn:",X," ",Y);  
  .act(ledLightOn).
```

```
+temp(X,Y) : Y>25 <-  
  .print("ledTempOff:",X," ",Y);  
  .act(ledTempOff).
```

```
+temp(X,Y) : Y<=25 <-  
  .print("ledLightOn:",X," ",Y);  
  .act(ledTempOn).
```

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.
```

```
+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

```
-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```

Continua se
movendo
enquanto a
distância
percebida é
maior que a
distância
limite

2. ARGO FOR JASON: CÓDIGO DO AGENTE

```
distanceLimit(40).
!config.

+!config : true <-
  .port(COM8);
  .limit(25);
  .percepts(open);
  .change_filter(frontSide);
  !start.
```

```
+!start : true <-
  .act(moveFront);
  +status(front);
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X>J & status(front) <-
  .print("distance: ", X);
  .act(moveFront);
  !moving.
```

```
+!moving: dist(f, X) & distanceLimit(J) & X<=J & status(front) <-
  .act(stop);
  ?timeDistance(A,B);
  -status(front);
  .print("20ms", " - ", "block", " - ", 80, " - ", A, " - ", X).
```

```
-!moving <-
  !!moving.

+light(X,Y) : Y>100 <-
  .print("ledLightOff:",X," ",Y);
  .act(ledLightOff).

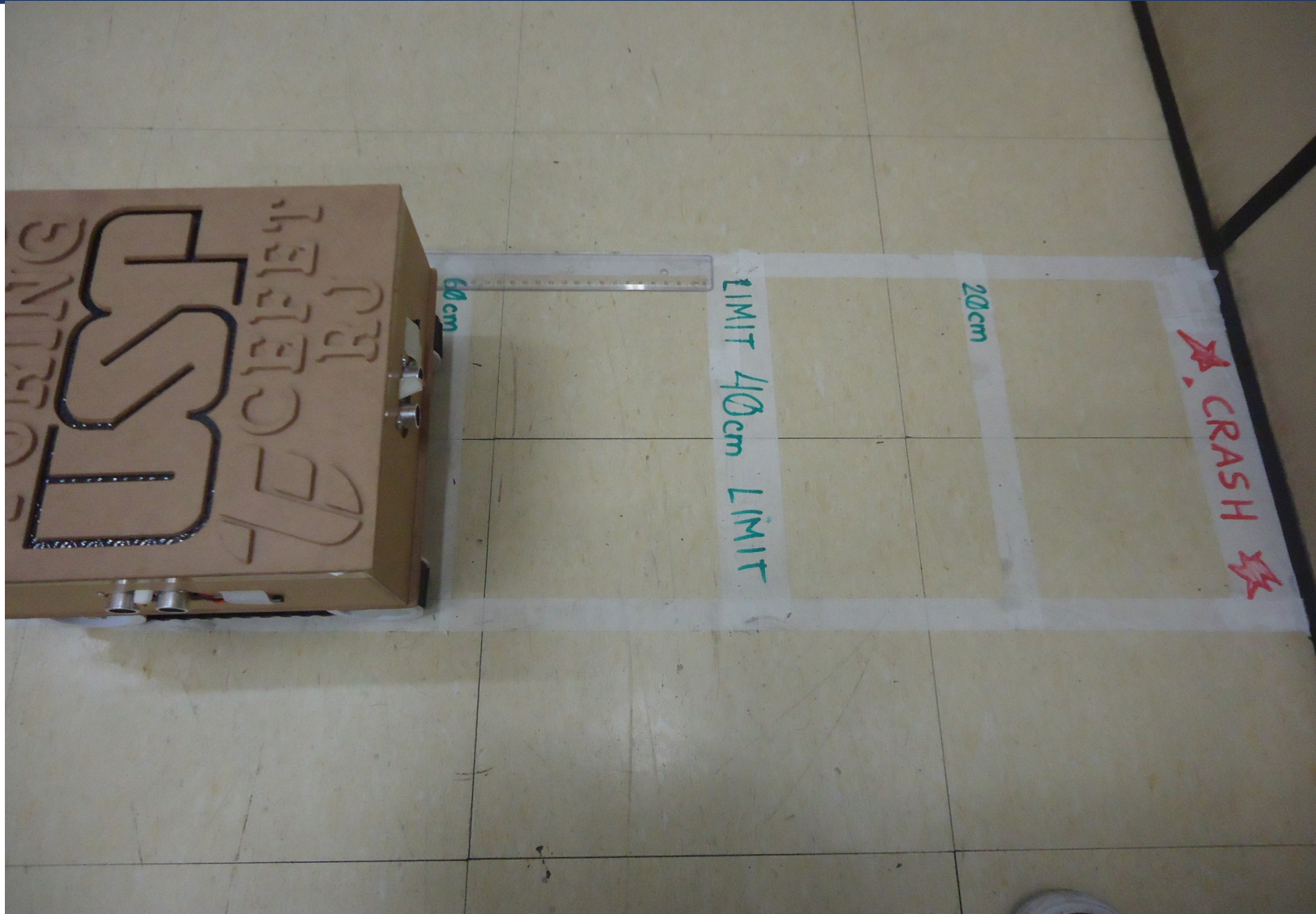
+light(X,Y) : Y<=100 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledLightOn).

+temp(X,Y) : Y>25 <-
  .print("ledTempOff:",X," ",Y);
  .act(ledTempOff).

+temp(X,Y) : Y<=25 <-
  .print("ledLightOn:",X," ",Y);
  .act(ledTempOn).
```

Para quando
perceber o
muro.

2. ARGO FOR JASON



OUTLINE

1. Introdução

2. A Arquitetura Robótica para SMA

3. Jason Framework

4. ARGO for Jason

5. Conclusão

6. Referências Bibliográficas

5. CONCLUSÃO

Desenvolver agentes robóticos controlados por plataformas cognitivas é um desafio na área.

O **ARGO** atua em conjunto com o framework **Jason** para prover a programação de agentes capazes de comandar sensores e atuadores de um robô.

A possibilidade de criação de protótipos com capacidades cognitivas.

OUTLINE

1. Introdução

2. A Arquitetura Robótica para SMA

3. Jason Framework

4. ARGO for Jason

5. Conclusão

6. Referências Bibliográficas

6. REFERÊNCIAS BIBLIOGRÁFICA

- [Bordini et al. 2007] Bordini, R.H., Hubner, J.F., Wooldridge, M. Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons Ltd., 2007.
- [Bratman, 1987] Bratman, M. Intentions, Plans, and Practical Reason. Harvard University Press, 1987.
- [Guinelli et al., 2016] Guinelli, J. V. ; Junger, D. S. ; Pantoja, C. E. . An Analysis of Javino Middleware for Robotic Platforms Using Jason and JADE Frameworks. In: Workshop-Escola de Sistemas de Agentes, Seus Ambientes e Aplicações, Maceió. Anais do X Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações, 2016.
- [Huber, 1999] Huber MJ. Jam: a bdi-theoretic mobile agent architecture. In Proceedings of the third annual conference on Autonomous Agents, AGENTS '99, pags. 236-243, New York, 1999
- [Lazarin and Pantoja, 2015] Lazarin, N.M., Pantoja, C.E. : A robotic-Agent Platform For Embedding Software Agents Using Raspberry Pi and Arduino Boards. In: 9th Software Agents, Environments and Applications School, 2015
- [Pantoja et al., 2016] Pantoja, C. E.; Stabile Jr, M. F. ; Lazarin, N. M. ; Sichman, J. S. ARGO: A Customized Jason Architecture for Programming Embedded Robotic Agents. In: Workshop on Engineering Multi-Agent Systems, 2016, Singapore. Proceedings of the Third International Workshop on Engineering Multi-Agent Systems (EMAS 2016), 2016.

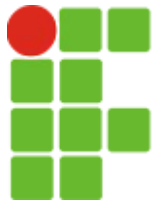
6. REFERÊNCIAS BIBLIOGRÁFICA

- [Rao 1996] Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde,W.V., Perram, J.W. (eds.) Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world. Lecture Notes in Artificial Intelligence, vol. 1038, pp. 42-55. Springer-Verlag, Secaucus. USA, 1996.
- [Stabile Jr. and Sichman, 2015] Stabile Jr., M.F., Sichman, J.S. Evaluating Perception Filters In BDI Jason Agents. In: 4th Brazilian Conference On Intelligent Systems, 2015.
- [Winikoff, 2005] Winikoff M. Jack intelligent agents: An industrial strength platform. Em Bordini R, Dastani M, Dix J, Fallah AS, Weiss G, editors. Multi-Agent Programming, volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, pages. 175-193. Springer US, 2005.
- [Wooldridge, 2000] Wooldridge, M. *Reasoning about rational agents. Intelligent robotics and autonomous agents. MIT Press*, 2000.
- [Wooldridge, 2009] Wooldridge M. An Introduction to MultiAgent Systems. John Wiley & Sons, 2009.
- [Zambonelli et al., 2001] Zambonelli F, Jennings NR, Omicini A, Wooldridge M. Agent-Oriented Software Engineering for Internet Applications. In: Omicini A, Zambonelli F, Klusch M, Tolksdorf R, editors. Coordination of Internet Agents. Springer Verlag; 2001. p.326-345, 2001.

AGRADECIMENTOS

OBRIGADO!

pantoja@cefet-rj.br



INSTITUTO FEDERAL
CATARINENSE
Câmpus Fraiburgo

