

# Introduction to Distributed and Embedded Multi-agent Systems

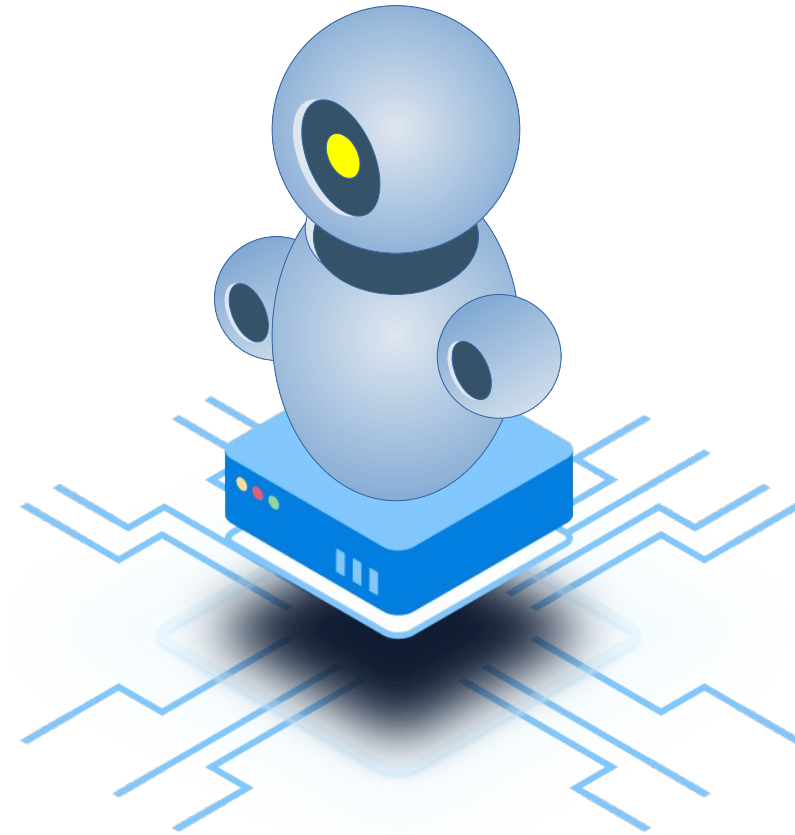
**Carlos Eduardo Pantoja<sup>1</sup>**  
**Nilson Mori Lazarin<sup>1,2</sup>**

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



Junho, 2024

# PHYSICAL AGENT

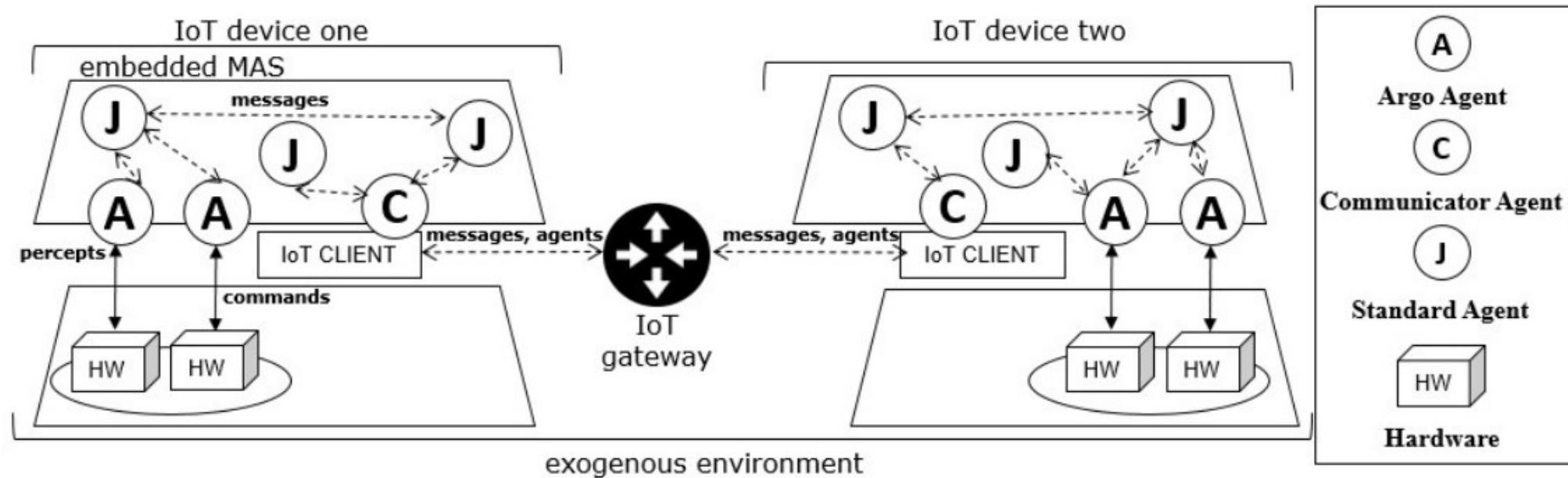


# Jason Embedded

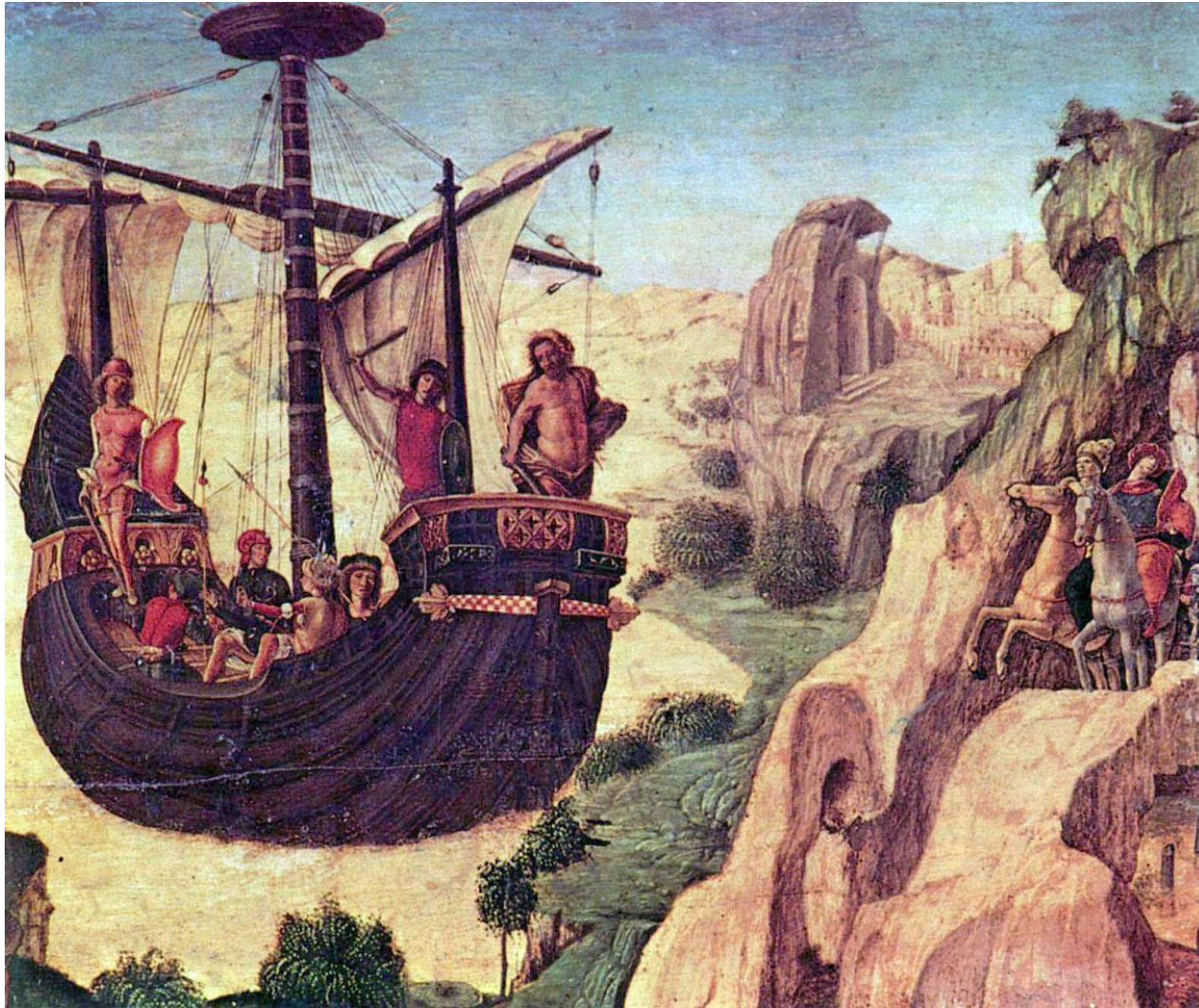
It is a spin-off Version of Jason for IoT and Embedded Multiagent Systems.

# Jason Embedded

It is a spin-off Version of Jason for IoT and Embedded Multiagent Systems.



# 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



# 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 em **Jason** e placas microcontroladas.

# Argo for Jason

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.

# Argo for Jason

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.



# Argo for Jason

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.

# Argo for Jason

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.

# Argo for Jason

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.

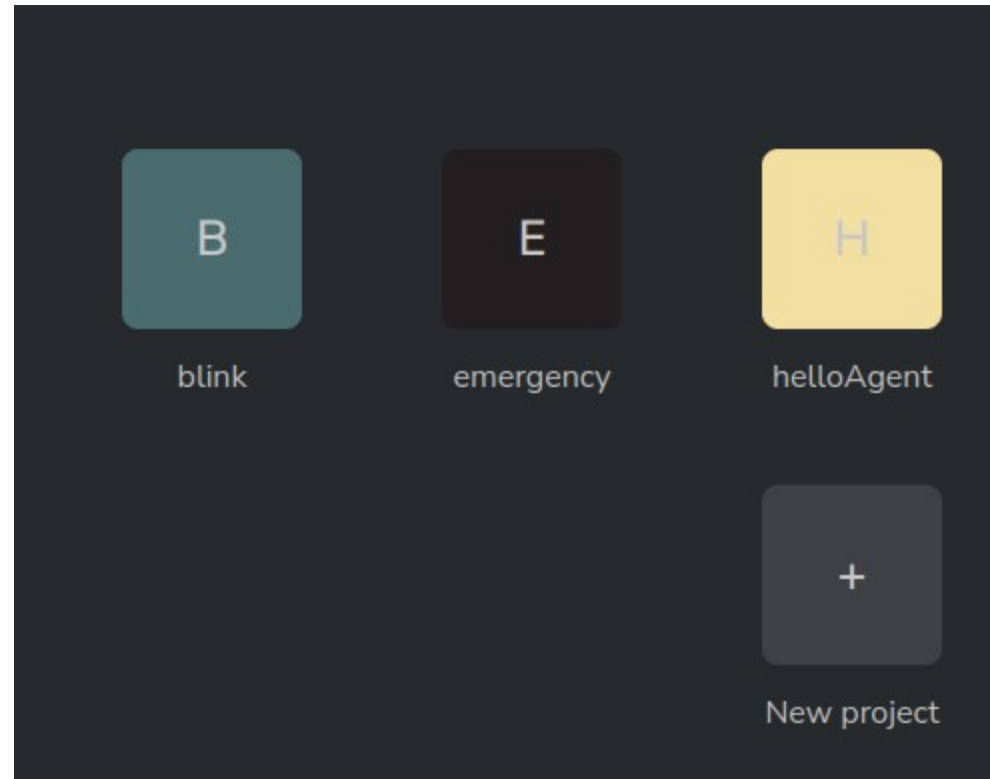
# Argo for Jason

O **ARGO** permite:

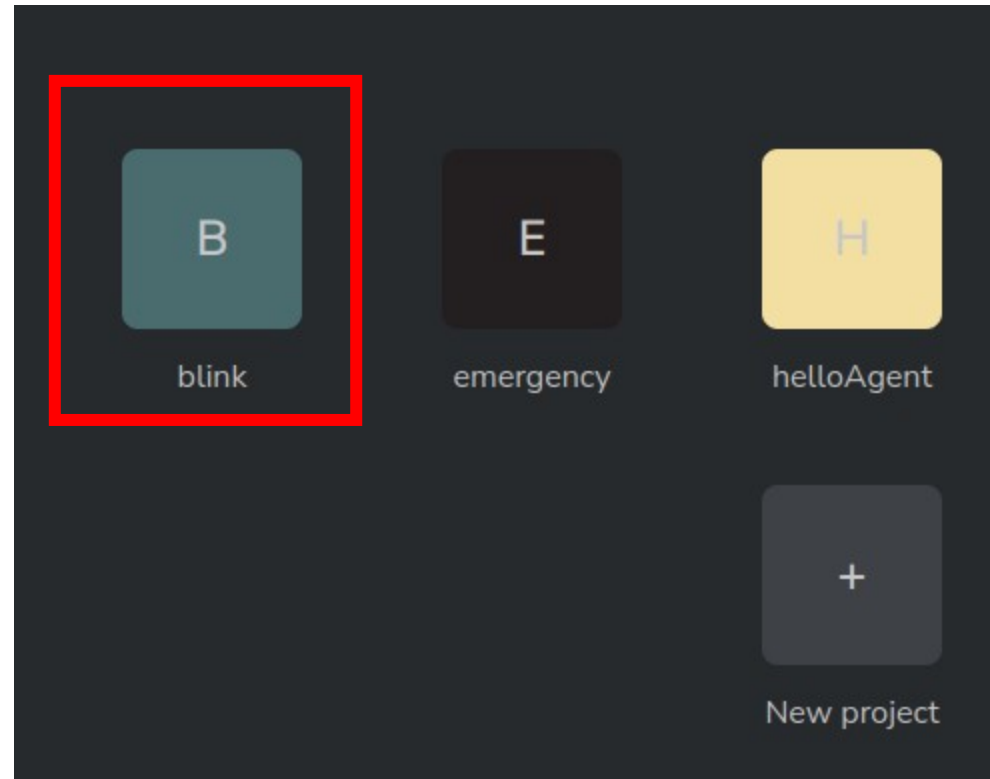
1. Controlar diretamente os atuadores em tempo de execução;
2. Receber percepções dos sensores automaticamente dentro de um período de tempo pré-definido;
3. Mudar os filtros de percepção em tempo de execução;
4. Alterar quais os dispositivos que estão sendo acessados em tempo de execução;
5. Se comunicar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

- **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|close): decide quando perceber ou não o mundo real
  - .act(w): envia ao microcontrolador uma ação para ser executada por um efetuador
  - .change\_filter(filterName): define um filtro de percepção para restringir percepções em tempo real

# Exemplo: blink



# Exemplo: blink

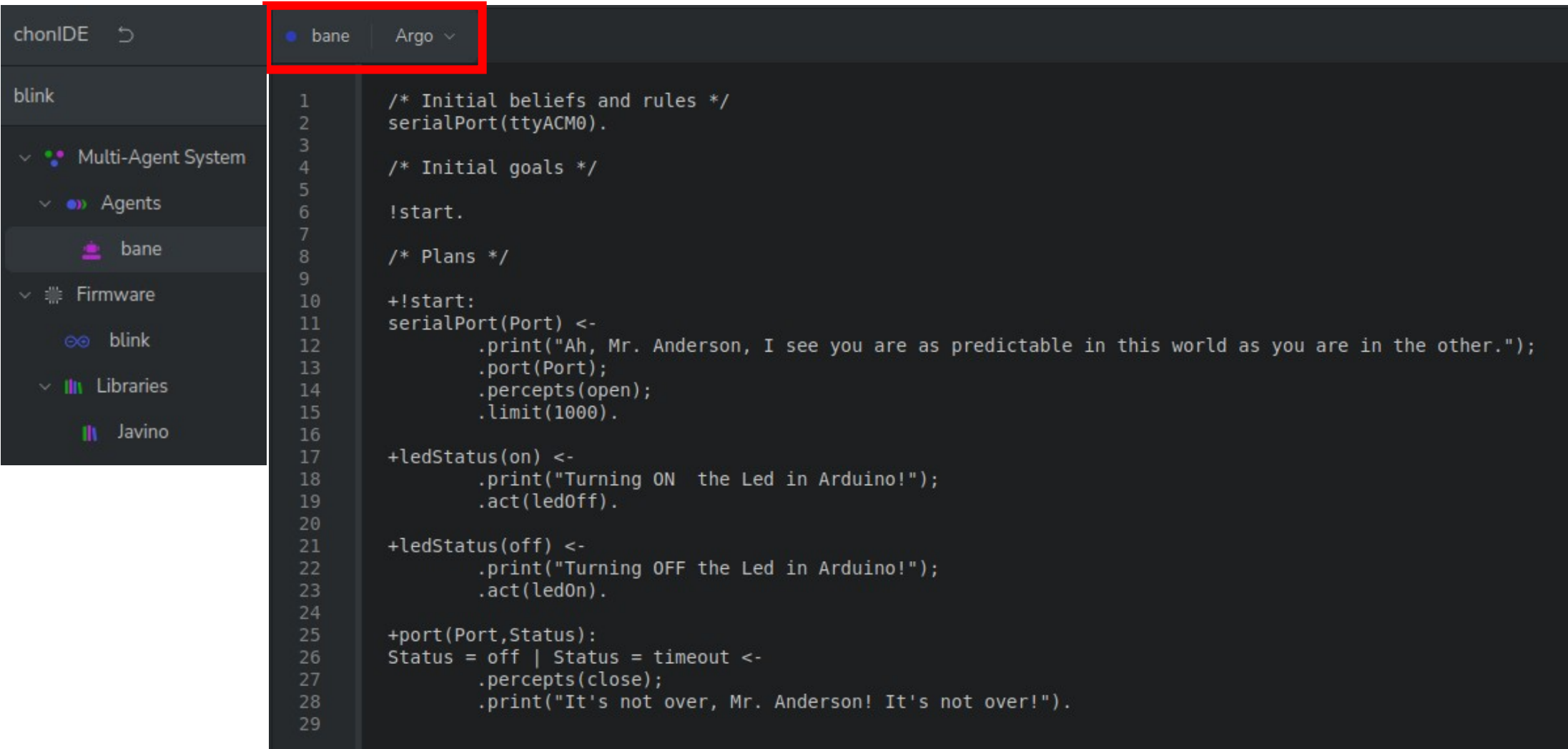




# Exemplo: blink

```
bane  Argo v
1      /* Initial beliefs and rules */
2      serialPort(ttyACM0).
3
4      /* Initial goals */
5
6      !start.
7
8      /* Plans */
9
10     +!start:
11     serialPort(Port) <-
12         .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13         .port(Port);
14         .percepts(open);
15         .limit(1000).
16
17     +ledStatus(on) <-
18         .print("Turning ON the Led in Arduino!");
19         .act(ledOff).
20
21     +ledStatus(off) <-
22         .print("Turning OFF the Led in Arduino!");
23         .act(ledOn).
24
25     +port(Port,Status):
26     Status = off | Status = timeout <-
27         .percepts(close);
28         .print("It's not over, Mr. Anderson! It's not over!").
29
```

# Exemplo: blink



chonIDE

Multi-Agent System

Agents

bane

Firmware

blink

Libraries

Javino

```
1  /* Initial beliefs and rules */
2  serialPort(ttyACM0).
3
4  /* Initial goals */
5
6  !start.
7
8  /* Plans */
9
10 +!start:
11   serialPort(Port) <-
12     .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13     .port(Port);
14     .percepts(open);
15     .limit(1000).
16
17 +ledStatus(on) <-
18   .print("Turning ON the Led in Arduino!");
19   .act(ledOff).
20
21 +ledStatus(off) <-
22   .print("Turning OFF the Led in Arduino!");
23   .act(ledOn).
24
25 +port(Port,Status):
26   Status = off | Status = timeout <-
27     .percepts(close);
28     .print("It's not over, Mr. Anderson! It's not over!");
29
```

# Exemplo: blink

```
bane  Argo v
1      /* Initial beliefs and rules */
2      serialPort(ttyACM0).
3
4      /* Initial goals */
5
6      !start.
7
8      /* Plans */
9
10     +!start:
11     serialPort(Port) <-
12         .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13         .port(Port);
14         .percepts(open);
15         .limit(1000).
16
17     +ledStatus(on) <-
18         .print("Turning ON the Led in Arduino!");
19         .act(ledOff).
20
21     +ledStatus(off) <-
22         .print("Turning OFF the Led in Arduino!");
23         .act(ledOn).
24
25     +port(Port,Status):
26     Status = off | Status = timeout <-
27         .percepts(close);
28         .print("It's not over, Mr. Anderson! It's not over!").
29
```

# Exemplo: blink

```
bane  Argo v
1      /* Initial beliefs and rules */
2      serialPort(ttyACM0).
3
4      /* Initial goals */
5
6      !start.
7
8      /* Plans */
9
10     +!start:
11     serialPort(Port) <-
12         .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13         .port(Port);
14         .percepts(open);
15         .limit(1000).
16
17     +ledStatus(on) <-
18         .print("Turning ON the Led in Arduino!");
19         .act(ledOff).
20
21     +ledStatus(off) <-
22         .print("Turning OFF the Led in Arduino!");
23         .act(ledOn).
24
25     +port(Port,Status):
26     Status = off | Status = timeout <-
27         .percepts(close);
28         .print("It's not over, Mr. Anderson! It's not over!");
29
```

# Exemplo: blink

```
bane  Argo v
1      /* Initial beliefs and rules */
2      serialPort(ttyACM0).
3
4      /* Initial goals */
5
6      !start.
7
8      /* Plans */
9
10     +!start:
11     serialPort(Port) <-
12         .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13         .port(Port);
14         .percepts(open);
15         .limit(1000).
16
17     +ledStatus(on) <-
18         .print("Turning ON the Led in Arduino!");
19         .act(ledOff).
20
21     +ledStatus(off) <-
22         .print("Turning OFF the Led in Arduino!");
23         .act(ledOn).
24
25     +port(Port,Status):
26     Status = off | Status = timeout <-
27         .percepts(close);
28         .print("It's not over, Mr. Anderson! It's not over!");
29
```

# Exemplo: blink

```
bane  Argo v
1      /* Initial beliefs and rules */
2      serialPort(ttyACM0).
3
4      /* Initial goals */
5
6      !start.
7
8      /* Plans */
9
10     +!start:
11     serialPort(Port) <-
12         .print("Ah, Mr. Anderson, I see you are as predictable in this world as you are in the other.");
13         .port(Port);
14         .percepts(open);
15         .limit(1000).
16
17     +ledStatus(on) <-
18         .print("Turning ON the Led in Arduino!");
19         .act(ledOff).
20
21     +ledStatus(off) <-
22         .print("Turning OFF the Led in Arduino!");
23         .act(ledOn).
24
25     +port(Port,Status):
26     Status = off | Status = timeout <-
27         .percepts(close);
28         .print("It's not over, Mr. Anderson! It's not over!").
29
```



# Exemplo: blink

```
blink  ✓ Compile  ↑ Deploy
1      #include <Javino.h>
2      Javino javino;
3
4      void serialEvent(){
5          /*
6           * The serialEvent() function handles interruptions coming from the serial port.
7           *
8           * NOTE: The serialEvent() feature is not available on the Leonardo, Micro, or other ATmega32U4 based boards.
9           * https://docs.arduino.cc/built-in-examples/communication/SerialEvent
10          */
11      }
12      javino.readSerial();
13  }
14
15  void setup() {
16      javino.start(9600);
17      pinMode(13,OUTPUT);
18  }
19
20  void loop() {
21      if(javino.availableMsg()){
22          if(javino.getMsg() == "getPercepts")javino.sendMsg(getPercepts());
23          else if(javino.getMsg() == "ledOn") ledOn();
24          else if(javino.getMsg() == "ledOff")ledOff();
25      }
26  }
27
28  /* It sends the exogenous environment's perceptions to the agent. */
29  String getPercepts(){
30      String beliefs =
31          "resourceName(myArduino);" +
32          getLedStatus();
```



# Exemplo: Argo



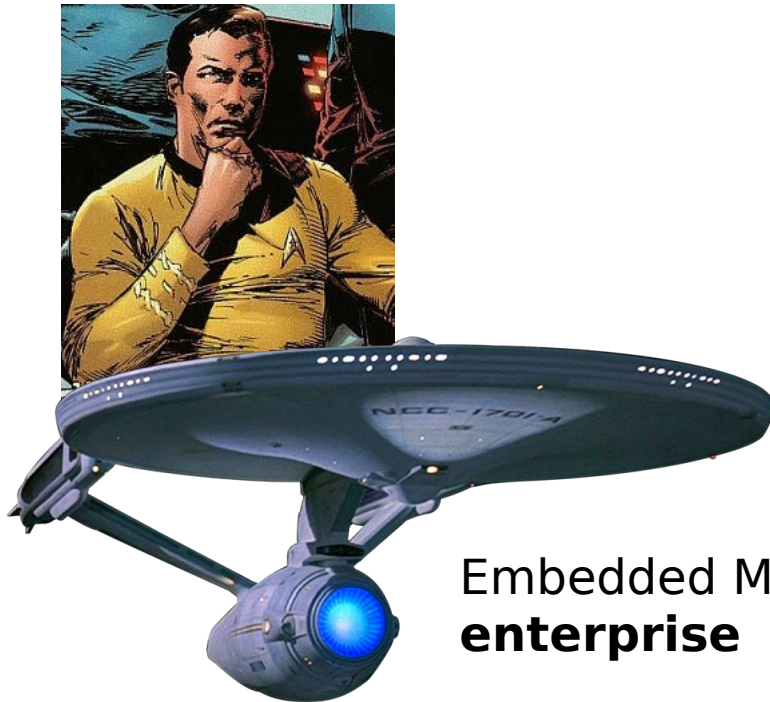
# Exemplo: Argo



Embedded MAS  
**enterprise**

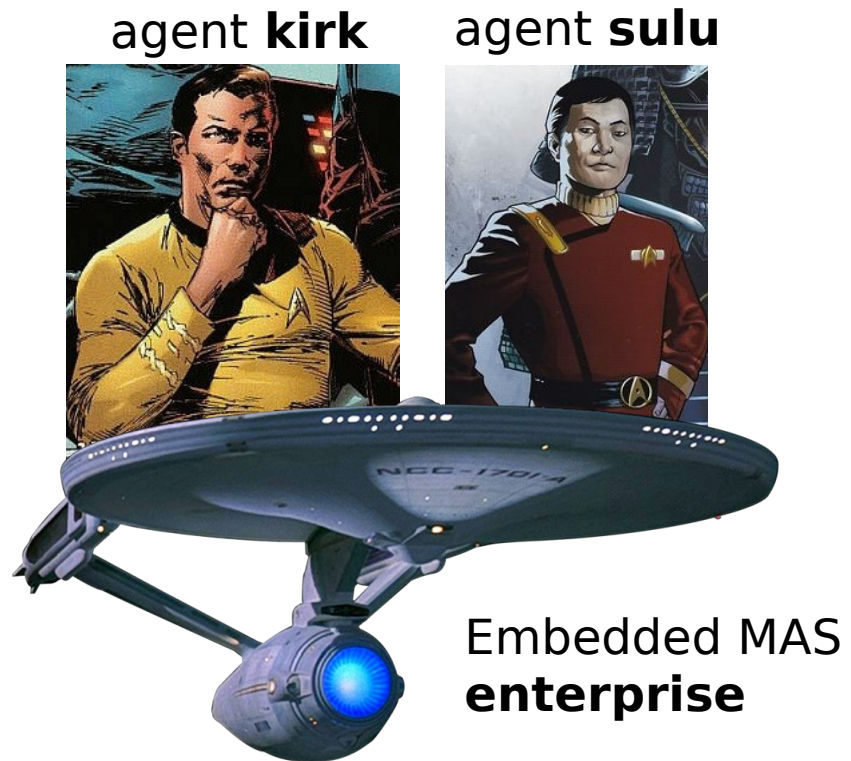
# Exemplo: Argo

agent **kirk**



Embedded MAS  
**enterprise**

# Exemplo: Argo



# Exemplo: Argo



# Exemplo: Argo

Sulu...  
Fire Photon  
Torpedo!





# Exemplo: Argo

Sulu...  
Fire Photon  
Torpedo!



Target Acquired!





# Exemplo: Argo

Sulu...  
Fire Photon  
Torpedo!



Target Acquired!

Photon torpedo fired.



# Exemplo: Argo

`.send(sulu,  
achieve, fire);`



# Exemplo: Argo

`.send(sulu,  
achieve, fire);`



`.port(ttyACM0);`



# Exemplo: Argo

`.send(sulu,  
achieve, fire);`



`.port(ttyACM0);`

`.act(buzzerOn);  
.act(buzzerOff);`



# Argo for Jason: Limitações

## Algumas características:

- Limite de 127 portas seriais
  - O limite da USB.
- Uma porta de cada vez
  - Sem competição de porta para evitar conflitos.
  - 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.



# Exemplo: Argo .act()



kirk

```
1      /* Initial beliefs and rules */
2
3      /* Initial goals */
4      !start.
5
6      /* Plans */
7
8      +!start <-
9          .print("This is Comma
10         .wait(200);
11         +ship(klingow).
12
13
14         +ship(Ship): Ship == klingow <-
15             .print("Sulu, fire the photon torpedo.");
16             .send(sulu, achieve, fire).
17
```

sulu

```
1      /* Initial beliefs and rules */
2
3      /* Initial goals */
4      !start.
5
6      /* Plans */
7
8      +!start <-
9          .port(ttyACM0).
10
11      +!fire <-
12          .print("Target Acquired!");
13          .act(buzzerOn);
14          .wait(100);
15          .act(buzzerOff);
16          .print("Photon torpedo fired.");
17
```

<https://sourceforge.net/p/chonos/examples/ci/master/tree/05-embeddedMultiAgent/argoAgentExample01/>

# Exemplo: Argo .percepts()



## Agents

- kirk
- sulu

by Jason

latest state 2 1 clear history

## Inspection of agent **sulu**

### - Beliefs

```
breakLStatus(off)[source(percept)].  
buzzerStatus(off)[source(percept)].  
distance(63)[source(percept)].  
ledStatus(off)[source(percept)].  
lightStatus(off)[source(percept)].  
lineLeft(1008)[source(percept)].  
lineRight(1006)[source(percept)].  
luminosity(198)[source(percept)].  
motorStatus(stopped)[source(percept)].
```

```
sulu  
1      /* Initial beliefs and rules */  
2  
3      /* Initial goals */  
4      !start.  
5  
6      /* Plans */  
7  
8      +!start <-  
9          .port(ttyACM0);  
10         .percepts(open).  
11
```



## OBRIGADO!

pantoja@cefet-rj.br  
nilson.lazarin@cefet-rj.br

