

Paradigmas de Programação

Prof. Maicon R. Zatelli

Programação Orientada a Agentes

Universidade Federal de Santa Catarina
Florianópolis - Brasil

2018/1

Sumário

- 1 Introdução
- 2 Engenharia de Software
- 3 Linguagens de Programação
- 4 Competições
- 5 Linguagem Jason
 - Jason x Java
 - Ferramentas
 - Demonstração
- 6 Conclusões



Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 Linguagem Jason

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Introdução

Programação Funcional

- Lisp (1958), Erlang (1986), Haskell (1990)

Programação Lógica

- Prolog (1972)

Programação Estruturada

- C (1972), Pascal (1970)

Programação Orientada a Objetos

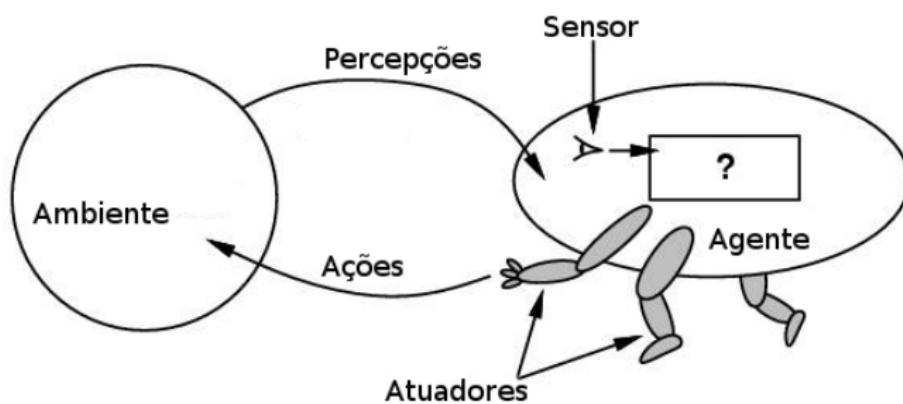
- Java (1995), Object Pascal (1995), C# (2001), VB.NET (2001)

Programação Orientada a Agentes

- Agent0 (1993), AgentSpeak(L) (1996), Jason (2007), 2APL (2008), GOAL (2009)

Introdução

Um agente é um sistema computacional que é situado em algum ambiente e é capaz de agir de forma *autônoma* nesse ambiente a fim de cumprir seus objetivos de projeto. O agente obtém dados do ambiente através de sensores e produz, como saída, ações que o afetam (Wooldridge).



Agentes x Objetos

Agentes estão sempre em execução

- Possuem em ciclo de raciocínio executado continuamente

	Encapsulamento Comportamento	Encapsulamento Estado	Encapsulamento Controle
Estruturada	✓		
Orientada a Objetos	✓	✓	
Orientada a Agentes	✓	✓	✓

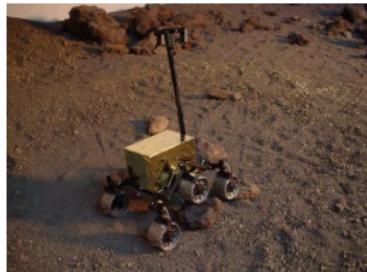
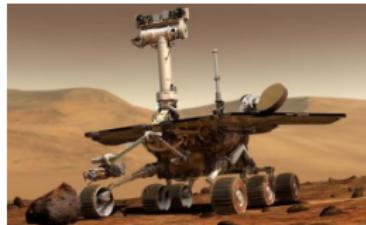
Veículos Aéreos Autônomos (UAVs)



Robôs Companheiros



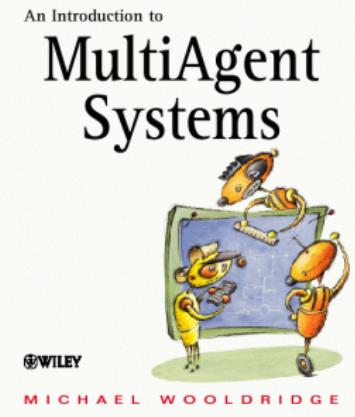
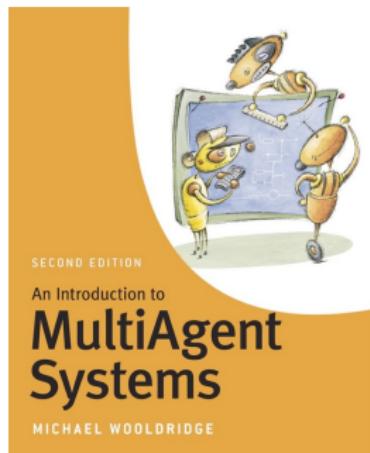
Exploração Planetária



Resgates



Introdução - Livros



Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 Linguagem Jason

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Engenharia de Software (AOSE)

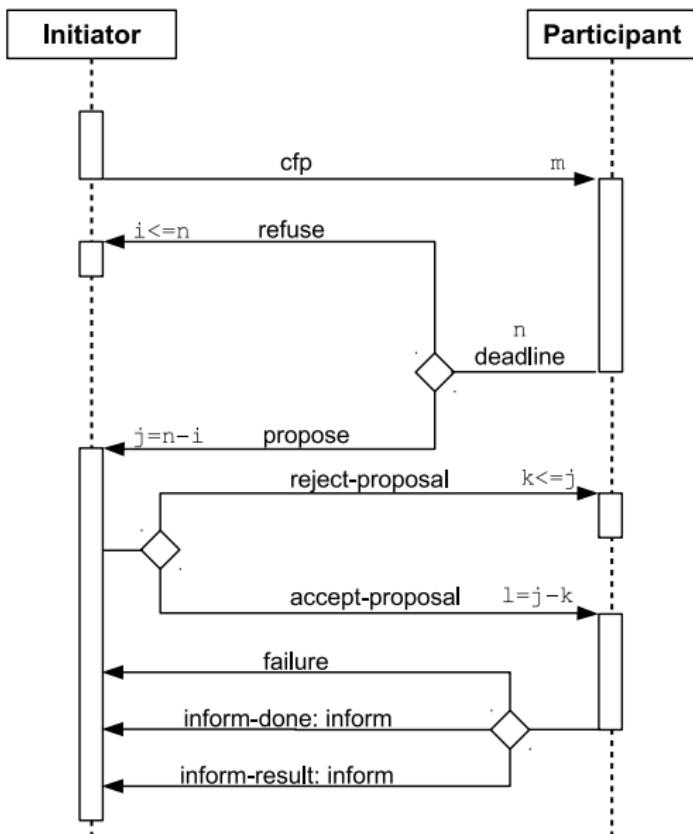
Metodologias

- Prometheus (Padgham, Winikoff, et al.)
- Gaia (Wooldridge, Jennings, Kinny, Zambonelli)
- Tropos (Giorgini, Mylopoulos, et al.)
- ...

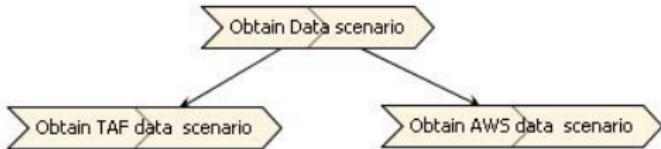
Linguagens de Modelagem

- MAS-ML (Viviane T. da Silva, Carlos Lucena, et al.)
- AUML (James Odell, H. Van Dyke Parunak, Bernhard Bauer)
- ...

Engenharia de Software - Interação (AUML)



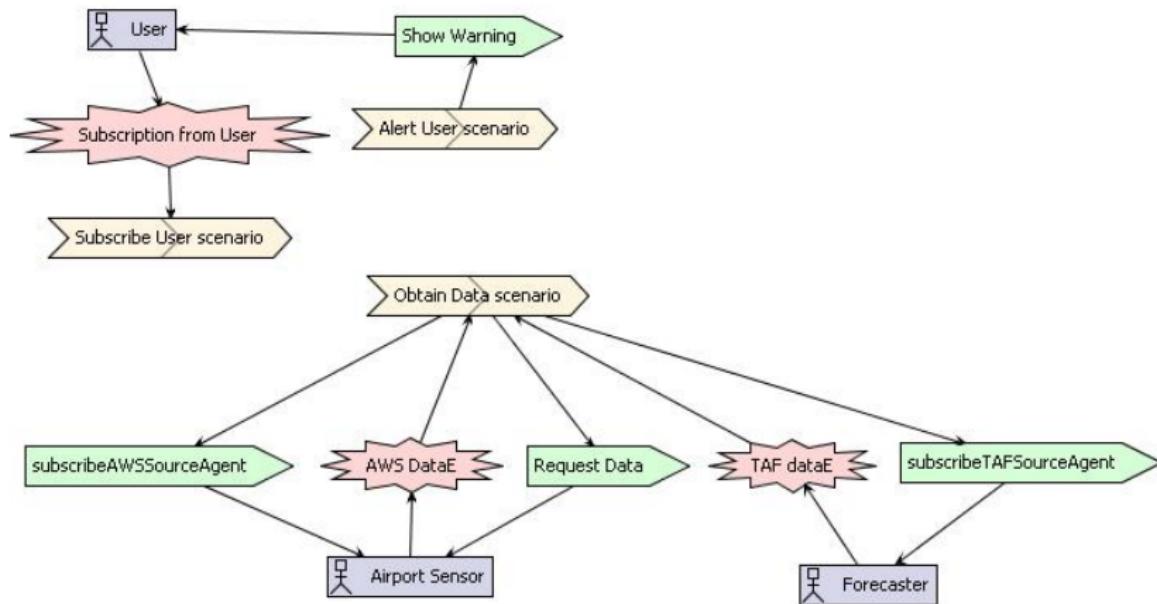
Engenharia de Software - Cenários (Prometheus)



Fonte:

<https://sites.google.com/site/rmitagents/software/prometheusPDT/tutorials/meteorology>

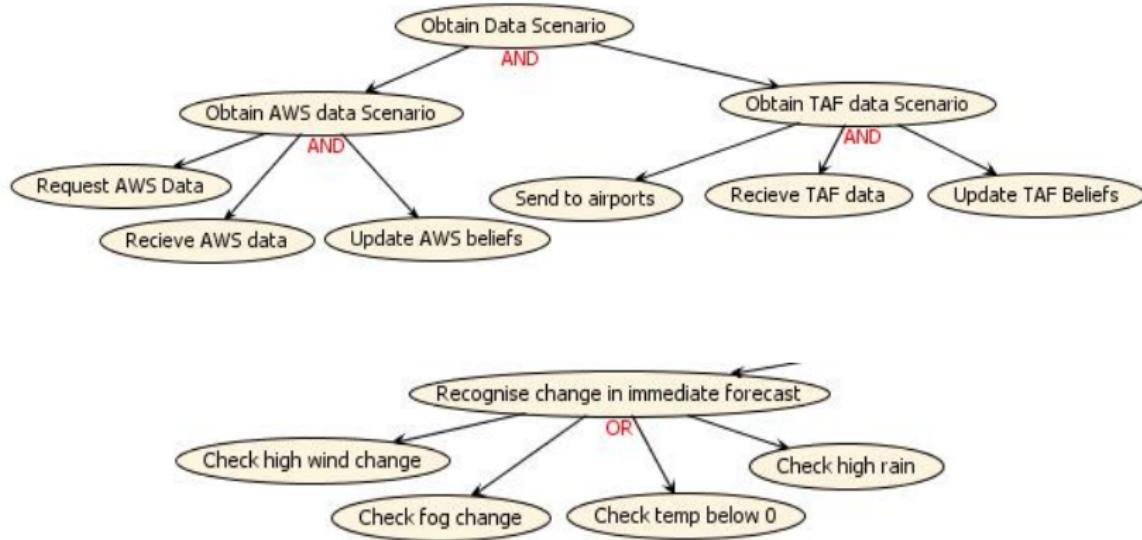
Engenharia de Software - Visão Geral Análise (Prometheus)



Fonte:

<https://sites.google.com/site/rmitagents/software/prometheusPDT/tutorials/meteorology>

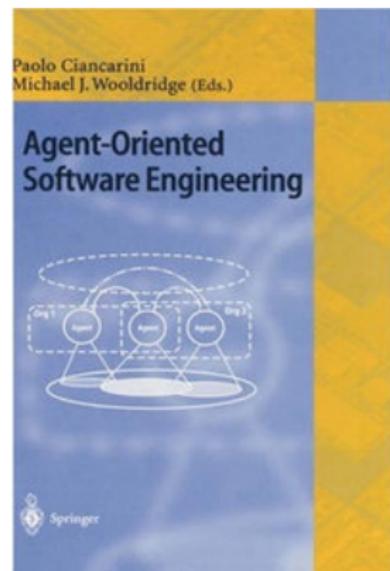
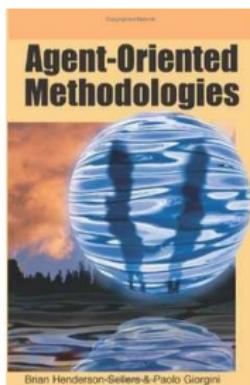
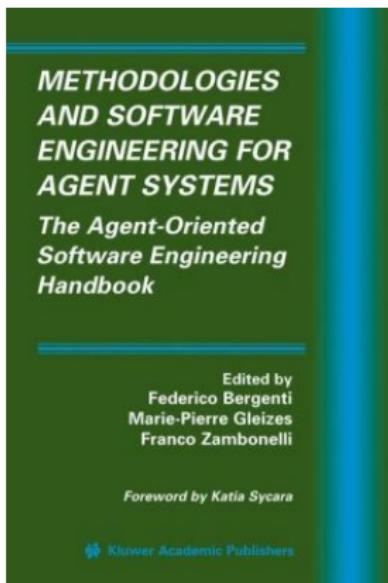
Engenharia de Software - Visão Geral Metas (Prometheus)



Fonte:

<https://sites.google.com/site/rmitagents/software/prometheusPDT/tutorials/meteorology>

Engenharia de Software - Livros



Sumário

- 1 Introdução
- 2 Engenharia de Software
- 3 Linguagens de Programação
- 4 Competições
- 5 Linguagem Jason
 - Jason x Java
 - Ferramentas
 - Demonstração
- 6 Conclusões

Linguagens

[Jason](#) (Hübner, Bordini, ...); [3APL](#) e [2APL](#) (Dastani, van Riemsdijk, Meyer, Hindriks, ...); [Jadex](#) (Braubach, Pokahr); [MetateM](#) (Fisher, Guidini, Hirsch, ...); [ConGoLog](#) (Lesperance, Levesque, ...); [DTGolog](#) (Boutilier); [Teamcore/MTDP](#) (Milind Tambe, ...); [IMPACT](#) (Subrahmanian, Kraus, Dix, Eiter); [CLAIM](#) (Amal El Fallah-Seghrouchni, ...); [GOAL](#) (Hindriks); [BRAHMS](#) (Sierhuis, ...); [SemantiCore](#) (Blois, ...); [STAPLE](#) (Kumar, Cohen, Huber); [Go!](#) (Clark, McCabe); [Bach](#) (John Lloyd, ...); [MINERVA](#) (Leite, ...); [SOCS](#) (Torroni, Stathis, Toni, ...); [FLUX](#) (Thielscher); [JIAC](#) (Hirsch, ...); [JADE](#) (Agostino Poggi, ...); [JACK](#) (AOS); [Agentis](#) (Agentis Software); [Jackdaw](#) (Calico Jack); [AF-AgentSpeak](#) e [ASTRA](#) (Rem Collier); ...

Linguagens - Sites

Jason - Baseada em AgentSpeak(L)

- <http://jason.sf.net/>

GOAL - Linguagem própria

- <http://mmi.tudelft.nl/trac/goal>

2APL - Linguagem própria

- <http://apapl.sourceforge.net/>

JIAC - Baseada em Java

- <http://www.jiac.de/>

Jadex - Baseada em Java/XML

- <http://www.activecomponents.org/>

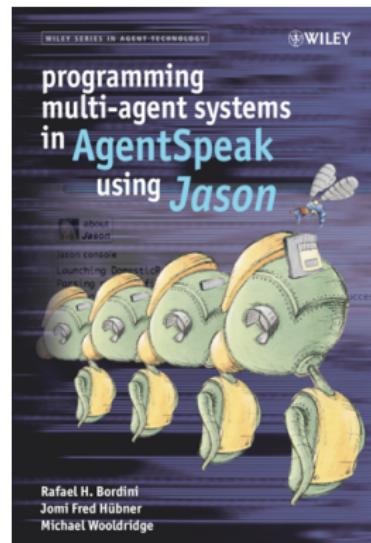
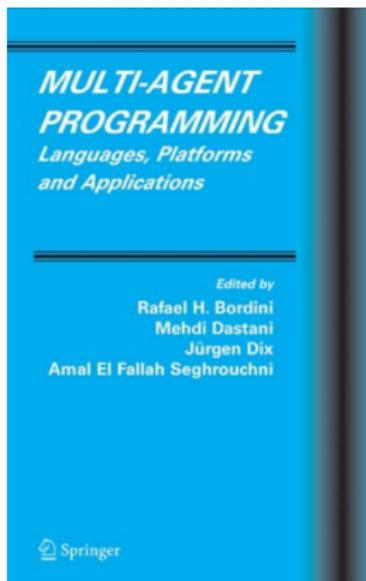
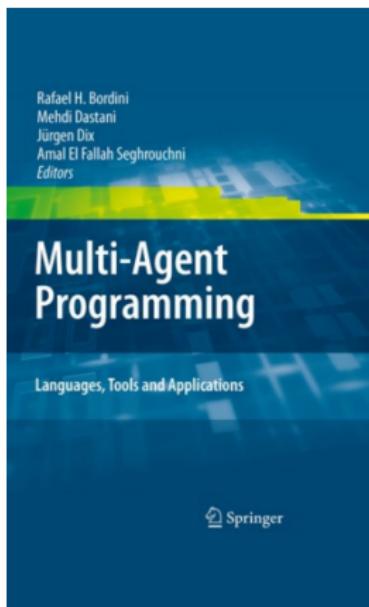
JADE - Baseada em Java

- <http://jade.tilab.com/>

JACK - Baseada em Java

- <http://www.agent-software.com.au/products/jack/>

Linguagens - Livros



Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 Linguagem Jason

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Competições

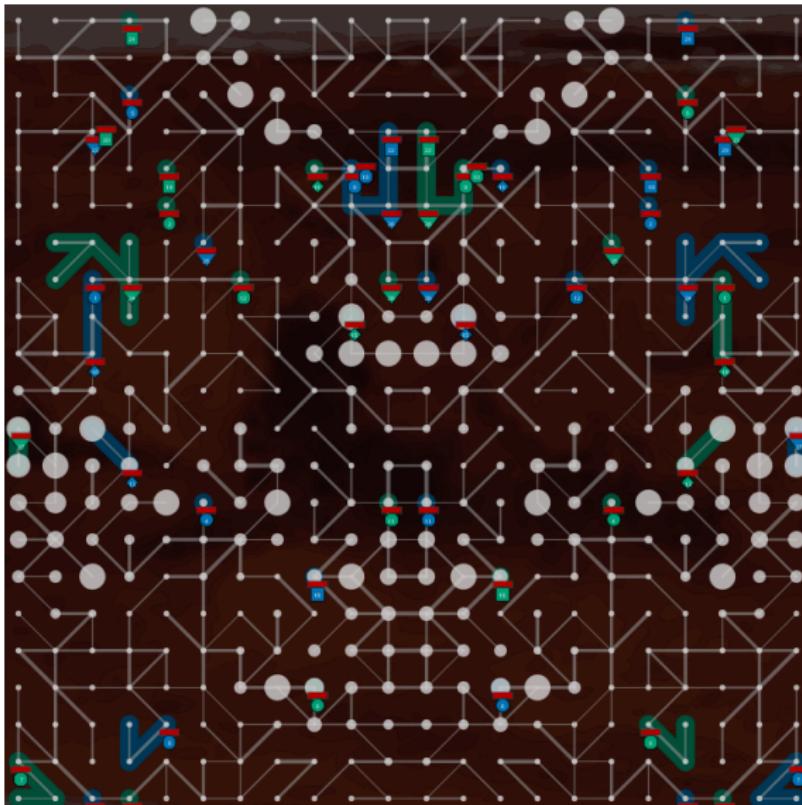
Multi-Agent Programming Contest (MAPC)

- <http://multiagentcontest.org/>

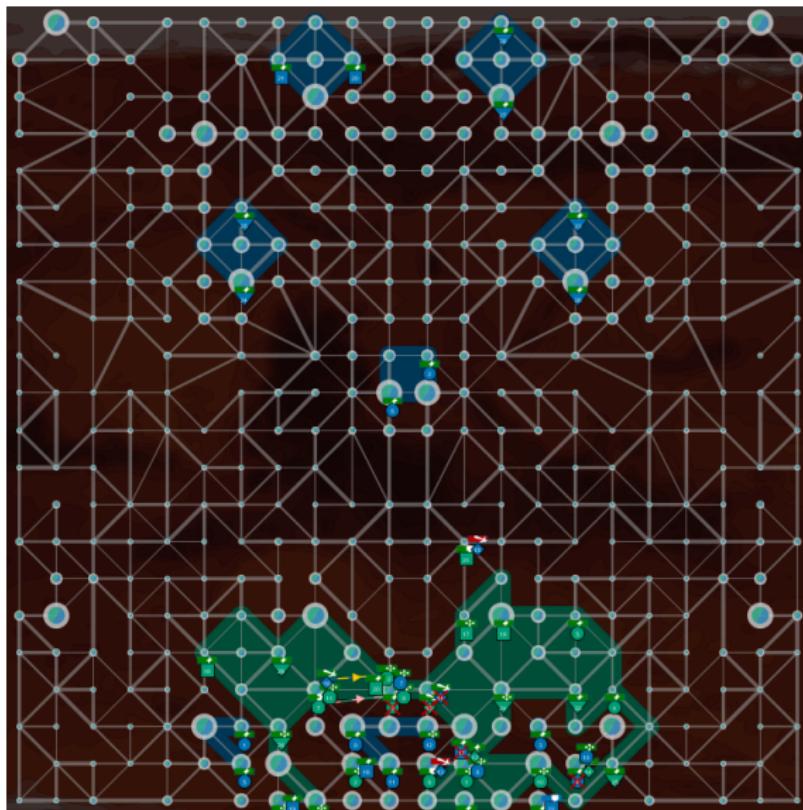
Robocup

- <http://www.robocup.org/>

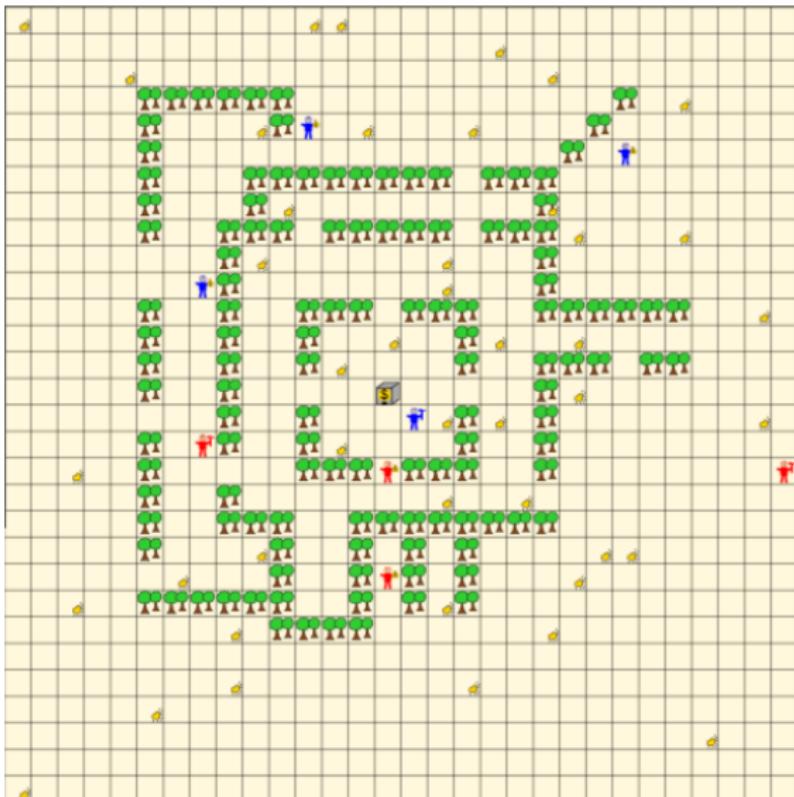
Multi-Agent Programming Contest



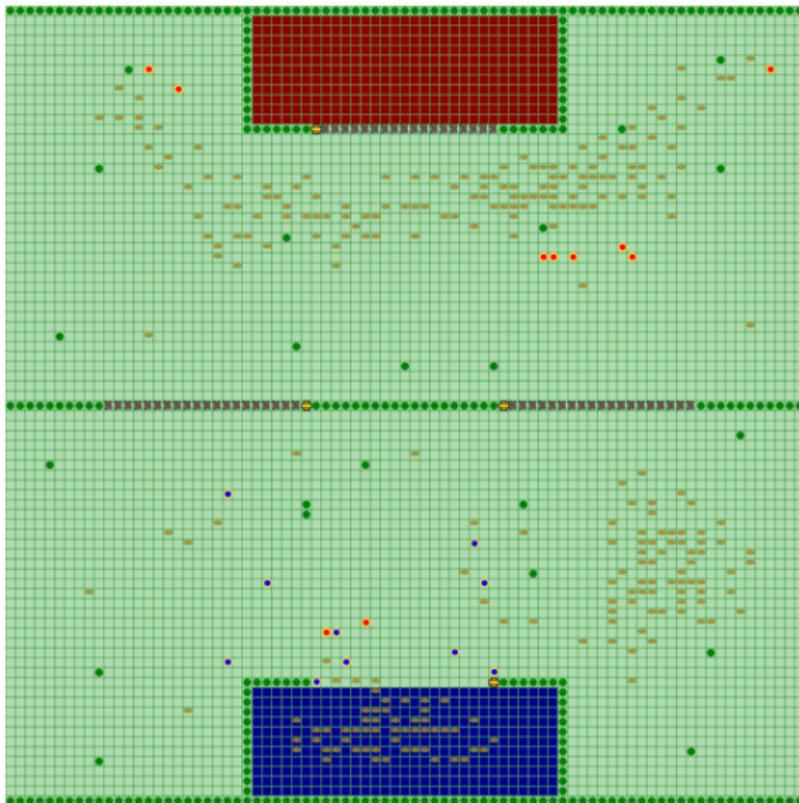
Multi-Agent Programming Contest



Multi-Agent Programming Contest



Multi-Agent Programming Contest



Robocup Soccer



Robocup Soccer



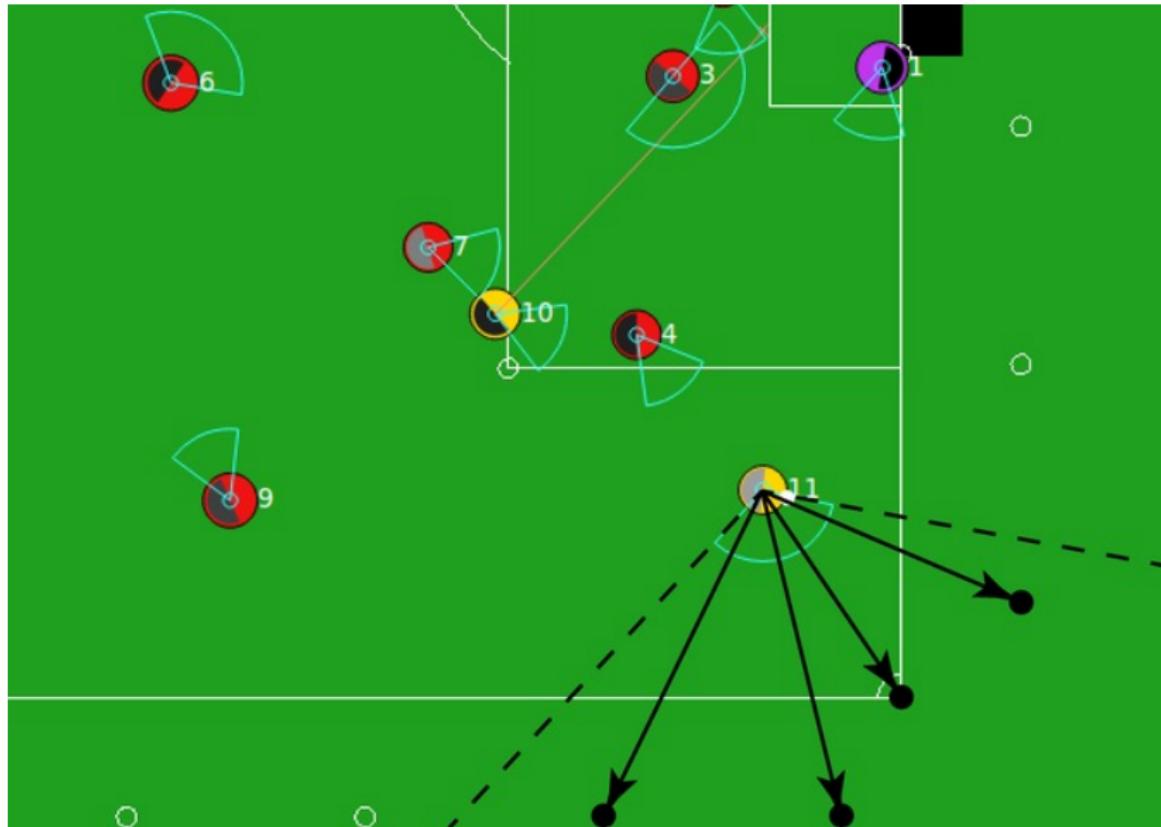
Robocup Soccer



Robocup Soccer



Robocup Soccer



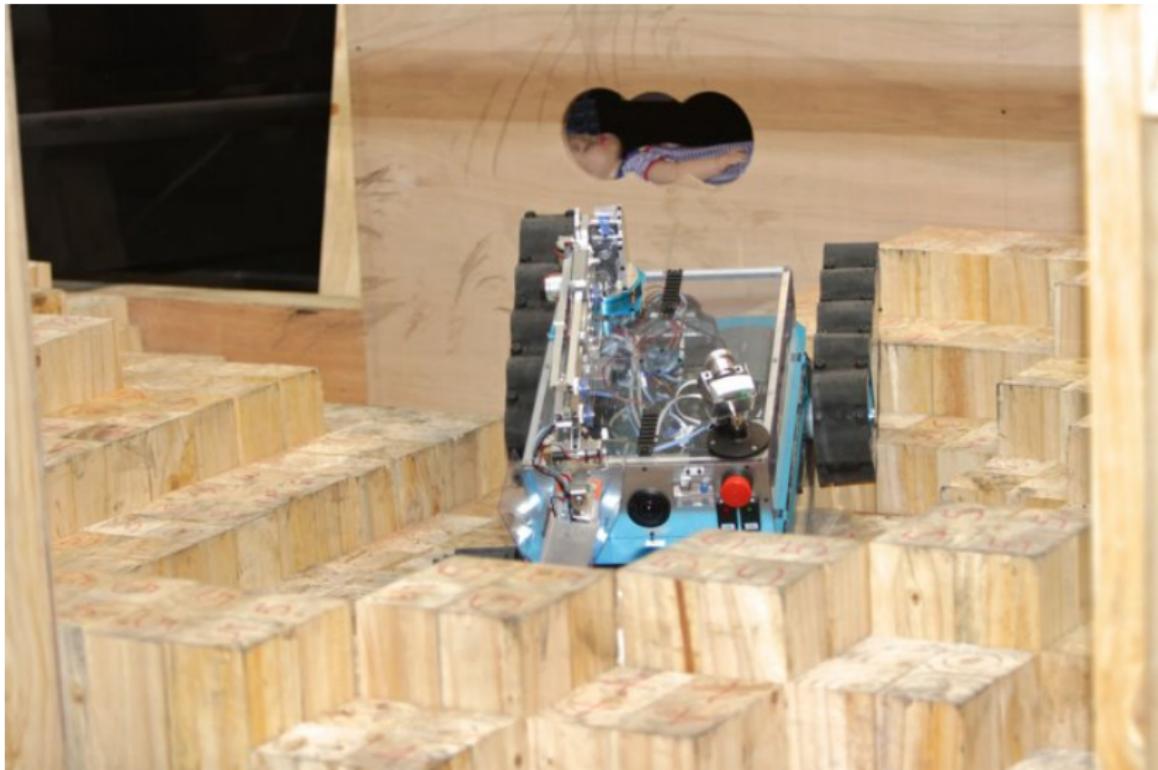
Robocup Rescue



Robocup Rescue



Robocup Rescue



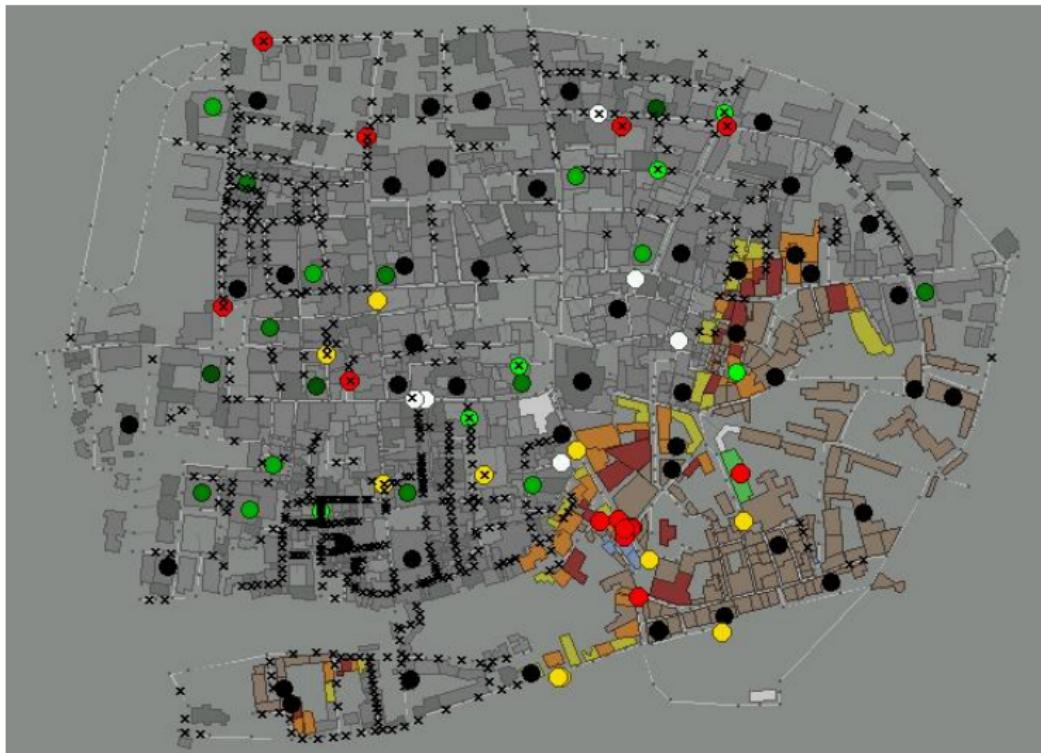
Robocup Rescue



Robocup Rescue



Robocup Rescue



Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 **Linguagem Jason**

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Jason - Conceitos Básicos

Crenças (*Beliefs*): representa as informações disponíveis para o agente (ex: sobre o ambiente ou outros agentes)

Metas (*Goals*): representa o estado das coisas que os agentes querem atingir

Planos (*Plans*): é a especificação das sequências de ações que os agentes fazem para atingir uma meta

Eventos (*Events*): acontecem como consequências de mudanças que ocorrem nas crenças ou metas do agente (ex: algo que pode mudar no ambiente)

Intenções (*Intentions*): são os planos instanciados para atingir alguma meta

Jason - Crenças (*Beliefs*)

```
pai(ana,antonio).  
pai(bob,antonio).  
pai(clara,robert).  
pai(antonio,alfredo).  
pai(robert,carlos).  
pai(alfredo,fred).  
~pai(amanda,fred).
```

Regras

```
avo(X,AvoX) :- pai(X, PaiX) & pai(PaiX, AvoX).  
irmao(X,Y) :- pai(X, PaiAmbos) & pai(Y, PaiAmbos) & X \== Y.
```

Jason - Crenças (*Beliefs*) - Atualizadas por Percepção

Crenças que são percebidas pelo ambiente possuem anotação `source(percept)` e são automaticamente atualizadas quando o ambiente também muda

Bob percebe que semáforo mudou de amarelo para vermelho

```
//remove semaforo(amarelo) [source(percept)] das crenças de Bob  
//adiciona semaforo(vermelho) [source(percept)] nas crenças de Bob
```

Jason - Crenças (*Beliefs*) - Atualizadas por Comunicação

Quando um agente “diz” algo para outro agente, ele atualiza crenças no agente destinatário

Bob envia mensagens para Tom

```
.send(tom, tell, quadrado(5, 25));  
//adiciona quadrado(5, 25) [source(bob)] nas crenças de Tom  
  
.send(tom, untell, quadrado(5, 25));  
//remove quadrado(5, 25) [source(bob)] das crenças de Tom
```

Jason - Crenças (*Beliefs*) - Atualizadas pelo Agente

Um agente pode atualizar as próprias crenças utilizando operadores + e -

Bob faz algumas operações

```
+quadrado(4,16); //acredito que o quadrado de 4 é 16
//adiciona quadrado(4,16) [source(self)] nas crenças de Bob

-pai(ana,bob); //não acredito que bob é pai de ana
//remove pai(ana,bob) [source(self)] das crenças de Bob

+~chuva; //acredito que não está chovendo
//adiciona ~chuva[source(self)] nas crenças de Bob
```

Jason - Metas (*Goals*)

Implementa metas para atingir (*achievement goals*) e metas de teste/verificação (*test goals*)

Achievement goals

Utiliza o operador !

```
!encontre(ouro) //quero encontrar ouro
!escreva(livro) //quero escrever um livro
!fibonacci(8) //quero calcular fibonacci(8)
```

Test goals

Utiliza o operador ?

```
?temperatura(T) //quero descobrir a temperatura
?pai(ana, P) //quero descobrir quem é o pai de ana
?chuva //quero descobrir se está chovendo
```

Jason - Metas (*Goals*) - Atualizadas por Intenções

É possível adicionar novas metas através dos operadores ! e ?

Achievement goals

Utiliza o operador !

```
!encontre(ouro) //adiciona a meta !encontre(ouro) [source(self)]  
!escreva(livro) //adiciona a meta !escreva(livro) [source(self)]  
!fibonacci(8) //adiciona a meta !fibonacci(8) [source(self)]
```

Test goals

Utiliza o operador ?

```
?temperatura(T) //adiciona a meta ?temperatura(T) [source(self)]  
?pai(ana, P) //adiciona a meta ?pai(ana, P) [source(self)]  
?chuva //adiciona a meta ?chuva [source(self)]
```

Jason - Metas (*Goals*) - Atualizadas por Comunicação

Em *achievement goals*, quando um agente “pede” para que outro agente faça algo, ele atualiza as metas no agente destinatário

Bob “pede” para Tom fazer (ou deixar de fazer) alguma coisa

```
.send(tom, achieve, encontre(ouro));  
//adiciona !encontre(ouro) [source(bob)] nas metas de Tom  
  
.send(tom, unachieve, encontre(prata));  
//remove !encontre(prata) [source(bob)] das metas de Tom
```

Jason - Metas (*Goals*) - Atualizadas por Comunicação

Em *test goals*, quando um agente “pede” alguma informação para outro agente, ele atualiza as metas no agente destinatário

Bob pede algumas informações para Tom

```
.send(tom, askOne, temperatura(T), RespostaTom);
//adiciona ?temperatura(T) [source(bob)] nas metas de Tom
//a resposta de Tom irá unificar com RespostaTom

.send(tom, askAll, amigo(A), RespostaTom);
//adiciona ?amigo(A) [source(bob)] nas metas de Tom
//a lista de respostas de Tom irá unificar com RespostaTom
```

Jason - Eventos

Acontecem quando as crenças ou metas do agente são atualizadas

Tratamento de Eventos

- +b adição de uma crença
- b remoção de uma crença
- +!g adição de *achievement goal*
- !g remoção de *achievement goal*
- +?g adição de *test goal*
- ?g remoção de *test goal*

Jason - Planos

Linguagem declarativa

```
evento : contexto <- corpo do plano
```

- **Evento:** denota o evento que aquele plano deve tratar
- **Contexto:** representa circunstâncias/condições que o plano pode ser utilizado
- **Corpo do plano:** sequência de ações para executar

Exemplo

```
+chuva: janela(aberta) & emCasa <- fechar(janela).  
  
+chuva: not emCasa <- .send(ana, achieve, fechar(janela)).  
  
+temperatura(T): T > 25 <- +quente;abrir(janela).  
  
+!encher(copo): not copo(cheio) <- abrir(torneira); !encher(copo).  
+!encher(copo) <- fechar(torneira).
```

Jason - Planos

Exemplo

```
@chuvaPlan
+chuva : horaSaida(T) & relogio.hora(H) & H >= T <-
    !g1;                                // nova sub-meta
    !!g2;                               // nova meta
    ?b(X);                             // nova meta de teste
    +b1(T-H);                          // adiciona uma crença
    -b2(T-H);                          // remove uma crença
    -+b3(T*H);                         // atualiza uma crença (remove e adiciona)
    jia.getMoedas(X);                  // ação interna (agente)
    X > 10;                            // restrição para continuar com o plano
    fechar(janela);                   // ação externa (ambiente)
    fechar(porta);                    // ação externa (ambiente)
    +~emCasa;                          // adiciona uma crença
    .print("Partiu!");                // ação interna (agente)
```

Jason - Planos - Atualizados por Comunicação

Da mesma forma que uma crença, um agente pode “dizer” para outro como realizar certas tarefas

Bob envia mensagens para Tom

```
.send(tom, tellHow, { @somaPlan +!soma(A,B) <-
                      .print(A, "+",B,"=",A+B)
                    });
//adiciona o plano somaPlan nos planos de Tom

.plan_label(Plan,somaPlan);
//procura a especificação do plano somaPlan
.send(tom, tellHow, Plan);
//adiciona o plano somaPlan nos planos de Tom

.send(tom, untellHow, somaPlan);
//remove o plano somaPlan dos planos de Tom
```

Jason - Planos - Atualizados por Comunicação

É possível um agente “pedir” para outro agente a respeito dos planos

Tom pede alguns planos para Bob

```
.send(bob, askHow, {+!soma(_,_)}, Resposta);
//Tom pergunta sobre o plano para somar dois valores
.add_plan(Resposta);
//Tom adiciona os planos enviados por Bob

/*Quando Tom não sabe executar algum plano
ele sempre pergunta para Bob*/
-!NoPlan[error(no_relevant)] <-
    .send(bob, askHow, {+!NoPlan}, Resultado);
    .add_plan(Resultado);
!NoPlan.
```

Jason - Operadores Booleanos e Aritméticos

Operadores Booleanos

& e
| ou
not não
= unificação
== igual
\== diferente
>, >= maior, maior/igual
<, <= menor, menor/igual

Operadores Aritméticos

+ soma
- subtração
* multiplicação
/ divisão
div divisão inteira
mod resto
** potenciação

Jason - Ações Internas (*Internal Actions*)

Definidas em Java, servem para ajudar o agente e não afetam o ambiente

.abolish apaga crenças de certo tipo

.findall encontra lista de crenças de certo tipo

.setof encontra conjunto de crenças de certo tipo

.count conta crenças de certo tipo

.send envia mensagem

.broadcast envia mensagem para todos

.my_name nome do agente

.wait espera algum evento

.stopMAS finaliza execução

math.random números randômicos

math.round arredondamento

system.time recupera a hora

.member membros lista

.length tamanho lista

.nth n-ésimo elemento lista

.union união de conjuntos

.concat concatenação de listas

.max maior elemento lista

.min menor elemento lista

... ...

Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 Linguagem Jason

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Robôs Mineradores

Considere um exemplo de um robô com apenas duas metas

- Quando ele vê ouro, ele vai buscar o ouro
- Quando a bateria está baixa, ele recarrega a bateria

Robôs Mineradores - Java

```
public class Robot extends Thread {  
    boolean seeGold, lowBattery;  
    public void run() {  
        while (true) {  
            if (seeGold)  
                a = selectDirection(gold);  
            else  
                a = randomDirection();  
  
            doAction(go(a));  
  
        }  
    }  
}
```

Robôs Mineradores - Java

```
public class Robot extends Thread {  
    boolean seeGold, lowBattery;  
    public void run() {  
        while (true) {  
            if (seeGold)  
                a = selectDirection(gold);  
            else  
                a = randomDirection();  
  
            if (lowBattery) charge();  
            doAction(go(a));  
            if (lowBattery) charge();  
        }  
    }  
}
```

Robôs Mineradores - Jason

```
direction(gold) :- see(gold) .  
direction(random) :- not see(gold) .  
  
+!find(gold) <-  
    ?direction(A) ;  
    go(A) ;  
    !find(gold) .  
  
+battery(low) <-  
    !charge.  
  
^!charge[state(started)] <-  
    .suspend(find(gold)) .  
^!charge[state(finished)] <-  
    .resume(find(gold)) .
```

Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

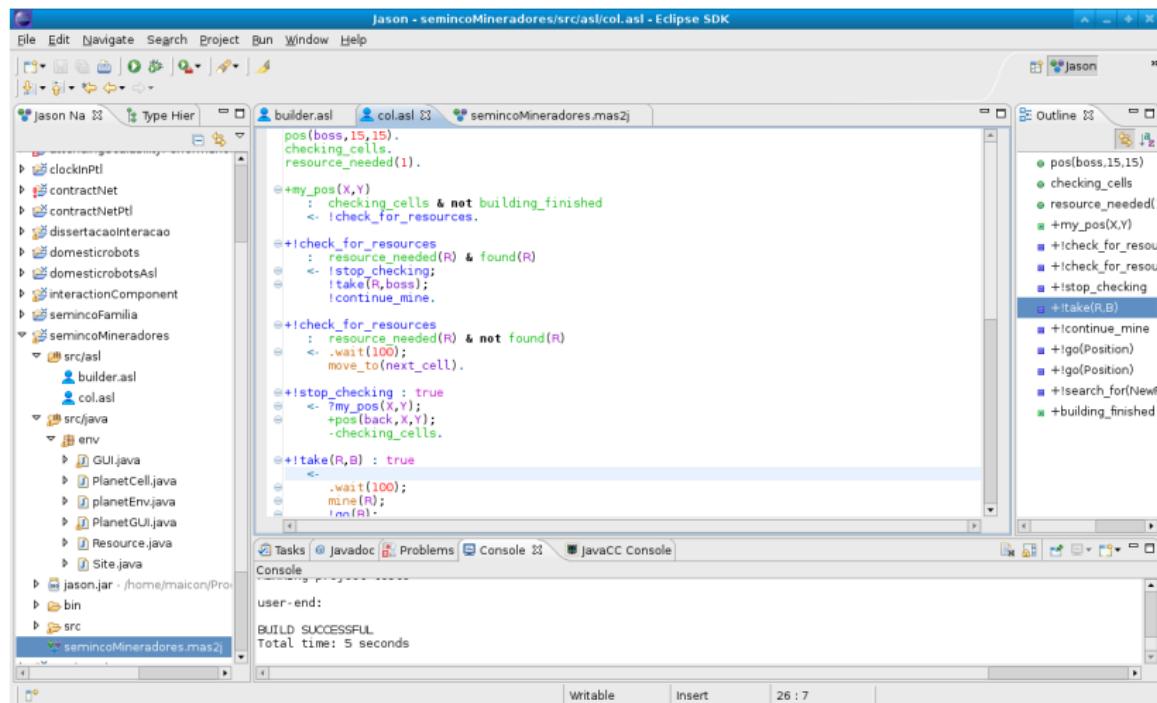
4 Competições

5 **Linguagem Jason**

- Jason x Java
- **Ferramentas**
- Demonstração

6 Conclusões

Jason - Plugin Eclipse



Jason - Mind Inspector/Debug

:: Jason Mind Inspector ::

Agent Inspection

Inspection of agent col1 (cycle #281)

- Beliefs

```
checking_cells[source(self)]:  
my_pos(22,16)[source(percept)]:  
pos(boss,15,15)[source(self)]:  
resource_needed(f)[source(self)]:
```

- Intentions

Sel	Id	Pen	Intended Means Stack (show details)
97	97/time100		+!check_for_resources[source(self)] +my_pos(22,16)[source(percept)]

Agent History

Cycle 0

Run cycle(s) for all agents view as: **html**

Jason - Mind Inspector/Debug Web

127.0.0.1:3272

Agents	Inspection of agent maria
- alice - francois - giacomo - maria	Beliefs commitment(francois,print_vowel,"hello_eng") commitment(giacomo,print_consonant,"hello_eng") commitment(maria,print_l,"hello_eng") commitment(alice,print_special_chars,"hello_eng") current_wsp(cobj_1,"hello_org","79deead6-6f5c-4884-81dc-768b14700e9a") formationStatus(ok) goalState("hello_eng",print_hello,[],[],satisfied) goalState("hello_eng",print_excl,[alice],[alice],satisfied) goalState("hello_eng",print_d,[giacomo],[giacomo],satisfied) goalState("hello_eng",print_l3,[maria],[maria],satisfied) goalState("hello_eng",print_r,[giacomo],[giacomo],satisfied) goalState("hello_eng",print_o2,[francois],[francois],satisfied) goalState("hello_eng",print_w,[giacomo],[giacomo],satisfied) goalState("hello_eng",print_spc,[alice],[alice],satisfied) goalState("hello_eng",print_o1,[francois],[francois],satisfied) goalState("hello_eng",print_l2,[maria],[maria],satisfied) goalState("hello_eng",print_l1,[maria],[maria],satisfied) goalState("hello_eng",print_e,[francois],[francois],satisfied) goalState("hello_eng",print_h,[giacomo],[giacomo],satisfied)

Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 **Linguagem Jason**

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Sumário

1 Introdução

2 Engenharia de Software

3 Linguagens de Programação

4 Competições

5 Linguagem Jason

- Jason x Java
- Ferramentas
- Demonstração

6 Conclusões

Algumas Conclusões

Agentes como um paradigma de programação recente

Há problemas que são resolvidos de maneira mais adequada com programação orientada a agentes

- Outros continuam tendo solução mais adequada utilizando outros paradigmas (ex: orientação a objetos)

Sistemas autônomos

Diversas linguagens e metodologias

É uma área em contínuo desenvolvimento

Maioria das linguagens/ferramentas são *free* e *open-source*

The End