Modelo e da Arquitetura BDI

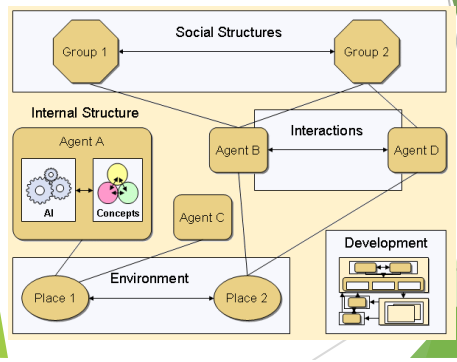
Motivação

- Sistemas Multi-Agentes (SMA)
  - Importante nova direção da Engenharia de Software
  - Alguns SMAs requerem agentes com raciocínio
    - Uso de técnicas de Inteligência Artificial
- Modelo BDI
  - Considerado a melhor forma de modelagem de agentes com raciocínio
- Implementação de Agentes BDI
  - Existência de várias linguagens e plataformas

Introdução

- Sistemas Multi-Agentes (SMA)
  - Outras Características dos Agentes
    - Adaptação: agente altera comportamento de acordo com um novo contexto
    - Aprendizado: agente altera comportamento com base na experiência
    - Racionalidade: agentes são capazes de selecionar ações de acordo com objetivos
    - Mobilidade: agentes são capazes de se mover de um ambiente para outro

Introdução



Introdução

- Agentes com Raciocínio
  - Agentes Cognitivos
  - Melhor forma de modelagem conhecida
    - Modelo belief-desire-intention (BDI)
- Modelo BDI (Bratman)
  - Explica comportamento do raciocínio humano
- Arquitetura BDI (Rao and Georgeff)
  - Teoria formal
  - Interpretador Abstrato

Modelo e Arquitetura BDI

- Muitas abordagens propõem diferentes atitudes mentais e seus relacionamentos
- Modelo BDI
  - Considerado o melhor modelo conhecido
  - Proposto por Bratman
    - Intention, Plans, and Practical Reason (1987)
  - Teoria filosófica que explica o comportamento humano com três atitudes mentais
    - Beliefs
    - Desires
    - Intentions

Modelo e Arquitetura BDI

- Modelo BDI
  - Beliefs (Crenças)
    - Características do ambiente
    - Atualizadas após a percepção de cada ação
    - Componente informativo do sistema
  - Desires (Desejos)
    - Informação sobre os objetivos a serem atingidos
    - Representação do estado motivacional do sistema
  - Intentions (Intenções)
    - Atual plano de ação escolhido
    - Componente deliberativo do sistema

Modelo e Arquitetura BDI

- Arquitetura BDI
  - Proposta por Rao and Georgeff
    - BDI-agents: from theory to practice (1995)
  - Adotou o modelo BDI para agentes de software
  - Introduziu
    - Teoria Formal
    - Interpretador BDI abstrato
  - Base para sistemas BDI históricos e atuais
- Procedural Reasoning Systems (PRS)
  - Primeiro sistema implementado com sucesso

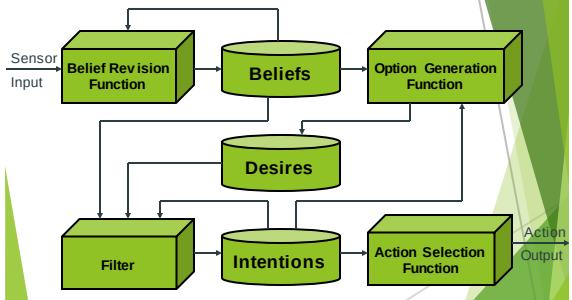
Modelo e Arquitetura BDI

- Beliefs
  - Informações do agente sobre seu ambiente
- Belief Revision Function
  - Determina novo conjunto de crenças a partir da percepção da entrada e das crenças atuais
- Option Generation Function
  - Determina as opções disponíveis ao agentes (desejos), com base nas crenças sobre o ambiente e nas suas intenções
- Desires
  - Possíveis planos de ações disponíveis ao agente

Modelo e Arquitetura BDI

- Filter
  - Processo de deliberação do agente
  - Determina intenções do agente com base nas suas crenças, desejos e intenções atuais
- Intentions
  - Foco atual do agente
  - Estados que o agente está determinado a alcançar
- Action Selection Function
  - Determina ação a ser executada com base nas intenções atuais

Modelo e Arquitetura BDI



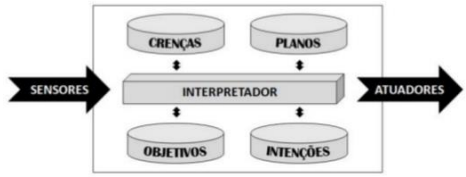
Implementação de Agentes BDI

- Variedade de Linguagens e Plataformas para Implementação de Agentes BDI
  - PRS (Bratman, 1987)
  - dMars (D'Inverno et al., 1998)
- Plataformas estudadas
  - JACK™ Intelligent Agents (Winikoff, 2005)
  - Jadex (Bellifemine et al., 2007)
  - JAM (Huber, 1999)
  - Jason (Bordini et al, 2007)

Procedural Reasoning System

- Utilizado em modelos computacionais de agentes quando precisam ser implementados **raciocínio prático**, usando a **deliberação e raciocínio fim-meio**
- Planos contem os seguintes componentes:
  - Metas (goals)
  - Contexto (contexto)
  - Corpo (Body)
- Ao iniciar um agente PRS terá uma coleção de **planos** e **crenças iniciais** sobre o ambiente;
- As crenças são representadas como formulas atômicas da linguagem de primeira ordem, e o agente possuirá uma **meta principal**

ARQUITETURA PRS



JACK™ Intelligent Agents

- Framework para o desenvolvimento de sistemas multi-agentes
- Desenvolvido pela *Agent Oriented Software Pty. Ltd.* (AOS)
  - Melbourne, Austrália
- Linguagem
  - JACK Agent Language

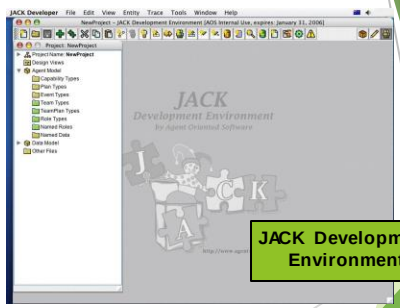
JACK

- Características
  - Leve, requer poucos recursos de sistema
  - Comunicação transparente entre agentes
  - Ferramentas de Desenvolvimento
  - Aplicações Comerciais
    - UVAs (Unmanned Aerial Vehicles)
    - Gerenciamento de tráfego aéreo
    - Real-time scheduling

JACK

- JACK Agent Language
  - Linguagem de programação orientada a agentes
  - Extensões à linguagem Java
    - Novas classes, interfaces e métodos
    - Extensões à sintaxe de Java
      - Compilador: JACK AL para Java
    - Extensões semânticas
      - Suporte ao modelo de execução requerido por um sistema orientado a agente

JACK



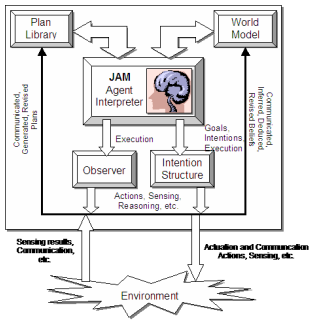
JAM

- Arquitetura de agentes inteligentes
- Desenvolvido pela Intelligent Reasoning Systems (I.R.S.)
  - Oceanside, California
- Linguagem
  - JAM

JAM

- Características
  - Alguns bugs / limitações reportados
  - Nenhuma aplicação comercial conhecida
  - Nenhuma ferramenta de desenvolvimento

JAM



Jadex

Jadex
BDI Agent System

Jadex

- Mecanismo de raciocínio BDI para agentes inteligentes
- Projeto conduzido pelo Distributed Systems and Information Systems Group
  - University of Hamburg, Alemanha
- Linguagem
  - Java e XML

Jadex

- Características
  - Não introduz nova linguagem
  - FIPA Compliant
    - Uso do JADE como plataforma SMA
  - Integração com ferramenta de projeto de Ontologias
    - Protégé
  - Ferramentas de Desenvolvimento
  - Aplicações Comerciais
    - MedPAge
    - Dynatech
    - Bookstore

Jadex

- Principais Componentes
  - Belief
    - Conhecimento do agente sobre ambiente e si mesmo
    - Podem ser qualquer objeto Java
    - Armazenadas em uma base de crenças
    - Permite consulta através de OQL-like query language
  - Goal
    - Orientam ações do agente
    - Desejos concretos e momentâneos do agente
    - Agente executa ações apropriadas até que o objetivo seja considerado
      - Atingido
      - Inatingível
      - Não mais desejado

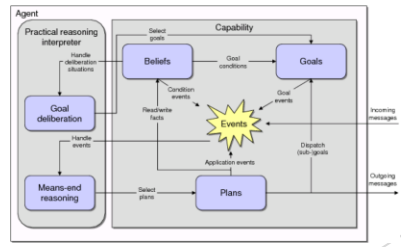
Jadex

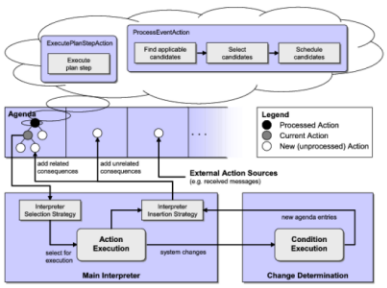
- Principais Componentes
  - Plan
    - Forma como o agente atuará em seu ambiente
    - Dependendo da situação corrente
      - Planos selecionados como resposta à ocorrência de eventos ou de objetivos
    - Seleção de planos feita automaticamente pelo sistema
  - Capability
    - Crenças, planos e objetivos podem ser colocados em um módulo de agente
    - Podem conter subcapacidades formando uma hierarquia
    - Possibilidade de reuso

Jadex

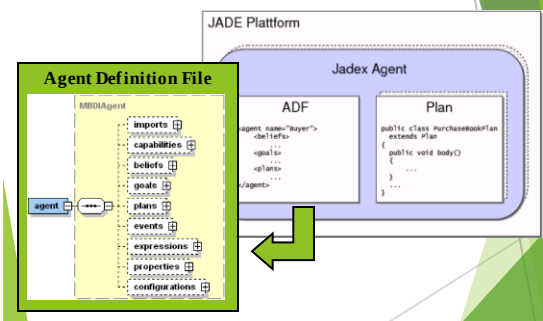
- Principais Componentes
  - Event
    - Importante propriedade dos agentes
      - Capacidade de reagir a diferentes tipos de eventos
    - Jadex suporta dois tipos de eventos a nível de aplicação
      - Eventos internos
        - Usados para denotar uma ocorrência dentro de um agente
      - Eventos mensagem
        - Comunicação entre dois agente ou mais
    - Normalmente tratados por planos

Jadex



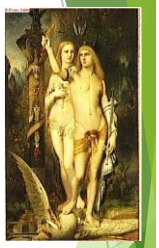


JADEX



Jason

- Interpretador para uma versão estendida da linguagem AgentSpeak(L)
- Desenvolvido por
  - Jomi F. Hübner (Blumenau, BR)
  - Rafael H. Bordini (Durham, UK)
- Linguagem
  - Agent Speak (L)



Jason

- Características
  - Agent Speak (L) possui semântica formal
    - Possibilita verificação formal
  - Ferramentas de Desenvolvimento
  - Nenhuma aplicação comercial conhecida

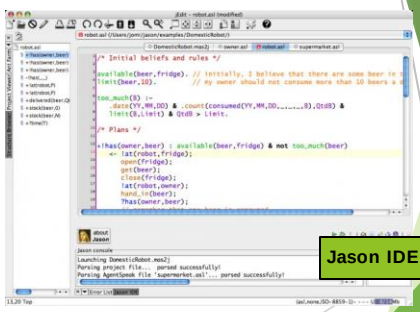
Jason

- Agent Speak (L)
  - Linguagem de programação orientada a agentes
  - Baseada na lógica de primeira ordem
  - Inspirada na
    - Arquitetura BDI
    - Lógica BDI

JASON

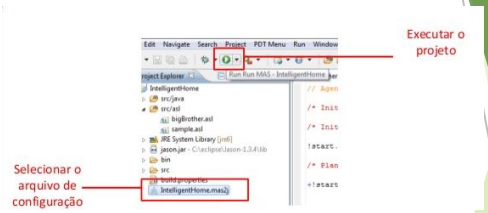
- Agente possui crenças, metas, planos e ações
- Agente são inseridos em um ambiente que estende a classe *Environment*
- Percepções e reações a estímulos do ambiente são programadas em Java

Jason

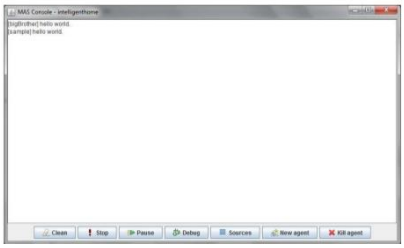


Jason

Executando Projeto



Console Jason



Crenças (Beliefs)

- Agente armazena informações percebidas do ambiente, informações internas e informações de comunicação através das crenças
- Armazenadas na Base de Crenças
- Representadas como predicados da lógica tradicional
  - Representam propriedades particulares

Tipos

- Percepções do ambiente (percepts)
  - Informações coletadas pelo agente que são relativas ao sensoriamente constante do ambiente
- Notas mentais
  - Informações adicionadas na base de crenças pelo próprio agente resultado de coisas que aconteceram no passado, promessas
    - Adicionada após execução de um plano constante no ambiente
- Comunicação
  - Informações obtidas pelo agente através da comunicação com outros agentes

Exemplos

salario(5000).

alto(giba).

missionStarted.

carro(c3, kadu).

- Toda crença inicial devem terminar com .
- Toda crença devem começar com letra minúscula
- Negação: deve começar com ~
  - ~missionStarted
  - ~alto(giba)
  - ~dia

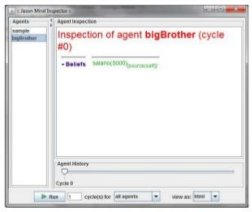
Sintaxe

- Crenças:
  - editora(wiley)
- Objetivos
  - Alcançar
    - !escrever(livro)
    - !dono(casa)
  - Teste (estão na base de crença)
    - ?publisher(P)
    - ?saldo(banco)

Crenças Iniciais

- Crenças do agente bigbrother

salario(5000).



Exercicio

```
MAS oieagente {
    agents:agente;
}

agente.asl
inicio.
inicio2.
+inicio <- .print("Tudo Bem?").
+inicio2 <- .print("Olá Mundo!"," Meu Nome é ...").
```

```
MAS room {
    infrastructure:    Centralised
    agents:
        claustrofobico;
        paranoico;
}
```

```
/* claustrofobico*/
fechada(porta).
+fechada(porta) : true
<-.print("Porta Fechada. Vou abrir!!!");
-fechada(porta);
+aberta(porta);
.print("Porta Aberta.").
```

```
/*paranoico*/
aberta(porta).
/* Plans */
+aberta(porta) : true
<-.print("Porta Aberta. Vou Fechar!!!");
-aberta(porta);
+fechada(porta);
.print("Porta Fechada...").
```

```
MAS agenteparimpar {
    agents: agenteparimpar;
}
/*Parimpar.asl*/
!print_par(15).
+!print_par(N)
   <- !par(N,F);
       .print(N, " é ",F).

+!par(N,F) : ((N mod 2) == 0) <- F=par.
+!par(N,F) : ((N mod 2) > 0) <- F=impar.
```

```
MAS agentefatorial {
    agents: agentefatorial;
}

!imprime_fatorial (5).

+!imprime_fatorial(N)
<- !fatorial(N,F);
.print("Fatorial de ", N, " é ", F).

+!fatorial (N,1) : N == 0.

+!fatorial (N,F) : N > 0
   <- !fatorial(N-1,F1);
      F = F1 * N.
```

```
MAS factorial {
    agents: fibonacci;
}
/*Agente Fibonacci.asl*/
!print_fibonacci(10).
+!print_fibonacci(N)
  <- !fibonacci(N,F);
     .print(N, " número da série de Fibonacci é ",F).
+!fibonacci(N,1) : N == 1.
+!fibonacci(N,1) : N == 2.
+!fibonacci(N,F) : N > 2
  <- !fibonacci(N-1,F1);
     !fibonacci(N-2,F2);
     F = F1 + F2.
```

Objetivos (Goals)

- Objetivos representam os estados do mundo em que o agente deseja atingir
- Tipos:
  - Achievement Goals (!)
    - É um objetivo para atingir determinado estado desejado pelo agente
  - Test Goals (?)
    - É um objetivo que tem basicamente a finalidade de resgatar informações da base de crenças do agente

Exemplos

- Goals iniciais

  !start.

  !thinking.

  !print("Juliana").
- Toda goal inicial deve ser um *Achievement Goal*, começa com **!** E termina com **.**
- Todo goal deve começar com letra *minúscula*

Exemplo Goals Iniciais

- Objetivos do Agente BigBrother

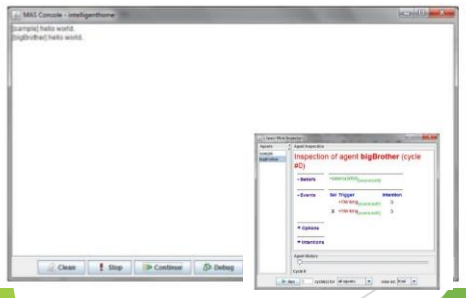
!start.

!thinking.

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.
```

Console Exibindo execução e Debug



Planos e Ações

- Composto por três partes:

Triggering_event : context <- body.

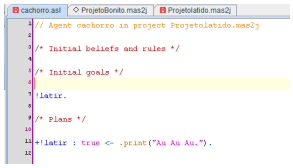
```
+!order(Product,Qtd)[source(Ag)] : true <-
    ?last_order_id(N);
    OrderId = N + 1;
    -+last_order_id(OrderId);
    deliver(Product,Qtd);
    .send(Ag, tell, delivered(Product,Qtd,OrderId)).
```

Plano

- Triggering Event
  - Agente pode ter diversos objetivos
  - Planos são ativados baseados nos eventos que podem ser ativados em determinado momento
- Context
  - São as condições para a ativação de um plano dentro de vários eventos
- Body
  - É o corpo do plano
  - Uma sequencia de ações a ser executada pelo agente

Tipos de Triggering Events

- Addition
  - Ativados quando um plano é transformado de um desejo para uma intenção na mente do agente



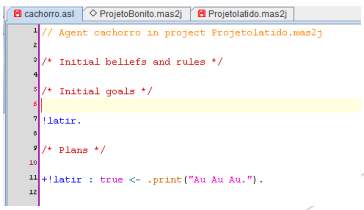
Tipos de Triggering Events

- Deletion
  - Funciona como um tratamento de erros para planos que não possuem ativação

```
/* Initial beliefs and rules */

/* Initial goals */

!latir.
/* DELEÇÃO*/

+!latir : cachorro(desconhecido) <- .print("Au Au Au.").

-!latir: true <-
    .print("snif snif snif");
    !latir.
```

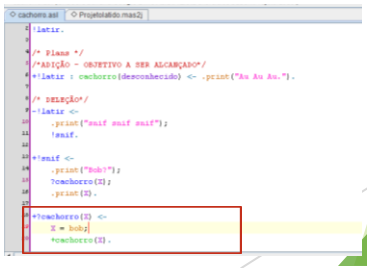
Tipos de Planos

- Achievement goal
  - Objetivos que os agentes se comprometem em atingir



Tipos de planos

- Test Goal
  - Objetivos que recuperam informações da base de crenças



Tipos de Planos

- Beliefs
  - Planos ativados quando o agente adiciona ou remove uma crença da sua base de crenças

```
!snif.

+!snif <-
    .print("Bob?");
    +dog(bob).

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

Ações de um Plano

- Achievemente e test Goals
  - São as chamadas para execução de um plano

Ações de Um plano

- Mental Notes
  - Ações que adicionam, removem ou atualizam uma crença na base de crença do agente

```
!snif.
+!snif <-
    .print("Bob?");
    +cachorro(bob).

+cachorro(bob) <- .print("sniff sniff").
```

```
fome.
comida(100).
estomago(0).

!comer.
+!comer: fome & comida(F) & estomago(S) & S<=50 <-
    .print("Comendo...");
    -+comida(F-1);
    -+estomago(S+1);
    .print(F);
    !comer.
+!comer: estomago(S) & S>50 <-
    .print("Eu estou satisfeito");
    -fome.
```

Ações de um Plano

- InternalAction
  - Ações pré-definidas executadas dentro do raciocínio do agente

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

Ações de um plano

- ExternalAction
  - Ações executadas no ambiente em que o agente estiver inserido

```
!walk.

+!walk <-
    .print("Lets sniff the environment!");
    sniff.
```

Ações de um Plano

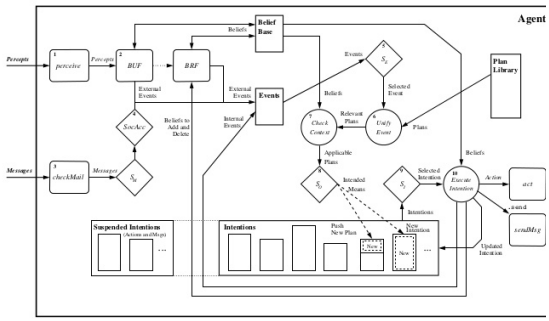
- Expressões

```
!latir.
+!latir<-
    .print("Sniff!");
    !snif;
    .print("sniff!").
+!snif <-
    .print("Bob?");
    ?cachorro(X);
    .print(X).
+?cachorro(X) <-
    X = bob;
    +cachorro(X);
    .print ("Eu encontrei ",X).
```

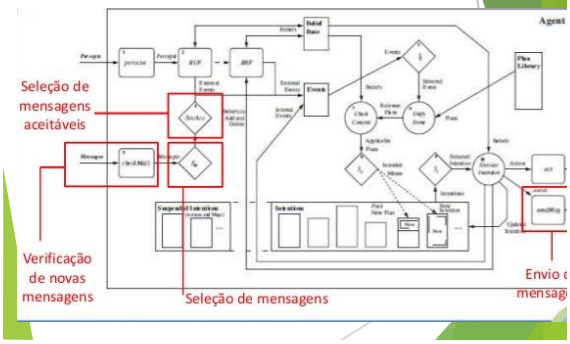
Comunicação entre agentes

- A cada inicio de racicionio, o agente verifica mensagens que ele possa ter recebido de outros agentes
- Baseado em Speech Act e KQML

Jason Rreasoning Cycle



Reasoning Cycle



Estrutura

<sender;illocutionary forces;content>

- Sender
  - Proposição atômica representando o nome do agente que enviou a mensagem
- Illocutionary forces
  - Performativas que denotam as intenções do remetente
- Content
  - Conteúdo da mensagem enviada

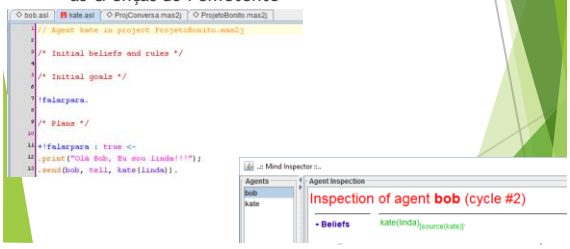
Estrutura - Jason

.send(receiver, illocutionary forces, propositional content)
.broadcast(illocutionary forces, propositional content)

- Receiver
  - Proposição atômica em agentSpeak representando nome do agente que enviou a mensagem
- Illocutionary Forces
  - Performativas que denotam as intenções do remetente
- PropositionalContent
  - Termo em AgentSpeak que varia de acordo com as forças ilocucionarias

Performativas Implementadas

- Tell
  - Agente remetente pretende que o receptor **acredite** que os conteúdo enviado é verdadeiro de acordo com as crenças do remetente

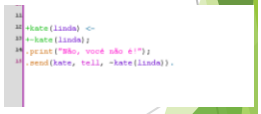


Performativas Implementadas

- Untell
  - Agente remetente pretende que o receptor **não acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente

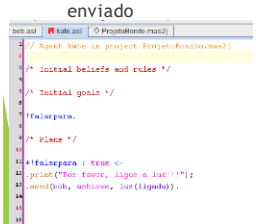
Kate

Bob



Performativas Implementadas

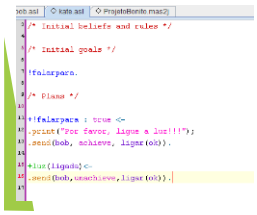
- achieve
  - Agente remetente pede que o receptor **tente atingir um objetivo** de estado verdadeiro de acordo com conteúdo enviado



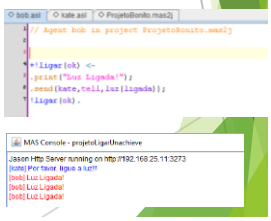
Performativas Implementadas

- unachieve
  - Agente remetente pede que o receptor **deixe de tentar atingir um objetivo** de estado verdadeiro de acordo com conteúdo enviado

Kate



Bob

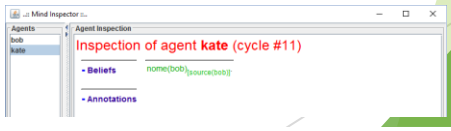


Performativas Implementadas

- askOne
  - Agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira

Agente Kate

Agente Bob

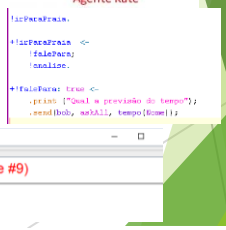


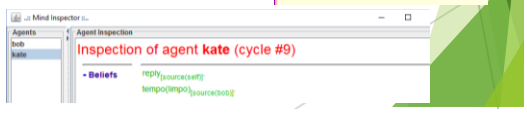
Performativas Implementadas

- askAll
  - Agente remetente deseja saber todas as respostas do receptor sobre uma questão

Agente Bob

Agente Kate





Performativas Implementadas

- askHow
  - Agente remetente deseja saber todas implementações de planos do receptor para determinado plano

Performativas Implementadas

- tellHow
  - Agente remetente informa ao agente receptor a implementação de um plano

Agente Bob

Performativas Implementadas

- untellHow
  - Agente remetente solicita ao agente receptor a remoção da implementação de um plano da biblioteca de planos do receptor

Agente Bob

Performativas Implementadas

- broadcast
  - Permite o uso de todas as performativas vistas anteriormente
  - Contudo não é preciso identificar o agente de destino, visto que ela será enviada a todos os agentes do SMA

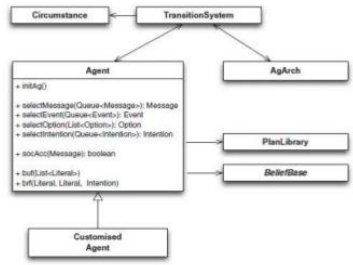
Exercicio

- Crie um SMA que simule um quarto inteligente. O quarto será controlado por um agente que começará com as seguintes crenças sobre o quarto:
  - É dia
  - A luz do quarto está apagada
  - Kate está no quarto
  - Bob não está no quarto
  - A temperatura atual do quarto
- Adicione aos SMA um novo agente que simule o despertador. Esse novo agente deverá ter 3 crenças, incluindo o horário para despertar da Kate e do Bob
- Analise no Mind inspector as crenças dos agentes SMA.

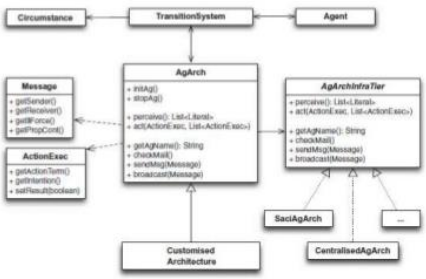
Exercicio

- Adicione ao agente quarto inteligente os seguintes objetivos iniciais (gere um plano de ação para cada objetivo abaixo, .print("Objetivo")):
  - Gerenciar a luminosidade
  - Gerenciar as pessoas que estão no quarto
- Adicione ao agente despertardor um objetivo para despertar. Quando o objetivo for executado, este deve imprimir alguma mensagem. O despertador deve despertar sem nenhuma condição específica.

ARQUITETURA DE UM AGENTE - JASON



ARQUITETURA DE MENSAGEM - JASON



SMA - Com ambiente

```
hungry.
stomach(0).

!eat.

+!eat: hungry & food(F) & stomach(S) & S<=50 <-
    .print("Eating...");
    eat;
    -+stomach(S+1);
    .print(F);
    .wait(500);
    !eat.

+!eat: stomach(S) & S>50 <-
    .print("I'm Satisfied.");
    -hungry.
```

Crença agente - Debug

A percepção global com origem do ambiente (percept).

- Agentes Competindo por Um Recurso

Comparação

	Linguagem
JACK	JACK (extensão de Java)
Jadex	Java e XML
JAM	JAM (extensão de Java)
Jason	Agent Speak (L)

ingridnunes@gmail.com

Comparação

	Ferramentas de Desenvolvimento
JACK	IDE e Debug
Jadex	Ferramentas para execução, debug e documentação
JAM	-
Jason	IDE e Mind Inspector

Comparação

	Aplicações Comerciais
JACK	UVAs, Tráfego Aéreo e Real-time Scheduling
Jadex	MedPAge, Dynatech e Bookstore
JAM	-
Jason	-

Comparação

	Outras Características
JACK	Não é livre, Leve, Preocupação com Indústria
Jadex	FIPA-Compliant, Facilidade de Integração com Ontologias
JAM	-
Jason	Semântica Formal

Comparação

- Todos os SMAs são escritos em Java ou uma extensão da linguagem
  - Herança das vantagens
    - Portabilidade
    - Variedade de
      - Ambientes de Desenvolvimento
      - Outras Ferramentas
    - Bibliotecas já existentes