

Modelo e da Arquitetura BDI

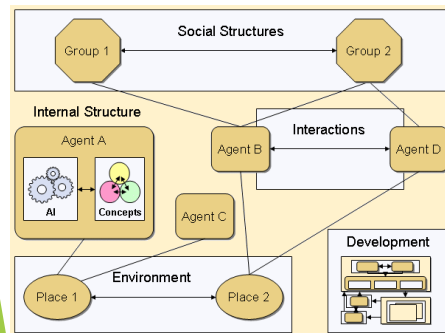
Motivação

- 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

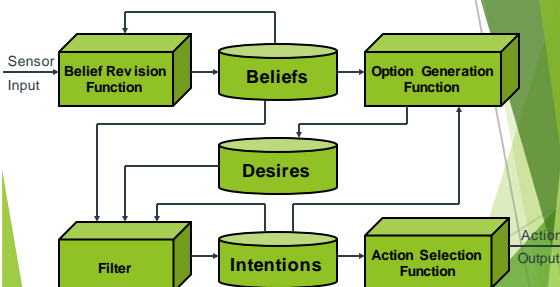
Modelo e Arquitetura BDI

- ▶ 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 agente (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 contêm 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



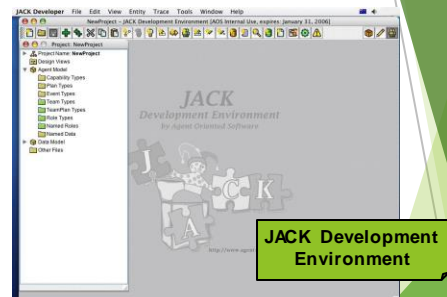
JACK

- Características
 - Leve, requer poucos recursos de sistema
 - Comunicação transparente entre agentes
 - Ferramentas de Desenvolvimento
 - Aplicações Comerciais
 - UAVs (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



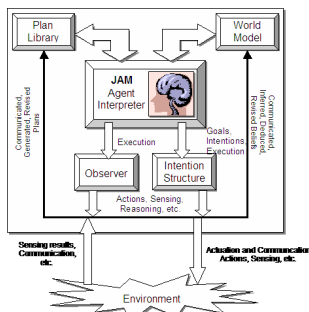
JAM

- ▶ 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

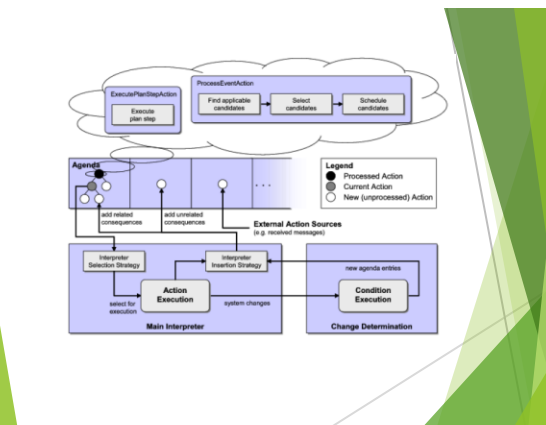
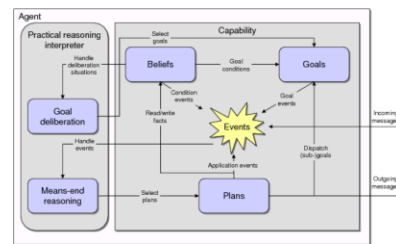
- Plan
 - Forma como o agente atuará em seu ambiente
 - Dependendo da situação corrente
 - Planos selecionados como resposta à ocorrência de eventos ou de objetivos
 - Seleção de planos feita automaticamente pelo sistema
- Capability
 - Crenças, planos e objetivos podem ser colocados em um módulo de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

Jadex

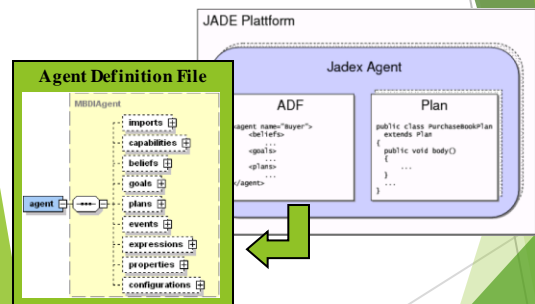
► Principais Componentes

- 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

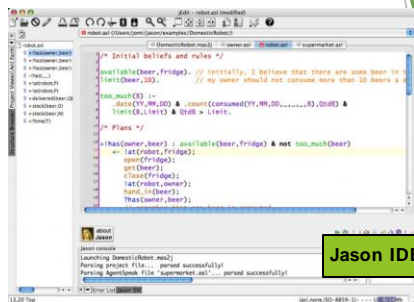
Jason

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

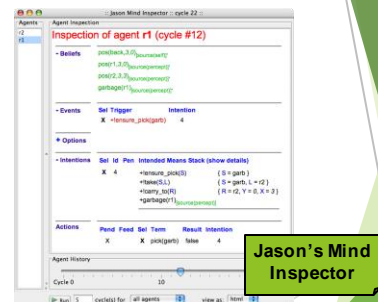
JASON

- 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

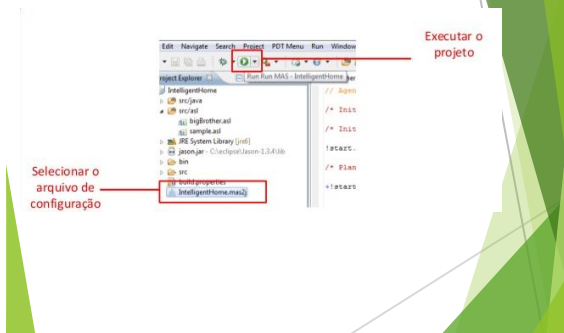
Jason



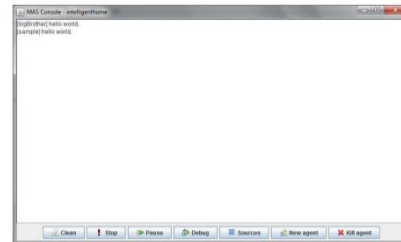
Jason



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 e 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

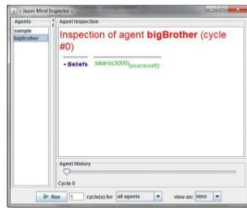
Sintaxe

- ▶ 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 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.

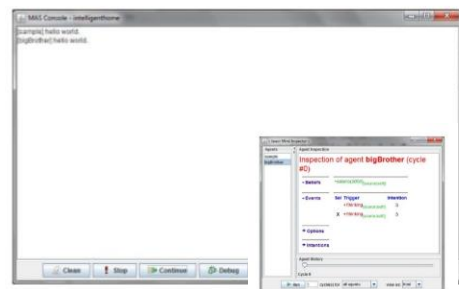
```
// bigbrother.ad
// Agent BigBrother in project IntelligentHome

/* 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 sequência de ações a ser executada pelo agente

Ações de Um plano

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

```

1 sniff.
2 +!sniff <-
3   .print("Bob?");
4   +cachorro(Bob);
5
6 +cachorro(bob) <- .print("sniff sniff").
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1
```

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.
```



The screenshot shows a console window titled "MAS Console - agActionExternalAction". It contains two lines of text: "[kate] Lets sniff the environment!" in blue and "[envGarden] Spot sniffed by kate" in red.

Ações de um Plano

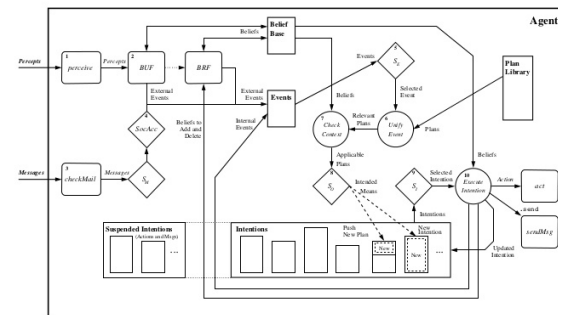
- Expressões

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

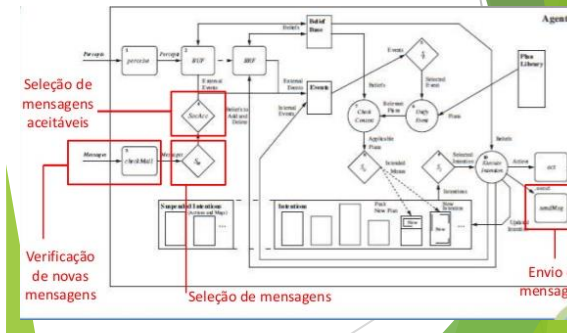
Comunicação entre agentes

- ▶ A cada início de racionio, 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

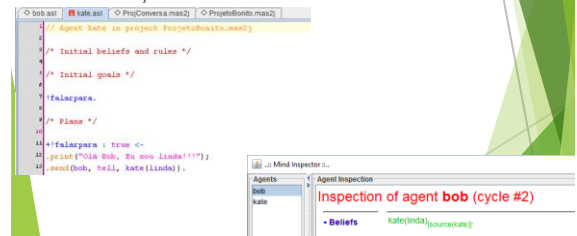
Estrutura - Jason

.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
- Propositional Content
 - Termo em AgentSpeak que varia de acordo com as forças illocucionárias

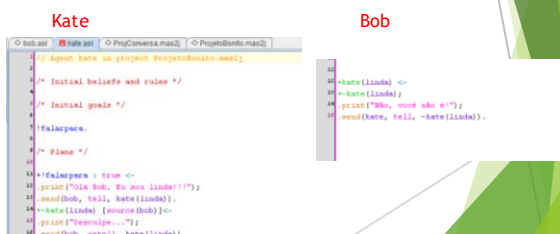
Performativas Implementadas

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



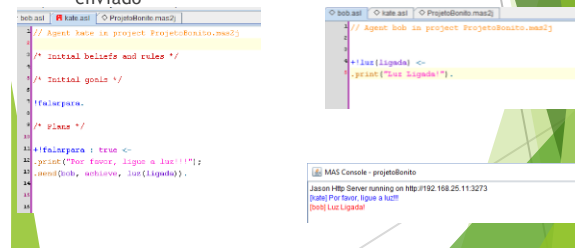
Performativas Implementadas

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



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

```

// Kate.asl
/* Initial beliefs and rules */
/* Initial goals */
/* Failure rules */
/* Plans */

+failureplan : true <-
  .print("Por favor, Ligue a luz!!");
  .send(Bob, unachieve, ligar(100));
+ligar(ligado) <-
  .send(Bob, unachieve, ligar(100));

```

Bob

```

// Bob.asl
/* Agent bob in project ProjetoBomito.mas2 */
+ligar(ok) <-
  .print("Luz Ligada!");
  .send(Kate, tell, luz(ligado));
  .ligar(ok);

```

MAS Console - projetoBomito.mas2

```

Jason Http Server running on http://92.168.25.11:3273
[Bob] Por favor, Ligue a luz!!
[Bob] Luz Ligada!
[Bob] Luz Ligada!
[Bob] Luz Ligada!

```

Performativas Implementadas

► askOne

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

Agente Kate

```

+failureplan : true <-
  .print("Quebra-me nome!");
  .send(Bob, askOne, nome(Bob), Reply);
  .Reply;

```

Agente Bob

```

nome(bob);

```

Mind Inspector - Agent Inspection

Inspection of agent kate (cycle #11)

- Beliefs: nome(bob)sourcebob
- Annotations

Performativas Implementadas

► askAll

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

Agente Bob

```

tempo(longo);
tempo(tempo(longo));

```

Agente Kate

```

+failureplan :
  +failureplan :
  failureplan :
  failureplan :
+failureplan : true <-
  .print("Qual a previsão do tempo?");
  .send(Bob, askAll, tempo(Bob));

```

Mind Inspector - Agent Inspection

Inspection of agent kate (cycle #9)

- Beliefs: tempo(longo)sourcebob

Performativas Implementadas

► askHow

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

```

+failureplan :
+failureplan : true <-
  .print("Ola Bob, Eu sou ligada!!");
  .send(Bob, tell, kate(ligado));
  .kate(ligado) {source(bob)} <-
    .print("Desligue...");
    .send(Bob, untell, kate(ligado));

```

```

+failureplan :
+failureplan : true <-
  .print("Ola Bob, Eu sou ligada!!");
  .send(Bob, tell, kate(ligado));
  .kate(ligado) {source(bob)} <-
    .print("Desligue...");
    .send(Bob, untell, kate(ligado));

```

MAS Console - projetoBomito.mas2

```

Jason Http Server running on http://172.16.0.125:3273
[Bob] Ola Bob, Eu sou ligada!!
[Bob] Por favor, voce pode me ensinar como ligar a luz?
[Bob] Eu estou aprendendo... aguarde...
[Bob] Luz Ligada.

```

Performativas Implementadas

► tellHow

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

Agente Bob

```

+teach(kate);
+teach(kate) <-
  .print("This is how we do it.");
  .send(Kate, tellHow, "lights(on)");
  .wait(2000);
  .send(Kate, achieve, turn(on));
+turn(on) <-
  .print("Lights On.");

```

```

+turn(on) <-
  .print("Lights On.");

```

MAS Console - agCommTellHow

```

[Bob] This is how we do it.
[Bob] Lights On.

```

Performativas Implementadas

► unteHow

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

Agente Bob

```

+teach(kate);
+teach(kate) <-
  .print("This is how we dance.");
  .wait(2000);
  .send(Kate, tellHow, "dance");
  .wait(1000);
  .dance;
+dance : true <-
  .print("I'm dancing with myself!");
  .wait(1000);
  .dance;

```

```

+teach(kate);
+teach(kate) <-
  .print("This is how we dance.");
  .wait(2000);
  .send(Kate, tellHow, "dance");
  .wait(1000);
  .dance;
+dance : true <-
  .print("I'm dancing with myself!");
  .wait(1000);
  .dance;

```

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



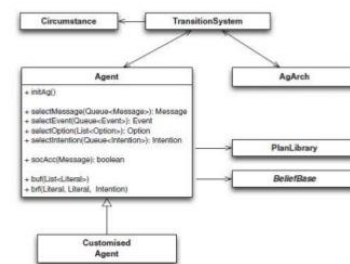
Exercicio

- 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.

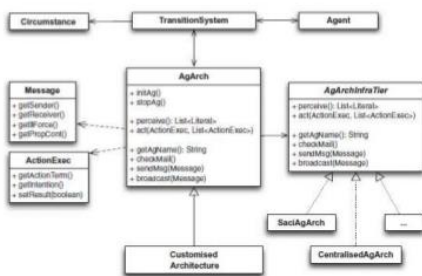
Exercicio

- 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 despertador 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).

leat.

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

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

```

Crença agente - Debug



Inspection of agent kate (cycle #53)

- Beliefs
 - food(90){source:percept}
 - hungry{source:self}
 - stomach(5){source:self}
- Intentions

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