

# Introdução aos Sistemas Multiagentes

*Rafael H. Bordini*

Instituto de Informática

Universidade Federal do Rio Grande do Sul

<http://www.inf.ufrgs.br/~bordini>

# Programa

- Conceitos Básicos e Áreas de Pesquisa
- Ferramentas e Aplicações
- Arquitetura BDI
- Comunicação entre Agentes
- Programação Orientada a Agentes BDI

# Conceitos Básicos e Áreas de Pesquisa

# Caracterização da Área

- Sistemas Multiagentes
  - Cognitivos
  - Reativos
- Enfoques
  - Teoria dos Jogos
  - Lógica Matemática

# Caracterização da Área

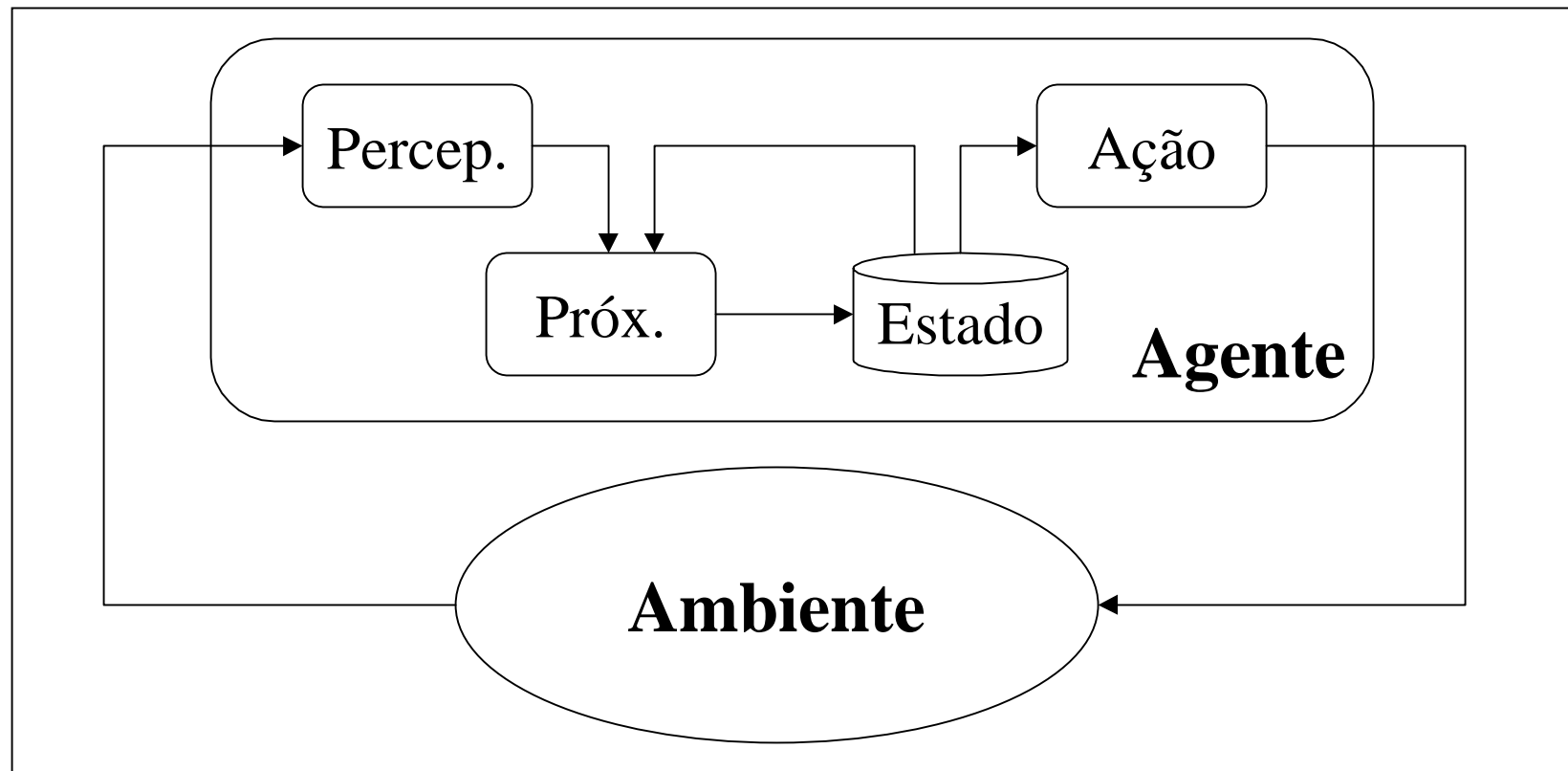
- Indefinição do conceito de agente

Entidade de software com capacidade de percepção, representação, raciocínio, motivação, deliberação, (aprendizado) e comunicação.

- Multidisciplinaridade

Psicologia, Sociologia, Economia, Teoria das Organizações, Entomologia, ...

# Modelo Geral de Agente



# Sociedades de Agentes

- Cada agente é limitado: agentes precisam interagir para alcançar seus objetivos
- Não há controle central
- Autonomia
- Coordenação (cooperação, competição, etc.)
  - Negociação, Leilão, etc.
  - Relações de Dependência
- Comunicação

# Motivação

- Antropomorfismo
- Ambientes Complexos e Dinâmicos:
  - Computação Distribuída
    - Decomposição de tarefas
    - Eficiência
    - Aplicações naturalmente distribuídas
  - Autonomia
    - Impossível prever todas as situações

# Áreas de Pesquisa

- Comunicação (Cohen) (Werner)
- Poder Social (Castelfranchi)
- Raciocínio Prático (Bratman)
- Arquitetura BDI (Georgeff)
- *Collective Misbeliefs* (Doran)
- Embasamento Sociológico (Gasser)
- Coordenação (Lesser, Durfee, Decker)

# Áreas de Pesquisa

- Programação Orientada a Agentes (Shoham)
- Engenharia de Software Orientada a Agentes (Wooldridge, Jennings)
- MAGMA (Demazeau)
- Sistemas Legados (Huhns)
- Aprendizagem por Reforço (Weiss)

# Áreas de Pesquisa

- Agentes Credíveis (Bates) (Hayes-Roth)
- Rede de Contratos (Smith,Davis)
- Teoria dos Jogos (Rosenschein) (Kraus)
- Planejamento (Grosz)
- Negociação (Sycara)
- Arquitetura SOAR (Tambe)
- Métodos Formais (Singh)

# Ferramentas e Aplicações

# Ferramentas

- <http://www.agentbuilder.com/AgentTools/>
  - DECAF Agent Framework
  - Java
    - JAM
    - ...
  - JACK
  - SIM\_AGENT
  - SWARM

# Aplicações

- Aplicações Industriais
  - Controle de Tráfego Aéreo (OASIS-dMARS)
  - Controle de Manufatura (Parunak)
  - Controle de Processos (Automação Industrial)
    - Distribuição de Energia Elétrica (ARCHON)
    - Controle de Espaçonaves
    - Monitoração de Usinas Nucleares
  - Telecomunicações
  - Sistemas de Transporte

# Aplicações

- Aplicações Comerciais
  - Gerência de Informações (PDA)
    - Filtro de Informações
    - Busca de Informações
  - Comércio Eletrônico
  - Gerência de Negócios
- Software Educativo
- Interface Humano-Computador

# Aplicações

- Aplicações Médicas
  - Monitoração de Paciente
  - Cuidados com a Saúde
- Entretenimento
  - Jogos
  - Cinema Interativo (agentes credíveis)
- Simulação Social

# Simulação Social

- Simulação de Sociedades (em particular de sociedades humanas):
  - Ciências Humanas: sociologia, economia, psicologia social, teoria das organizações, ciências políticas, antropologia, arqueologia...
- Simulação de Organizações (interesse comercial também)

# Simulação Social

- Objetivos:
  - Compreender determinada situação (em uma sociedade) e sua origem
  - Prever comportamentos futuros
- Uso de SMA para Simulação Social:
  - indivíduos/comunidade  $\Rightarrow$  agentes/SMA
  - “two-way micro-macro link” (dos comportamentos individuais às estruturas sociais)

# Simulação Social

- Reduccionismo...
- Exemplos:
  - Origem de estruturas sociais hierárquicas
  - Dominação em sociedades de primatas
  - Gerência de sistemas ecológicos (por exemplo, recursos hidráulicos)

# Simulação Social

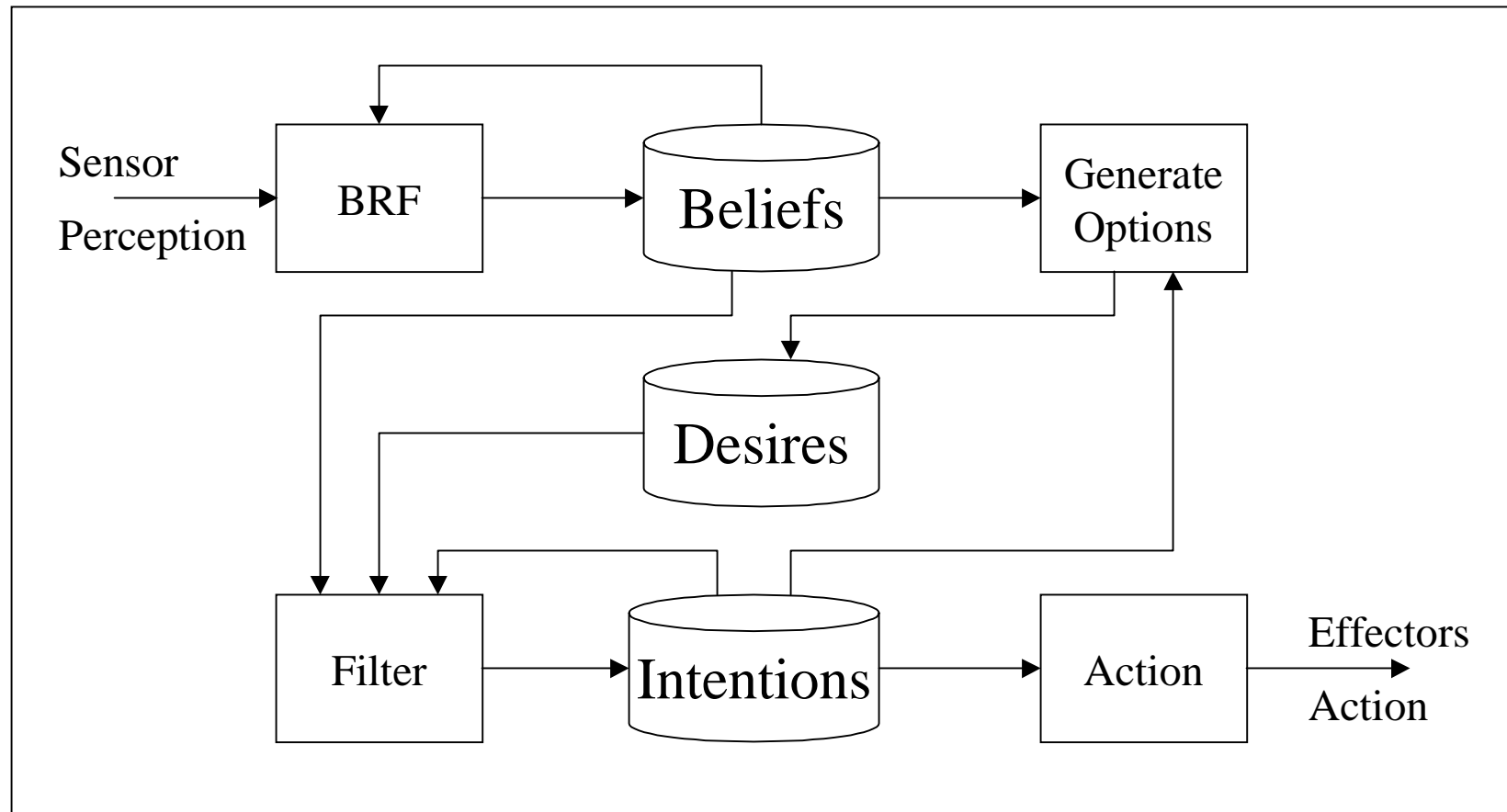
- Problemas do enfoque de SMA para SS:
  - Capacitação da equipe
  - Que tipo de modelo? (GT, CogSci)
  - Qual nível de abstração?
  - Controle de uma enorme quantidade de parâmetros
  - Validação!

# Arquitetura BDI

# Raciocínio Prático

- Intentional Stance (Dennett)
- Raciocínio Prático (Bratman, Isreal, Pollack)
- Arquitetura BDI (Georgeff, Rao)

# Arquitetura BDI



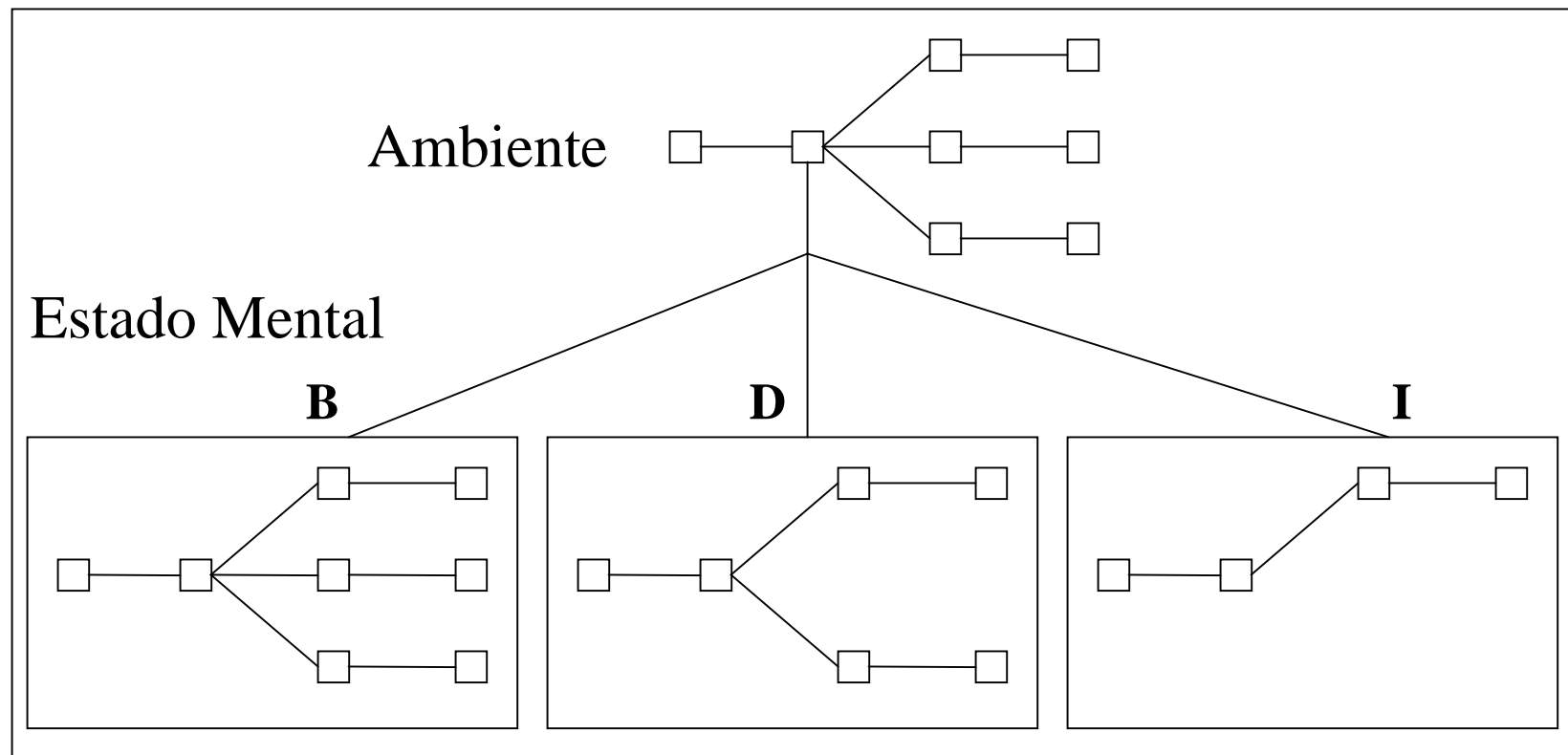
# Interpretador BDI

```
BDI-interpreter  
initialize_state();  
do  
    options := option_generator(event_queue, B, G, I);  
    selected_options := deliberate(options, B, G, I);  
    update_intentions(selected_options, I);  
    execute(I);  
    get_new_external_events();  
    drop_successful_attitudes(B, G, I);  
    drop_impossible_attitudes(B, G, I);  
until quit.
```

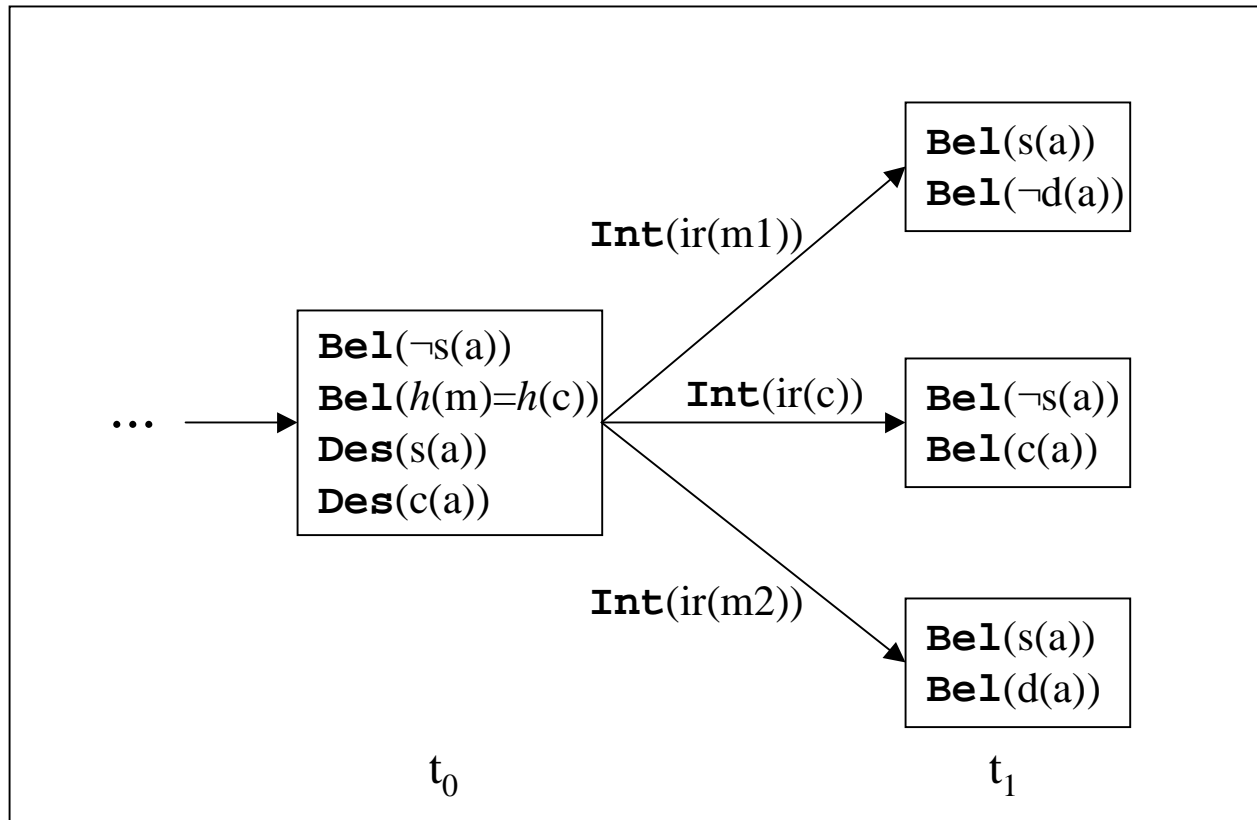
# Lógica BDI

- Lógica BDI é uma lógica multi-modal, temporal de tempo ramificado, e de ação (Georgeff and Rao)
- Operadores Modais
  - **Bel**( $x,p$ )
  - **Des**( $x,p$ )
  - **Int**( $x,p$ )onde  $x$  é um agente e  $p$  expressa estados de mundo

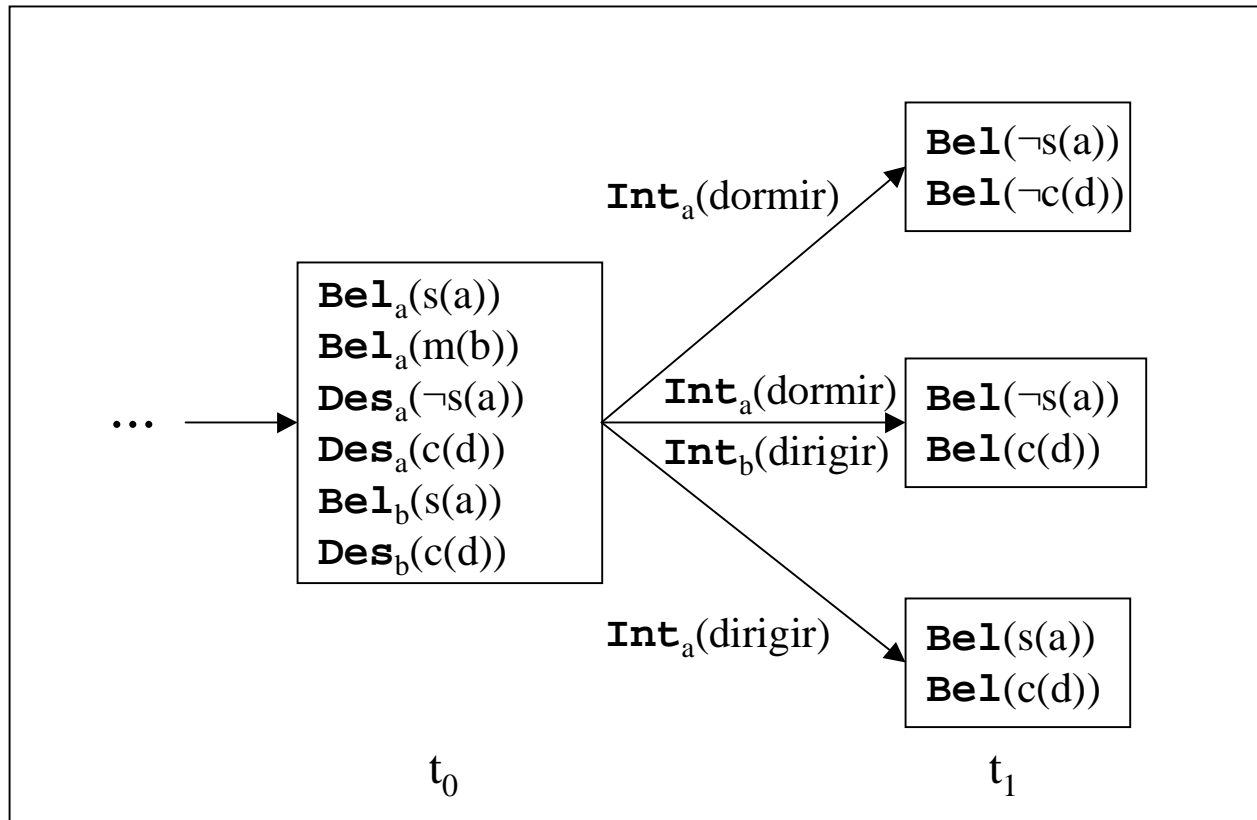
# Modelo de Mundos Possíveis e Tempo Ramificado



# Exemplo I



# Exemplo II



# Comunicação entre Agentes

# Atos de Fala

- Nem todas as sentenças expressam fatos
- Algumas sentenças realizam **ações**
  - Chamadas *performativas*
  - Não são *verdadeiras* mas *bem-sucedidas*
- Atos em sentenças performativas
  - locucionários
  - ilocucionários
  - perlocucionários

# Atos de Fala

- Exemplos de atos ilocucionários:
  - afirmar, requisitar, avisar, ordenar, desculpar-se
- Sentença performativa possui:
  - força ilocucionária
    - caracteriza a natureza do ato
  - conteúdo proposicional
    - especifica o que está sendo requisitado, avisado, etc.

# KSE - ACL (USDD - ARPA)

- KQML
  - Arquitetura, protocolos e linguagem para comunicação entre agentes
- KIF
  - Linguagem para representação de conhecimento
- ONTOLINGUA
  - Ontologias (vocabulário)

# KQML

- Formato de mensagens trocadas por agentes
- Baseada em Atos de Fala - Performativas
- Arquitetura: facilitadores
- Padrão? (FIPA e outros)
- <http://agents.umbc.edu/kqml/>
- <http://www.lti.pcs.usp.br/saci/> (Jomi Hübner)

# Formato Básico

performativa (        :sender        <word>  
                          :receiver      <word>  
                          :language     <word>  
                          :ontology    <word>  
                          :content     <expression>        )

# Information Performatives

- *tell* - S informa para R que seu conteúdo é verdadeiro, ou seja, que a sentença está em sua base de conhecimento;
- *deny* - S informa R que a performativa incluída na mensagem **não** é verdadeira para S;
- *untell* - um *deny* de um *tell*;

# Query Performatives

- *ask-if* - S quer saber se o conteúdo de sua mensagem é verdadeiro para R;
- *ask-all* - S quer todas as instâncias para as quais o conteúdo da mensagem é verdadeiro para R;

# Basic Responses

- *error* - S indica a R que recebeu uma mensagem não compreendida;
- *sorry* - S diz a R que compreende sua mensagem, mas não pode prover uma resposta;

# Database Performatives

- *insert* - S pede para R acrescentar o conteúdo da mensagem na base de conhecimento de R;
- *delete* - S pede para R deletar o conteúdo da mensagem da base de conhecimento de R;

# Capabilities Definition

- *advertise* - S quer que R saiba que S pode e processará mensagens do tipo da que está em seu conteúdo;

# Basic Effector Performatives

- *achieve* - S solicita que R tente fazer o conteúdo da mensagem vir a ser verdadeiro no sistema;
- *unachieve* - um *deny* de um *achieve*;

# Networking Performatives

- *register* - S anuncia para R (facilitador) sua presença e nome simbólico associado com seu endereço físico;
- *unregister* - cancela um register feito anteriormente;
- *transport-address* - S anuncia um novo endereço físico na rede;
- *forward* - S quer que o agente “:to” processe a performativa que está no parâmetro “:content” como se ela viesse diretamente do agente “:from”;
- *broadcast* - S pede a R para enviar a mensagem para todos agentes que R conhece;

# Facilitation Performatives

- *broker-one* - S pede a R para achar uma resposta para a performativa do seu conteúdo;
- *recommend-one* - S pede a R para sugerir um agente que possa processar seu conteúdo;

# Exemplos

- Capabilities Definition
- o agente A envia a seguinte mensagem para o agente B:  
(*advertise*

:sender           A  
:receiver        B  
:reply-with      id1  
:language        KQML  
:ontology        kqml-ontology  
:content         (*ask-if*

                  :sender           B  
                  :receiver        A  
                  :in-reply-to     id1  
                  :language        prolog  
                  :ontology        II-UFRGS  
                  :content         “professor(X,Y)”))

# Exemplos (cont.)

- Query Performatives
- o agente **B** pergunta então ao agente **A**:  
(*ask-if*

|              |                           |
|--------------|---------------------------|
| :sender      | B                         |
| :receiver    | A                         |
| :in-reply-to | id1                       |
| :reply-with  | id2                       |
| :language    | prolog                    |
| :ontology    | II-UFRGS                  |
| :content     | “professor(rafael, lc)” ) |

# Exemplos (cont.)

- Informative Performatives
- o agente A responde ao agente B com a mensagem:  
*(tell*

|              |                           |
|--------------|---------------------------|
| :sender      | A                         |
| :receiver    | B                         |
| :in-reply-to | id2                       |
| :reply-with  | id3                       |
| :language    | prolog                    |
| :ontology    | II-UFRGS                  |
| :content     | “professor(rafael, lc)” ) |

# Exemplos (cont.)

- Informative Performatives
- o agente A envia a seguinte mensagem para o agente B:

*(tell*            :sender            A  
                  :receiver        B  
                  :reply-with       id2  
                  :in-reply-to      id1  
                  :language         Prolog  
                  :ontology         II-UFRGS )

# Exemplos (cont.)

- Basic Responses
- em resposta, o agente **B** envia para o agente **A**:

*(error*       :sender        B  
              :receiver     A  
              :in-reply-to   id2  
              :reply-with    id3)

# Exemplos (cont.)

- Facilitation Performatives
- o **facilitador** recebe a seguinte mensagem:

```
(broker-one :sender      C
              :receiver   facilitador
              :reply-with id3
              :language   KQML
              :ontology   kqml-ontology
              :content     (ask-all :sender      C
                              :reply-with   id4
                              :language     Prolog
                              :ontology     II-UFRGS
                              :content      “professor(rafael,Y)” ) )
```

# Exemplos (cont.)

- Query Performatives
- então o agente **facilitador**, depois de procurar pelas mensagens “*advertise*” que recebeu, decide enviar a seguinte mensagem para o agente **A**:

(*ask-all*           :sender           facilitador  
                      :receiver         A  
                      :in-reply-to     id1  
                      :reply-with     id4  
                      :language        Prolog  
                      :ontology        II-UFRGS  
                      :content         “professor(rafael,Y)” )

# Exemplos (cont.)

- Informative Performatives
- e o agente A responde com a seguinte mensagem:

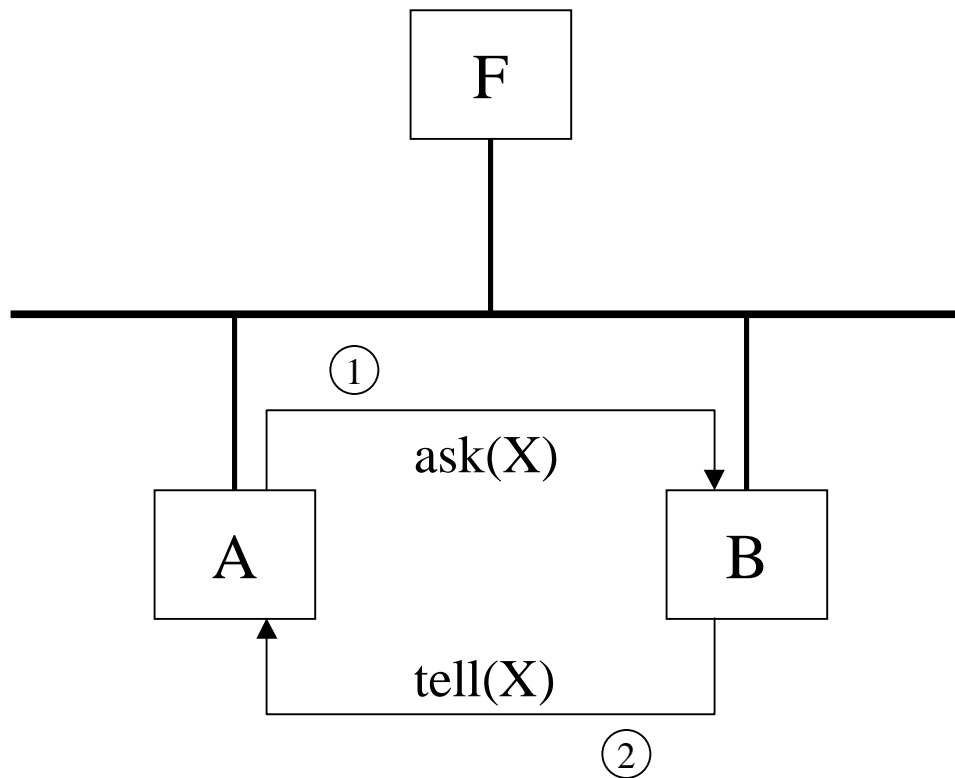
|              |              |   |
|--------------|--------------|---|
| <i>(tell</i> | :sender      | A   |
|              | :receiver    | facilitador   |
|              | :in-reply-to | id4   |
|              | :reply-with  | id5   |
|              | :language    | Prolog  |
|              | :ontology    | II-UFRGS  |
|              | :content     | “professor(rafael, lc), ... ,<br>professor(rafael, sf)” ) |

# Exemplos (cont.)

- Networking performatives:
- finalmente o agente **facilitador** envia para o agente **C**

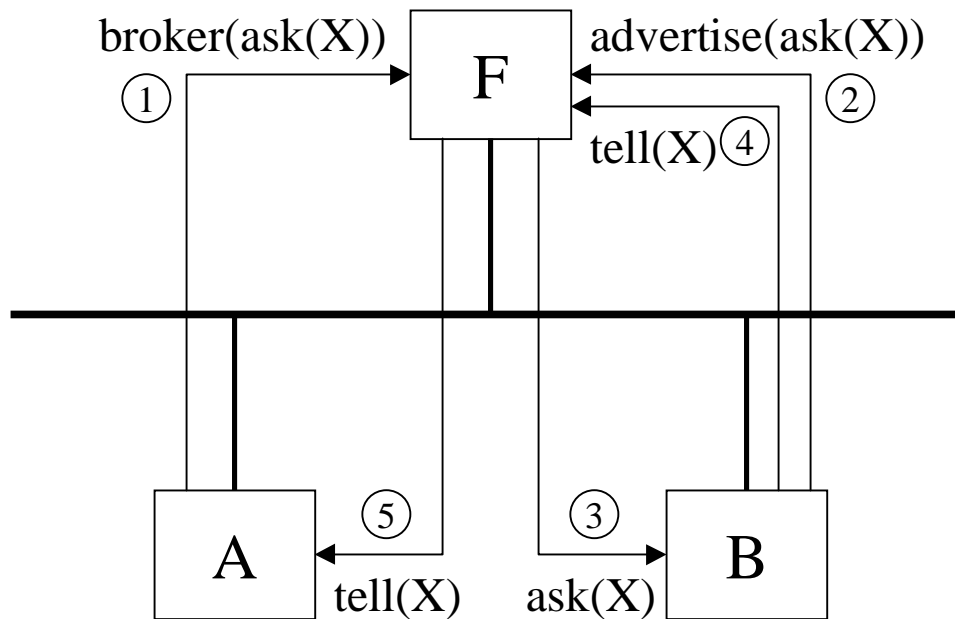
|                 |              |   |
|-----------------|--------------|---|
| <i>(forward</i> | :from        | A   |
|                 | :to          | C   |
|                 | :sender      | facilitador   |
|                 | :receiver    | C   |
|                 | :in-reply-to | id3   |
|                 | :reply-with  | id6   |
|                 | :language    | KQML  |
|                 | :ontology    | kqml-ontology   |
|                 | :content     | <i>(tell</i>  |
|                 |              | :receiver C   |
|                 |              | :language Prolog  |
|                 |              | :ontology II-UFRGS  |
|                 |              | :content  |
|                 |              | “professor(rafael, lc), ... ,<br>professor(rafael, sf)” ) |

# Protocolos KQML



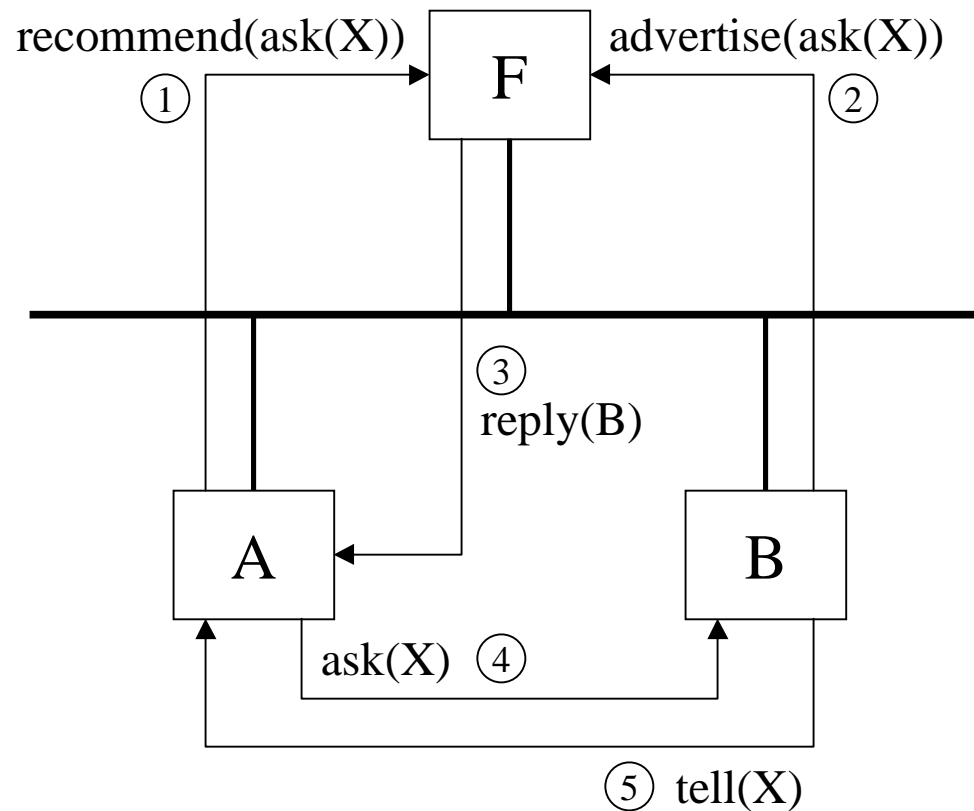
Protocolo usado quando A conhece B e que ele pode responder sobre X

# Protocolos KQML



Protocolo usando as performativas `broker` e `advertise`

# Protocolos KQML



Protocolo usando as performativas `recommend` and `advertise`

# Programação Orientada a Agentes BDI

# AgentSpeak(L)

- Linguagem criada por Rao (MAAMAW 1996)
- Formalização em Z feita por d'Inverno e Luck (JLC 1998)
- SIM\_Speak
  - Interpretador para AgentSpeak utilizando a plataforma de Sloman chamada SIM\_AGENT proposto por Machado e Bordini (ATAL 2001)
  - [http://www.inf.ufrgs.br/~bordini/SIM\\_Speak](http://www.inf.ufrgs.br/~bordini/SIM_Speak)

# AgentSpeak(L): Sintaxe

- Se  $\mathbf{b}$  é um símbolo predicativo, e  $\mathbf{t}_1, \dots, \mathbf{t}_n$  são termos, então:
  - $\mathbf{b}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  é um *átomo de crença*
  - $\mathbf{b}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  e  $\text{not } \mathbf{b}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  são *literais de crença*
- Se  $\mathbf{g}$  é um símbolo predicativo, e  $\mathbf{t}_1, \dots, \mathbf{t}_n$  são termos, então  $!\mathbf{g}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  e  $?\mathbf{g}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  são *objetivos*

# AgentSpeak(L): Sintaxe

- Se  $\mathbf{b}(\mathbf{t})$  é um átomo de crença,  $!g(\mathbf{t})$  e  $?g(\mathbf{t})$  são objetivos, então  $+\mathbf{b}(\mathbf{t})$ ,  $-\mathbf{b}(\mathbf{t})$ ,  $!g(\mathbf{t})$ ,  $?g(\mathbf{t})$ ,  $-!g(\mathbf{t})$  e  $-?g(\mathbf{t})$  são *triggering events*
- Se  $\mathbf{a}$  é um símbolo de ações e  $\mathbf{t}_1, \dots, \mathbf{t}_n$  são termos de primeira ordem, então  $\mathbf{a}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  é uma *ação*

# AgentSpeak(L): Sintaxe

- Se  $e$  é um triggering event,  $b_1, \dots, b_n$  são literais de crença, e  $h_1, \dots, h_n$  são objetivos, ações, adição ou remoção de crenças, então  
$$e : b_1 \ \& \ \dots \ \& \ b_m \ \leftarrow \ h_1 ; \ \dots \ ; \ h_n$$
é um *plano*
- Um programa AgentSpeak(L) é especificado como um conjunto de átomos de crença e um conjunto de planos

# AgentSpeak(L): Exemplo

- Em um hospital as enfermarias devem encaminhar pacientes para os ambulatórios para realização de exames
- Agentes
  - Enfermarias
  - Ambulatórios

# Eventos e Percepção do Ambiente

- Quando o médico envia uma requisição de exame, uma atualização das crenças dos agentes é feita através do predicado **requisitou(Medico, Paciente, Exame)**
- Quando o paciente **P** vai da enfermaria para o ambulatório, a crença **paciente\_disponivel(P)** é adicionada ao conjunto de crenças do agente ambulatório

# Enfermaria 1 (crenças)

```
exame_feito(barium_raioX,ambulatorio1).  
exame_feito(sangue_ex1,ambulatorio2).  
exame_feito(sangue_ex2,ambulatorio2).  
exame_feito(fisioterapia,ambulatorio3).  
nosso_paciente(paciente1).  
nosso_paciente(paciente2).  
nosso_nome(unidade1).
```

# Enfermaria 1 (planos)

```
+requisitou(Medico, Paciente, Exame) :
```

```
  nosso_paciente( Paciente ) <-
```

```
    ?teste_feito( Exame, Ambulatorio );
```

```
    ?nosso_name( Unidade );
```

```
    communicate( Ambulatorio,
```

```
      precisa_horario(Paciente,Unidade,  
        Exame) ).
```

# Enfermaria 1 (planos)

```
+horario_marcado( Paciente, Exame,  
  Ambulatorio, Horario ) :
```

```
  nosso_paciente(Paciente) &
```

```
  hora_atual(Hora) &
```

```
  not ( Hora > Horario ) <-
```

```
    +levar_paciente(Paciente,  
      Ambulatorio, Horario);
```

```
  -requisitou(Medico, Paciente, Exame) .
```

# Enfermaria 1 (planos)

`+hora_atual(Hora) :`

`levar_paciente(Paciente,Ambulatorio,  
Hora) &`

`paciente_disponivel(Paciente) <-`

`mover_paciente(Paciente,Ambulatorio).`

# Ambulatório 1 (crenças)

```
fazemos_exame(barium_raioX).  
nosso_nome(ambulatorio1).  
proximo_horario(8:00).  
tempo_por_paciente(00:15). // tempo que  
    só a aplicação de barium leva!  
intervalo_entre_exames(00:10).
```

# Ambulatório 1 (planos)

```
+precisa_horario(Paciente,Unidade,Exame) :  
  fazemos_exame(Exame) <-  
    ?proximo_horario(Horario);  
    +marcado(Paciente,Exame,Unidade,Horario);  
    -proximo_horario(Horario);  
    ?tempo_por_paciente(TP);  
    +proximo_horario( Horario + TP );  
    ?nosso_nome(Ambulatorio);  
    communicate(Unidade, marcado(Paciente,  
      Exame,Ambulatorio,Horario));  
    -precisa_horario(Paciente,Unidade,Exame).
```

# Ambulatório 1 (planos)

+hora\_atual(Hora) :

```
marcado(Paciente, Exame, Unidade, Horario) &  
paciente_disponivel(Paciente) <-
```

```
    aplicar_barium(Paciente);  
    ?intervalo_entre_exames(TI);  
    deixar_paciente_esperando(Paciente, TI);  
    aplicar_raioX(Paciente);  
    -marcado(Paciente, Exame, Unidade, Horario);  
    mover_paciente(Paciente, Unidade).
```