

# Inteligência Artificial Distribuída: Uma Introdução aos Sistemas Multiagentes

Rafael H. Bordini

Instituto de Informática

Universidade Federal do Rio Grande do Sul

[bordini@inf.ufrgs.br](mailto:bordini@inf.ufrgs.br)

# Programa

- Introdução aos Sistemas Multiagentes
- Arquitetura BDI
- Comunicação entre Agentes – KQML
- Programação Orientada a Agentes – AgentSpeak(L)

# Introdução aos Sistemas Multiagentes

# Caracterização da Área

- Indefinição do conceito de *agente*

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

- Sistemas Multiagentes

- Cognitivos

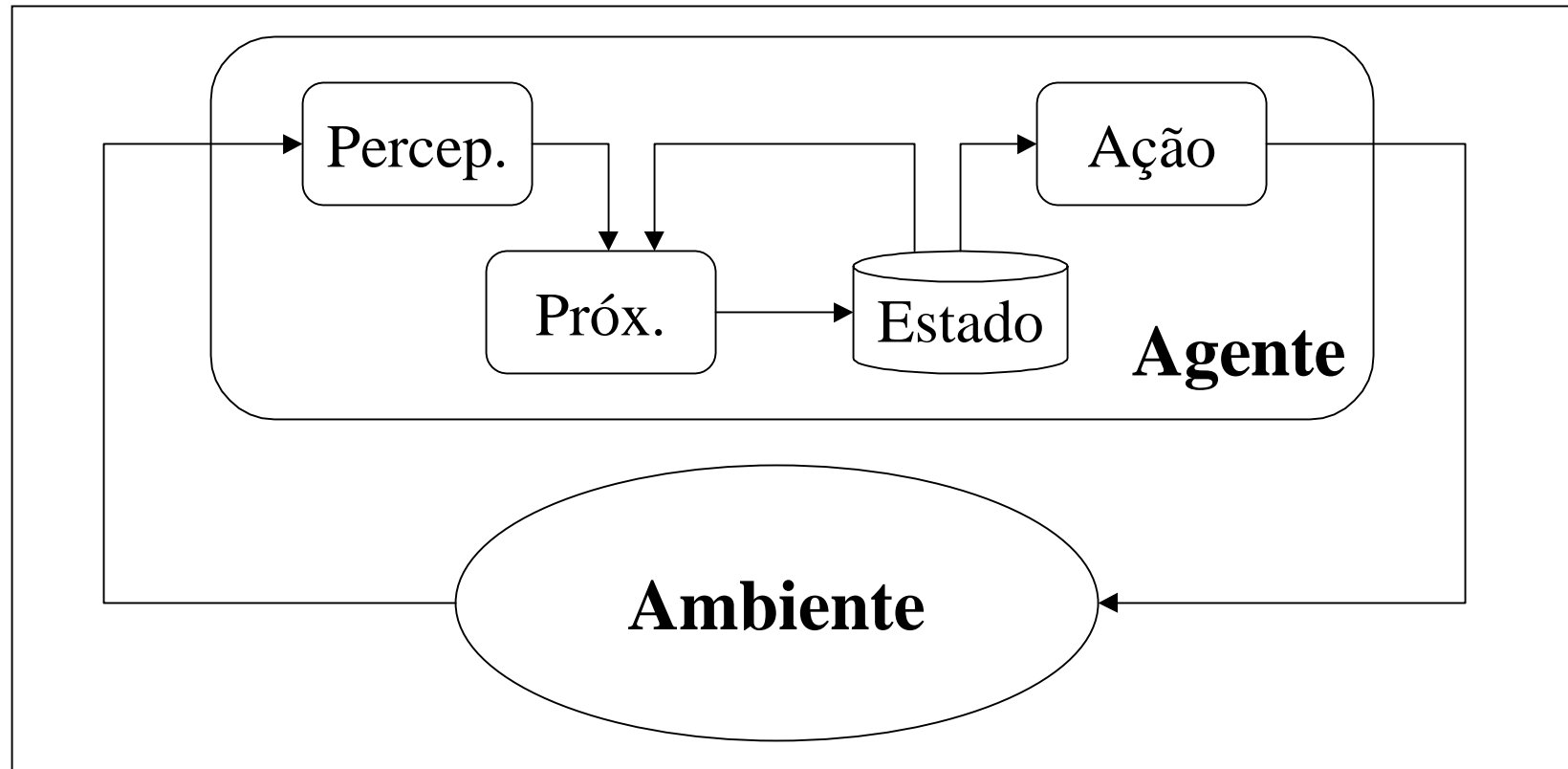
- Reativos

- Enfoques

- Teoria dos Jogos

- Lógica Matemática

# Modelo Geral de Agente



# Questões - SMA

- Cada agente é limitado - precisam interagir
- Não há controle central
- Autonomia
- Coordenação (cooperação/competição)
  - Teoria dos Jogos / Economia
  - Relações de Dependência
- Comunicação

# Motivação

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

# Trabalhos de Importância Histórica

- Comunicação (Cohen) (Werner)
- Poder Social (Castelfranchi)
- MAGMA (Demazeau)
- *Collective Misbeliefs* (Doran)
- Agent Oriented Programming (Shoham)
- Agentes Credíveis (Bates)
- Coordenação (Lesser)
- Embasamento Sociológico (Gasser)

# Aplicações

- Controle de Tráfego Aéreo (PRR)
- Indústria / Gerência de Negócios
- Interface Homem-Máquina
- Software Educativo
- Telecomunicações/Transporte/Eletricidade/Saúde
- Jogos/Entretenimento
- Internet: Busca e Gerência de Informações (PDA)/  
Comércio Eletrônico
- Simulação Social

# Simulação Social

- Simulação de Sociedades (inclusive humanas):
  - Ciências Sociais: da sociologia à economia, da psicologia social à teoria das organizações e ciências políticas
- 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

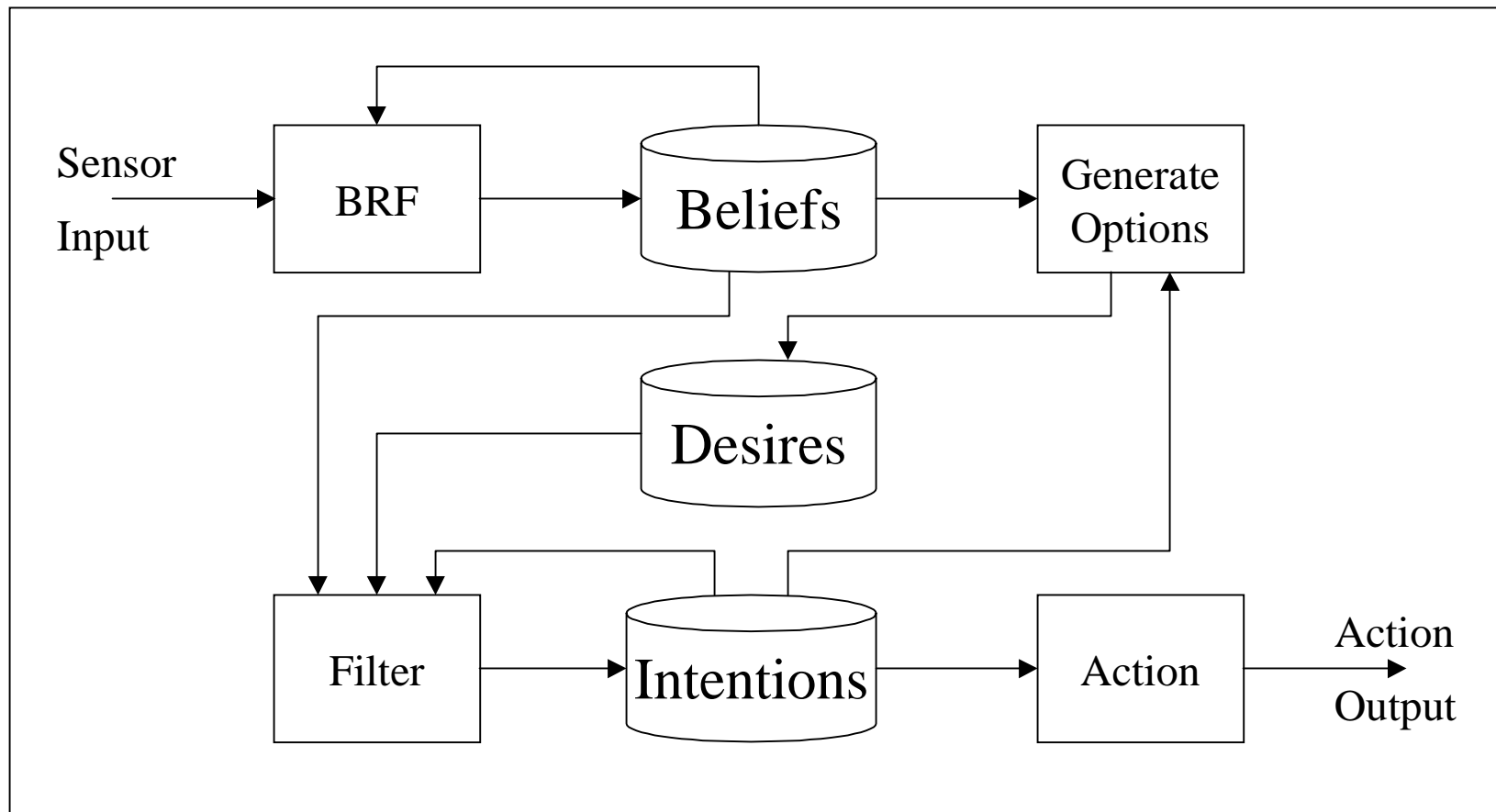
- Problemas do enfoque de SMA para SS:
  - Capacitação da equipe
  - Que tipo de modelo? (GT, CogSci)
  - Qual nível de abstração?
  - Controle de uma enorme quantidade de parâmetros
  - Validação!

# Arquitetura BDI

# Raciocínio Prático

- Intentional Stance (Dennett)
- Practical Reasoning (Bratman, Isreal, Pollack)
- Arquitetura BDI (Georgeff, Rao)

# Arquitetura BDI



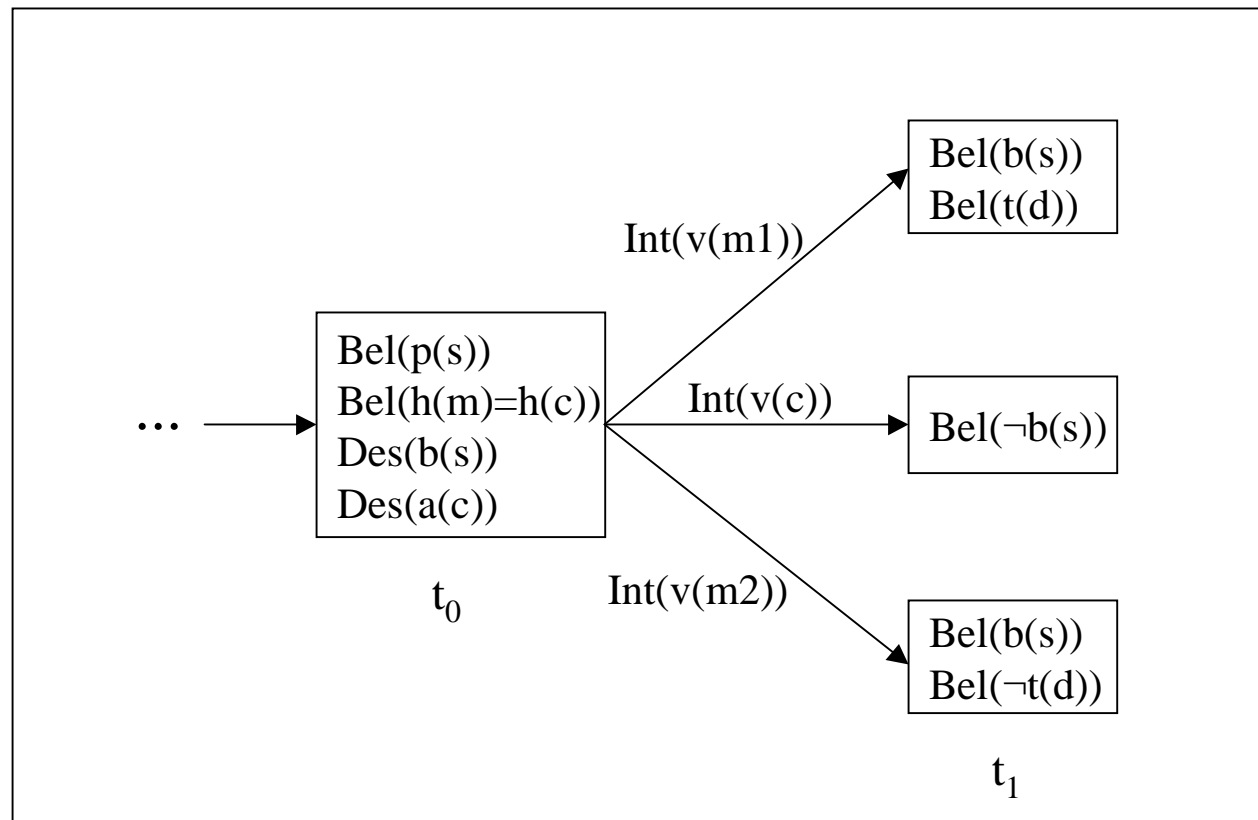
# Interpretador BDI

```
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.
```

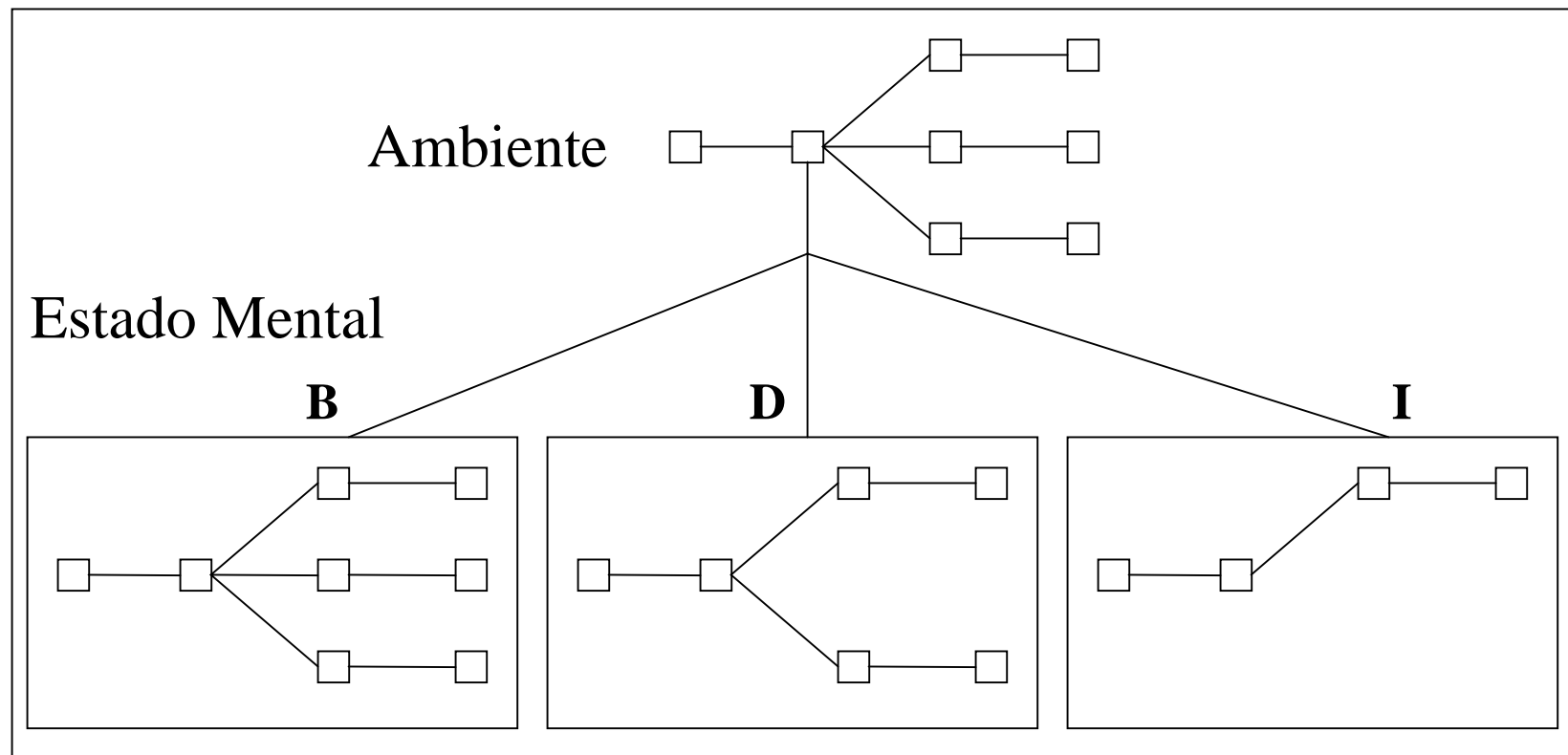
# Lógica BDI

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

# Lógica BDI



# Modelo de Mundos Possíveis e Tempo Ramificado



# Raciocínio Prático

- Princípios:
  - Inconsistência Intenção-Crença: é irracional um agente ter a intenção de fazer uma ação, e também acreditar que ele não vai fazê-la.
  - Incompleteza Intenção-Crença: é racional um agente ter a intenção de fazer uma ação, mas não acreditar que vai fazê-la.
  - Incompleteza Crença-Intenção: é racional um agente acreditar que uma ação é possível, sem necessariamente ter a intenção de fazê-la.

# Exercício

- Remodelar algum sistema já desenvolvido utilizando a idéia de *agentes*
- Especificar para cada agente suas:
  - crenças
  - desejos
  - intenções (planos)

# 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.

KQML

# 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**:

<i>(ask-all</i>	: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:

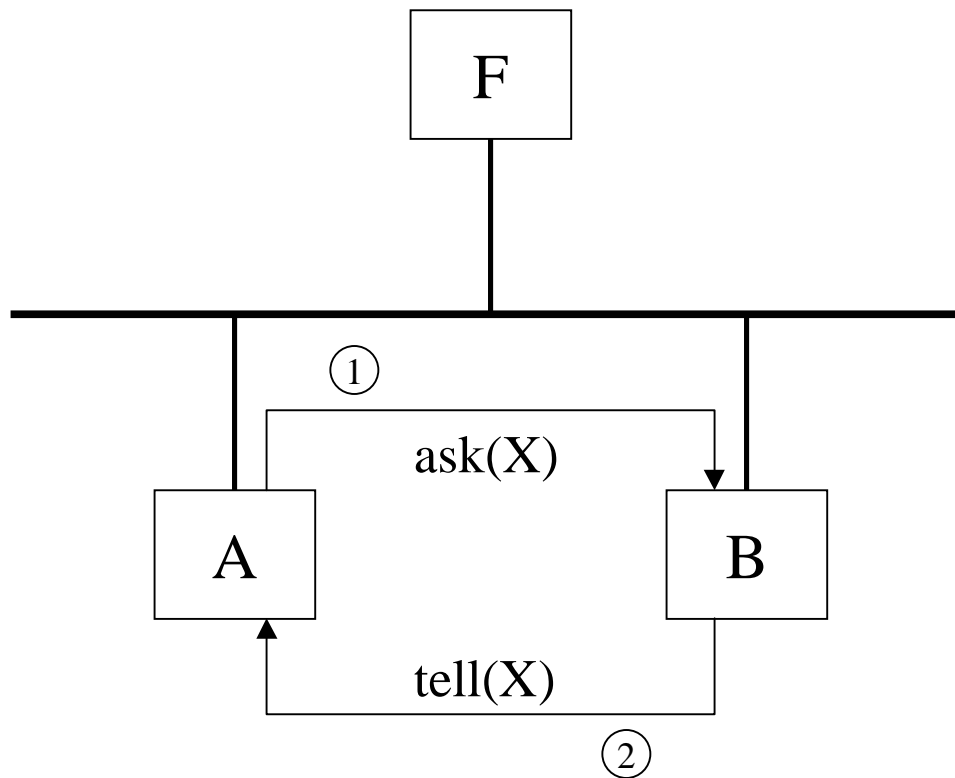
<i>(tell</i>	:sender	A
	:receiver	facilitador
	:in-reply-to	id4
	:reply-with	id5
	:language	Prolog
	:ontology	II-UFRGS
	:content	“professor(rafael, lc), ... , professor(rafael, sf)” )

# Exemplos (cont.)

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

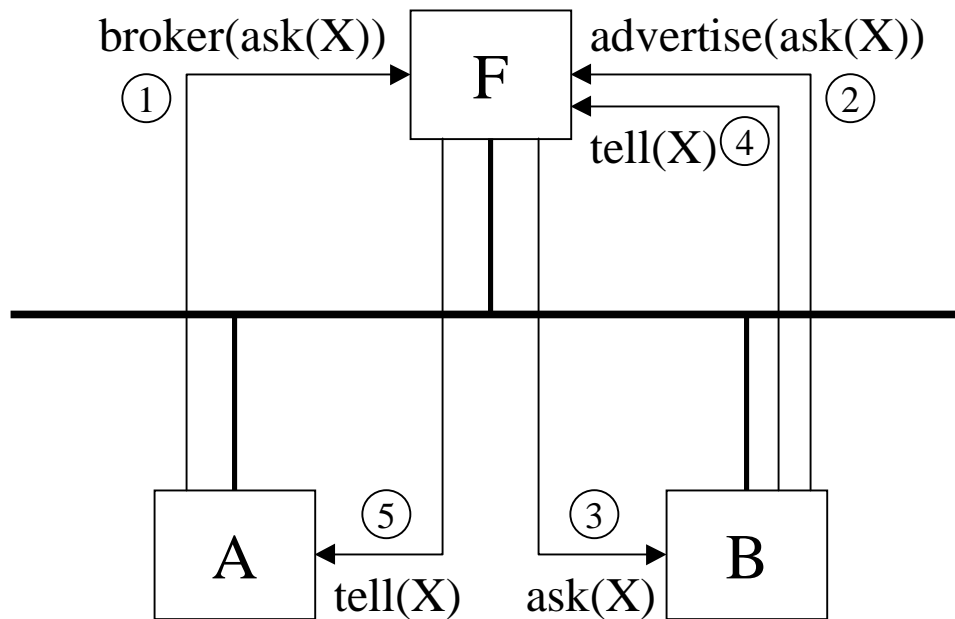
<i>(forward</i>	:from	A
	:to	C
	:sender	facilitador
	:receiver	C
	:in-reply-to	id3
	:reply-with	id6
	:language	KQML
	:ontology	kqml-ontology
	:content	<i>(tell</i>
		:receiver C
		:language Prolog
		:ontology II-UFRGS
		:content
		“professor(rafael, lc), ... , professor(rafael, sf)” )

# Protocolos KQML



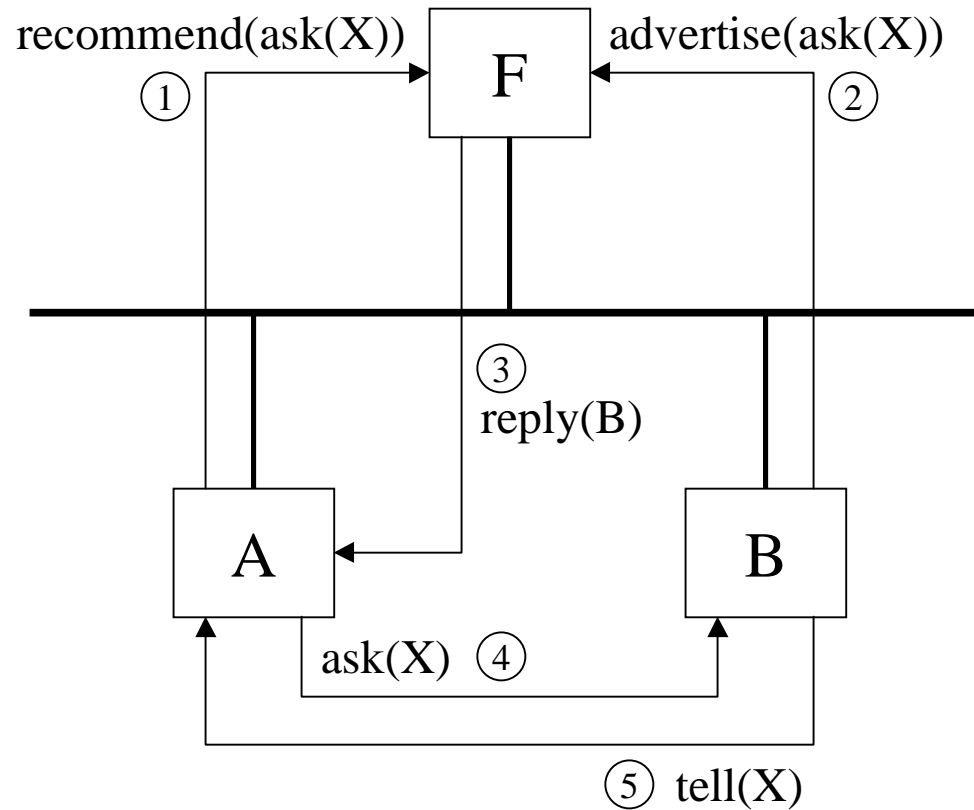
Protocolo usado quando A conhece B e que ele pode responder sobre X

# Protocolos KQML



Protocolo usando as performativas `broker` e `advertise`

# Protocolos KQML



Protocolo usando as performativas `recommend` and `advertise`

# Programação Orientada a Agentes

# 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 (Machado e Bordini 2001)
- [http://www.inf.ufrgs.br/~bordini/SIM\\_Speak](http://www.inf.ufrgs.br/~bordini/SIM_Speak)

# AgentSpeak(L): Sintaxe

- If **b** is a predicate symbol, and  $t_1, \dots, t_n$  are terms, **b(t1, ..., tn)** is a *belief atom*,  
**not b(t1, ..., tn)** and  
**b(t1, ..., tn) & c(t1, ..., tn)** are *beliefs*
- If **g** is a predicate symbol, and  $t_1, \dots, t_n$  are terms, **!g(t1, ..., tn)** and  
**?g(t1, ..., tn)** are *goals*

# AgentSpeak(L): Sintaxe

- If  $\mathbf{b}(\mathbf{t})$  is a belief atom,  $!\mathbf{g}(\mathbf{t})$  and  $?\mathbf{g}(\mathbf{t})$  are goals, then  $+\mathbf{b}(\mathbf{t})$ ,  $-\mathbf{b}(\mathbf{t})$ ,  $+\mathbf{!g}(\mathbf{t})$ ,  $+\mathbf{?g}(\mathbf{t})$ ,  $-\mathbf{!g}(\mathbf{t})$  and  $-\mathbf{?g}(\mathbf{t})$  are *triggering events*
- If  $\mathbf{a}$  is an action symbol and  $\mathbf{t}_1, \dots, \mathbf{t}_n$  are first order terms, then  $\mathbf{a}(\mathbf{t}_1, \dots, \mathbf{t}_n)$  is an *action*

# AgentSpeak(L): Sintaxe

- If  $e$  is a triggering event,  $b_1, \dots, b_n$  are belief literals, and  $h_1, \dots, h_n$  are goals or actions, then

$e : b_1 \ \& \ \dots \ \& \ b_m \ \leftarrow \ h_1 ; \ \dots \ ; \ h_n$

is a *plan*

- An AgentSpeak(L) program is specified as a set of beliefs and a set of plans

# 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 **requested(Doctor, Patient, Test)**
- Quando o paciente **P** vai da enfermaria para o ambulatório, a crença **patient\_available(P)** é adicionada ao conjunto de crenças do agente ambulatório

# Enfermaria 1 (crenças)

## Beliefs

```
test_done_at( barium_XRay, ancillary1 ).
test_done_at( blood_test1, ancillary2 ).
test_done_at( blood_test2, ancillary2 ).
test_done_at( physical_therapy,
  ancillary3 ).
our_patient( patient1 ).
our_patient( patient2 ).
our_name( unit1 ).
```

# Enfermaria 1 (planos)

## Plans

```
+requested(Doctor, Patient, Test) :  
  
  our_patient( Patient ) <-  
  
    ?test_done_at( Test, Ancillary );  
    ?our_name( Unit );  
    communicate( Ancillary,  
    needs_schedule(Patient,Unit,Test)).
```

# Enfermaria 1 (planos)

```
+scheduled( Patient, Test, Ancillary,  
  Time ) :  
  
  our_patient( Patient ) &  
  ( not past(Time) ) <-  
  
    +take_patient( Patient, Ancillary,  
      Time);  
    -requested( Doctor, Patient, Test ).
```

# Enfermaria 1 (planos)

```
+current_time( Time ) :
```

```
take_patient(Patient,Ancillary,Time) &  
patient_available( Patient ) <-
```

```
move_patient( Patient, Ancillary ).
```

# Enfermaria 2

- Mudam apenas as crenças do tipo:

`our_patient(_)`.

- **E a crença:**

`our_name(unit2)`.

# Ambulatório 1 (crenças)

## Beliefs

```
we_do_test( barium_XRay ).  
our_name( ancillary1 ).  
next_free_slot( 8:00 ).  
time_per_patient( 00:15 ). // time that  
    just the barium test takes!  
delay_between_tests( 00:10 ).
```

# Ambulatório 1 (planos)

## Plans

```
+needs_schedule( Patient, Unit, Test ) :  
  we_do_test(Test) <-  
    ?next_free_slot( Time );  
    +scheduled( Patient, Unit, Test, Time );  
    -next_free_slot( Time );  
    ?time_per_patient( TP );  
    +next_free_slot( Time + TP );  
    ?our_name(Ancillary);  
    communicate( Unit,  
      scheduled(Patient,Test,Ancillary,Time) );  
    -needs_schedule( Patient, Unit, Test ).
```

# Ambulatório 1 (planos)

```
+current_time( Time ) :  
  scheduled( Patient,Unit,Test,Time ) &  
  patient_available( Patient ) <-  
    barium_test( Patient );  
    ?delay_between_tests(TD);  
  keep_patient( TD );  
  xRay( Patient );  
  -scheduled(Patient,Unit,Test,Time);  
  send_back( Patient, Unit ).
```

# Ambulatório 2 (crenças)

## Beliefs

```
we_do_test( blood_test1 ).  
we_do_test( blood_test2 ).  
our_name( ancillary2 ).  
next_free_slot( 8:00 ).  
time_per_patient( 0:05 ). // time to  
draw blood only!
```

# Ambulatório 2 (planos)

## Plans

(o mesmo plano `+needs_schedule` acima)

```
+current_time( Time ) :  
    scheduled( Patient,Unit,AnyTest,Time ) &  
    (not blood_sample_available(Patient)) &  
    patient_available( Patient ) <-  
        draw_blood( Patient );  
    +blood_sample_available( Patient ).
```

# Ambulatório 2 (planos)

```
+current_time( Time ) :  
  scheduled( Patient, Unit, test1,  
            AnyTime ) &  
  blood_sample_available( Patient ) <-  
  
  do_test1( Patient );  
  -scheduled( Patient, Unit, test1,  
            AnyTime ).
```

# Ambulatório 2 (planos)

```
+current_time( Time ) :  
  scheduled( Patient, Unit, test2,  
            AnyTime ) &  
  blood_sample_available( Patient ) <-  
  
  do_test2( Patient );  
  -scheduled( Patient, Unit, test2,  
            AnyTime ).
```

# Ambulatório 2 (planos)

```
+blood_sample_available( Patient ):  
  
  patient_available( Patient ) <-  
  
    send_back( Patient, Unit ).
```

# Ambulatório 3 (crenças)

## Beliefs

```
we_do_test( physical_therapy ).  
our_name( ancillary3 ).  
next_free_slot( 8:00 ).  
time_per_patient( 01:00 ).
```

# Ambulatório 3 (planos)

## Plans

(o mesmo plano `+needs_schedule` acima)

```
+current_time( Time ) :  
    scheduled( Patient,Unit,Test,Time ) &  
    patient_available( Patient ) <-  
    physical_therapy( Patient );  
    -scheduled(Patient,Unit,Test,Time);  
    send_back( Patient, Unit ).
```

# Exercício

- Especificar o mesmo sistema anterior utilizando AgentSpeak(L) para cada agente
- Utilizar mensagens em KQML para comunicação entre os agentes, se houver necessidade