ENUCOMP
ENCONTRO UNIFICADO DE COMPUTAÇÃO EM PARNAÍBA

Minicurso 4

Introdução à Engenharia de Software Orientada a Agentes com JaCaMo

Nécio de Lima Veras (IFCE),
Anderson Couto Palácio de Queiroz (UECE),
Francisco Robson Oliveira de Lima (UECE),
Robert Marinho da Rocha Júnior (UECE),
Igor Brasil Nogueira (UECE),
Mariela Inés Cortés (UECE) e
Gustavo Augusto Lima de Campos (UECE)

Parceria

Universidade Estadual do Ceará (UECE)

INSTITUTO FEDERAL
CEARÁ
Campus Tianguá

Suporte financeiro

CAPES

Agenda

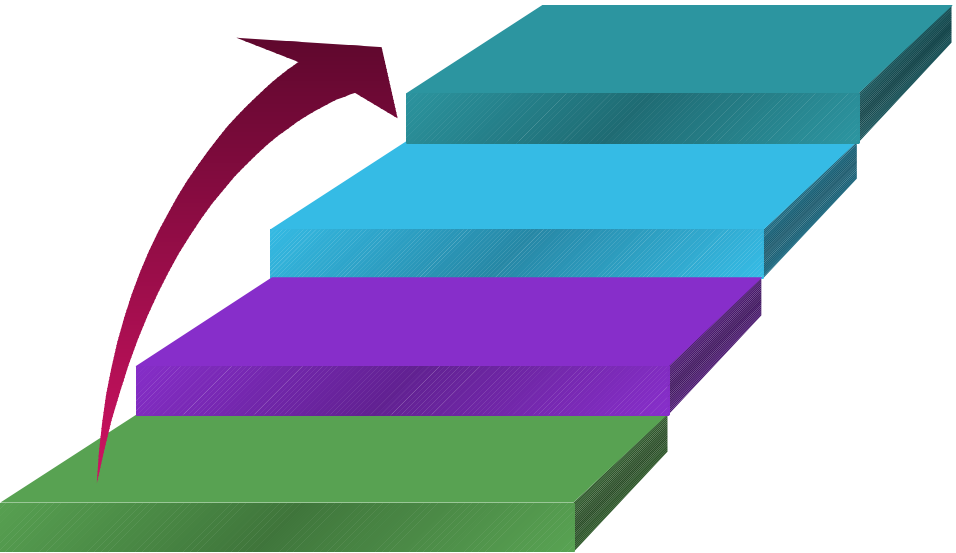
- Introdução
- Engenharia de Software Orientada a Agentes (ESOA)
- Plataforma de desenvolvimento de Agentes
- Ambientes de Agentes
- Estudos de casos
- Introdução aos Sistemas Multi-Agentes

Introdução

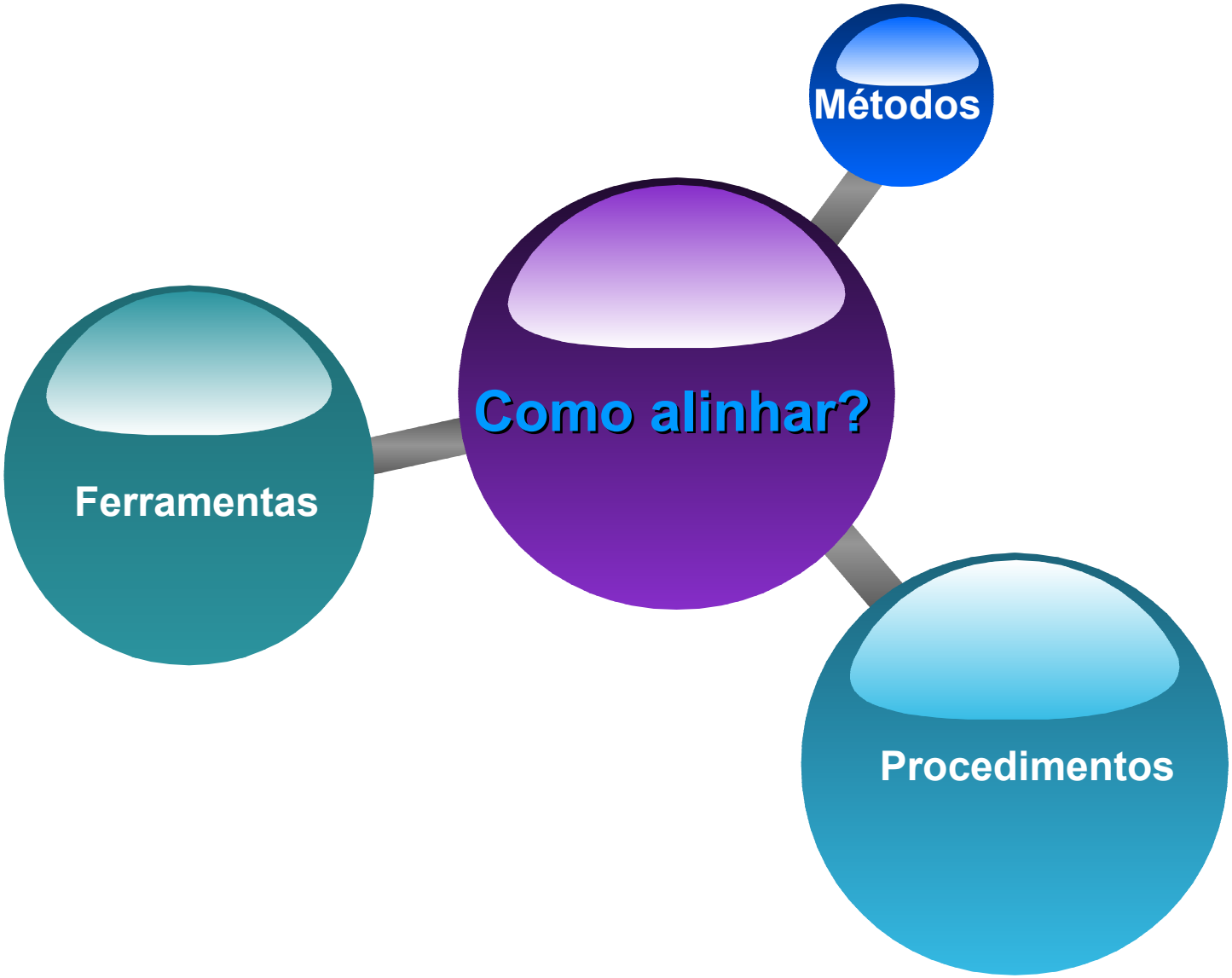
Aumento da Complexidade

Complexidade de Requisitos

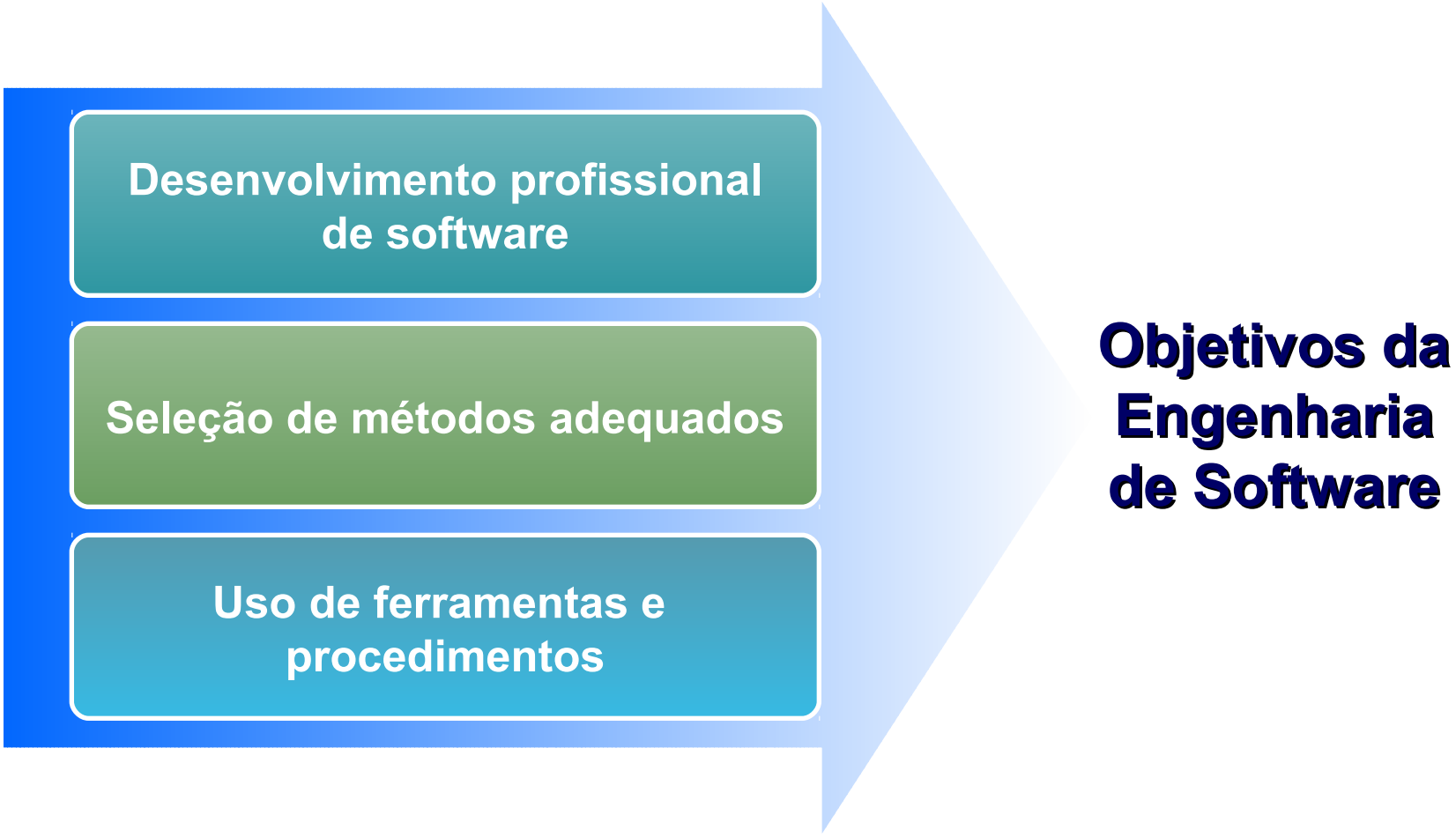
Complexidade de Software



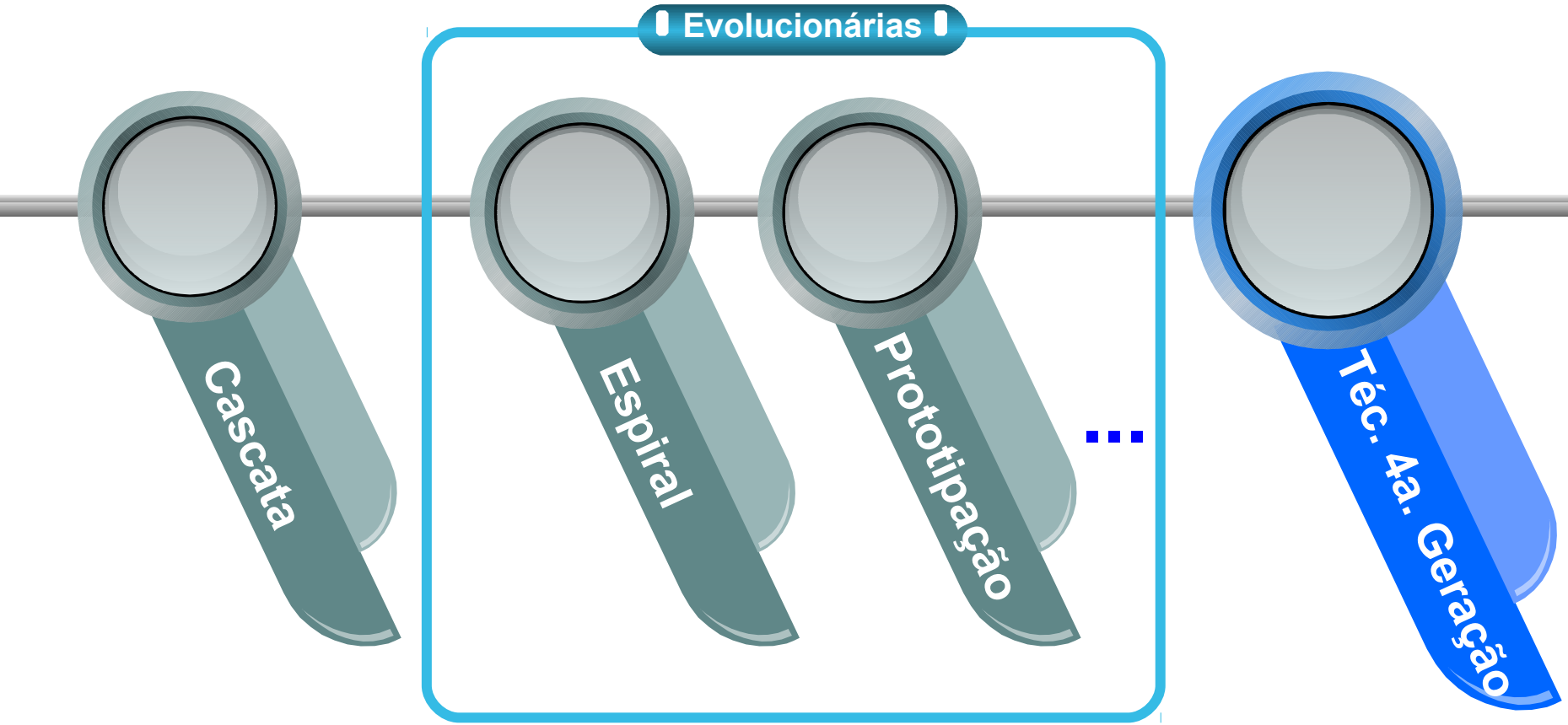
Alinhamento da produção



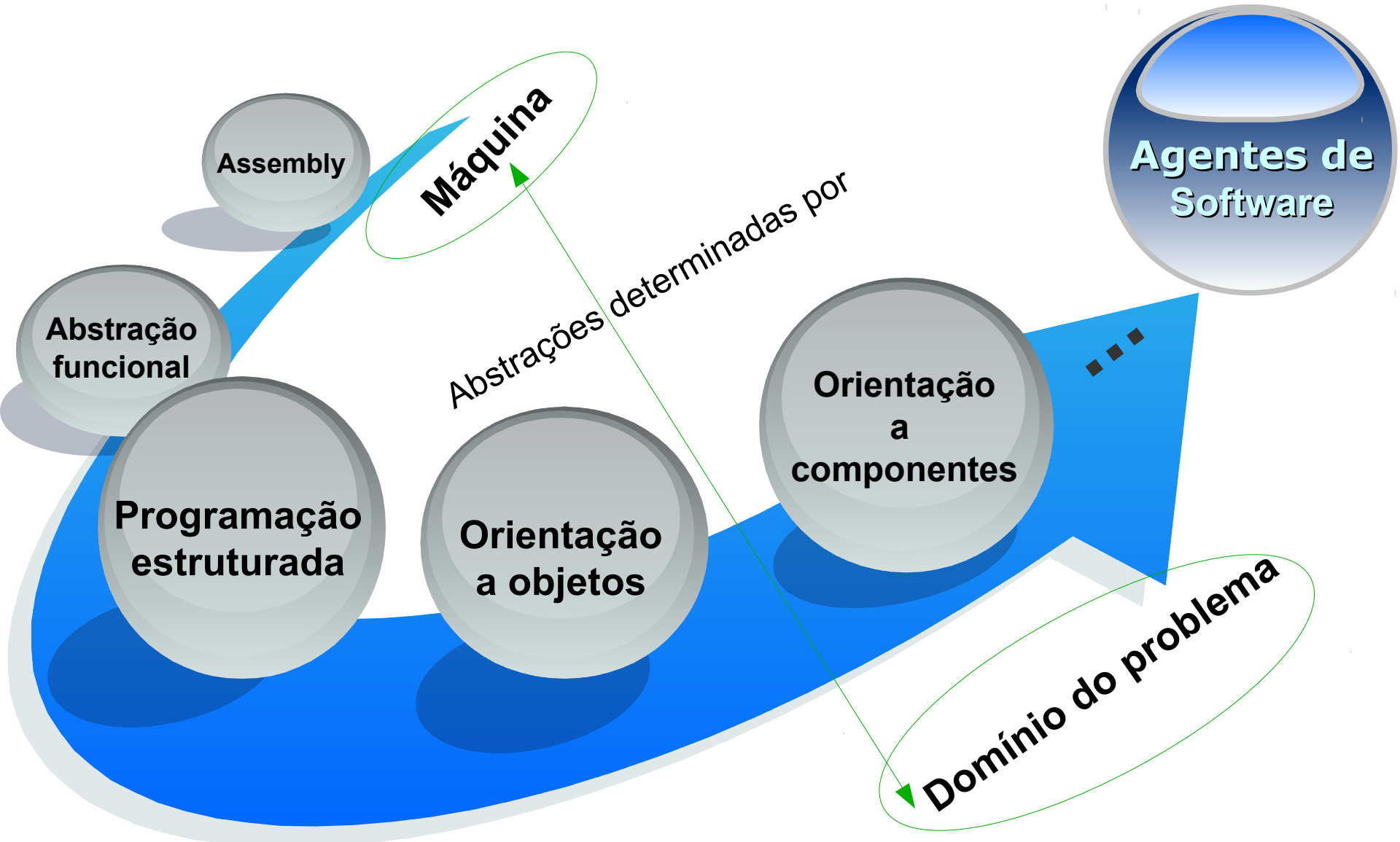
Engenharia de software



Evolução dos paradigmas



Evolução dos paradigmas



Engenharia de Software Orientado a Agentes

- **Conceitos fundamentais sobre Agentes Inteligentes**
- **Arquiteturas**
- **O modelo BDI**

Conceitos sobre Agentes Inteligentes

1. Um agente é um sistema informático situado em um ambiente, e que é capaz de realizar ações de forma autônoma para atingir seus objetivo [Wooldridge 1997]

2. Uma entidade capaz de perceber um ambiente através de sensores e atuar sobre o mesmo através de atuadores [Russel e Norvig 2009]

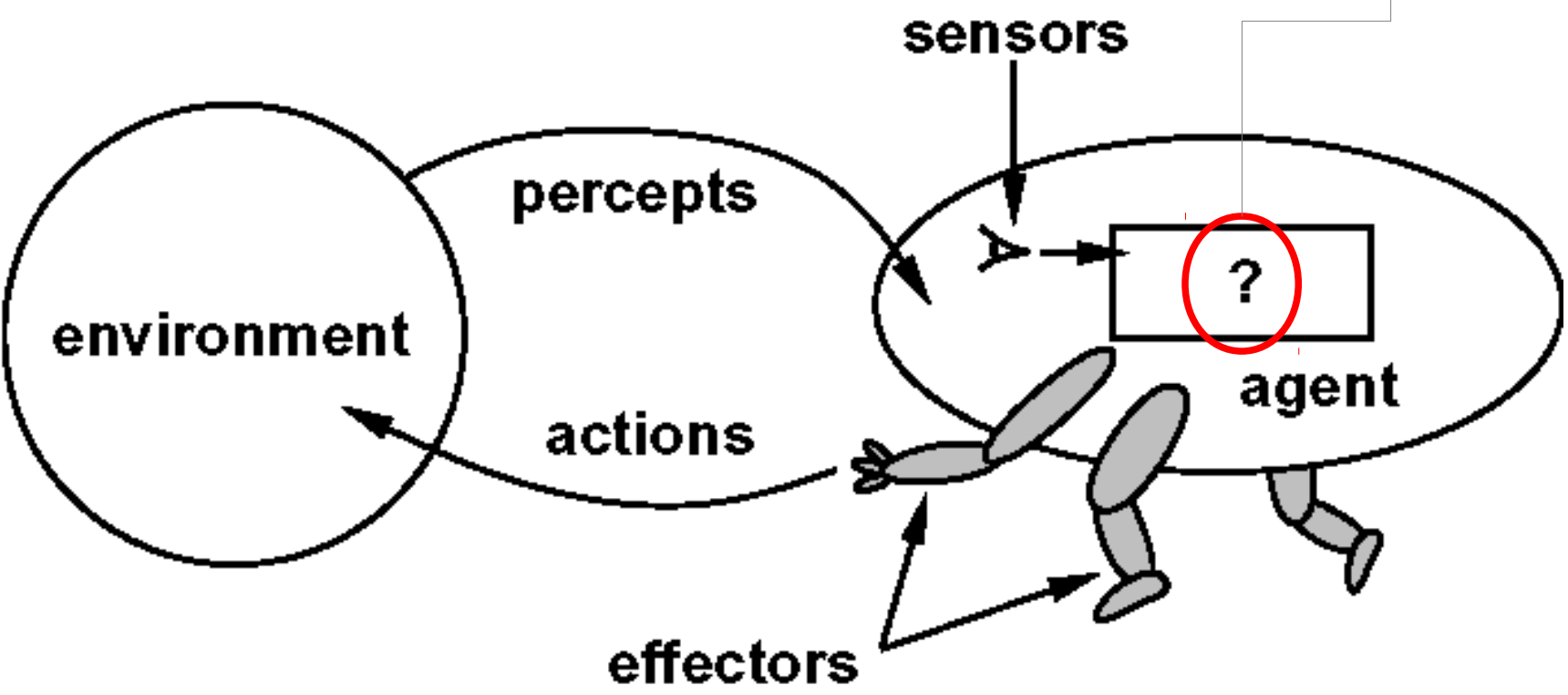
3. "Qualquer coisa" que pode **perceber** seu ambiente através de sensores e **interagir** com ele por meio de atuadores.

Exemplos

- O ser humano possui visão e audição como sensores e mão e pernas como atuadores;
- Um robô possui câmeras e localizadores como sensores e motores como atuadores;
- Um software.....

Em imagem

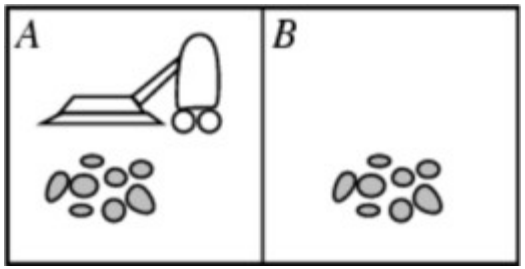
Técnicas de Inteligência Computacional



[Russel e Norvig 1995]

Agent Function

- É o **comportamento** de um agente
  - Serve para mapear qualquer sequência de percepção para uma ação;
  - Pode ser descrito com uma tabela de percepções e ações;
  - Exemplo:



Agente Aspirador de Pó

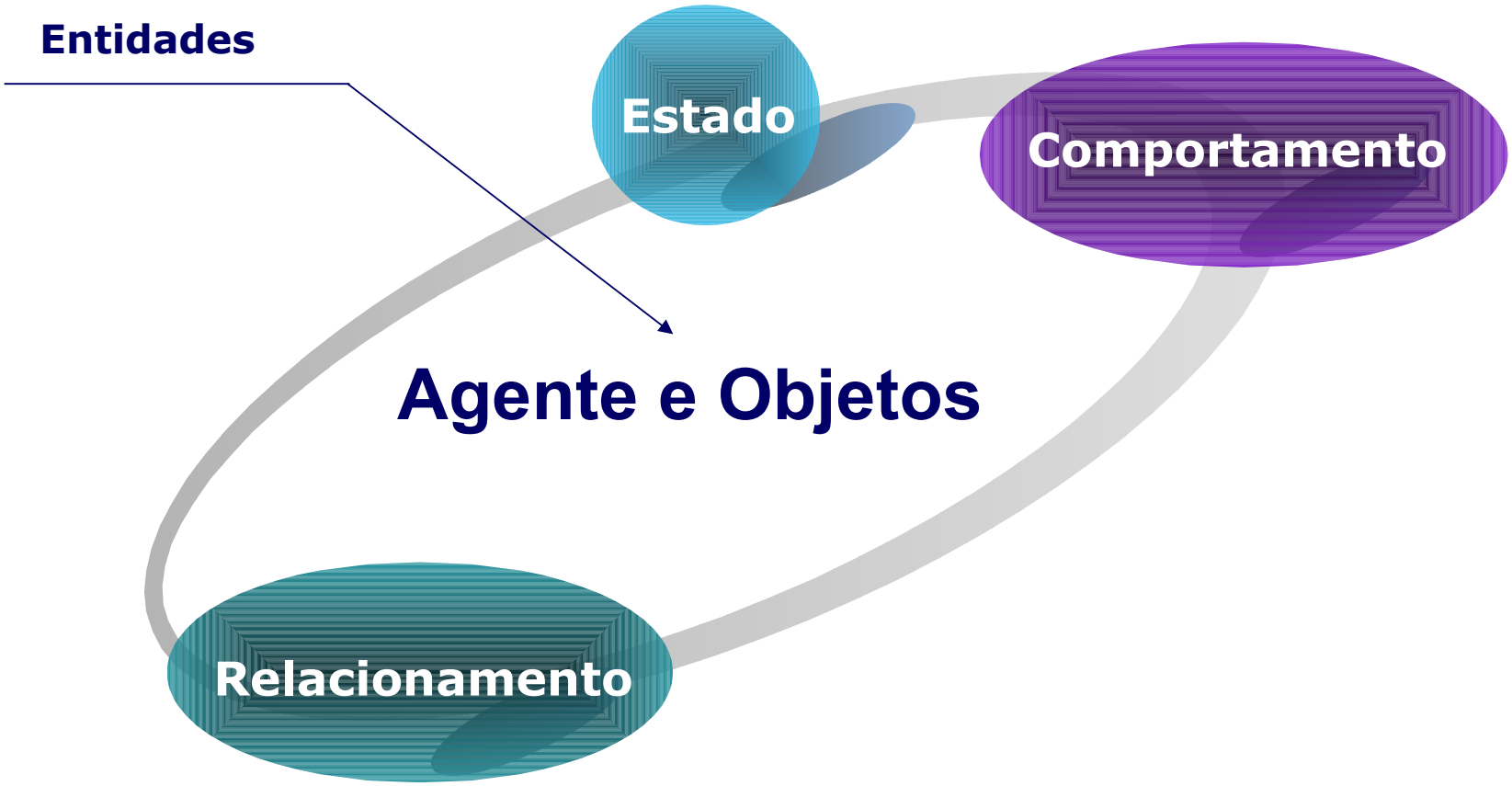
Sequência de Percepções	Ação
[A, Limpo]	Direita
[A, Sujo]	Aspirar
[B, Limpo]	Esquerda
[B, Sujo]	Aspirar
[A, Limpo], [A, Limpo]	Direita
[A, Limpo], [A, Sujo]	Aspirar
...	
[A, Limpo], [A, Limpo], [A, Limpo]	Direita
[A, Limpo], [A, Limpo], [A, Sujo]	Aspirar
...	

Tabela de funções do agente

Quando usar agentes de software?

- Quando a tarefa é grande e complexa;
- Quando é necessário que as decisões sejam feitas com rapidez;
- Quando existem riscos envolvendo as pessoas;
- Quando é muito caro ou difícil manter um grupo de pessoas controlando um software (ou um robô);

Agentes vs Objetos



Objetos

Atributos

Armazena informações sobre os objetos ou outras entidades

Métodos

São tarefas, operações que podem ser invocadas e modificar os atributos

Entre Objetos

Troca de mensagens, chamada ou execução de métodos

Possuem comportamento estático e controle parcial sobre seu estado

São entidades passivas

Agentes

Crenças objetivos, planos e ações

Comportamento "estado mental"

Os planos são executados conforme os objetivos

Comunicação

Envio de mensagens usando uma linguagem de comunicação

Possuem comportamento dinâmico e controle total sobre seu estado e comportamento (modificação e aprendizagem)

São entidades ativas

Algumas propriedades dos agentes

Agent program

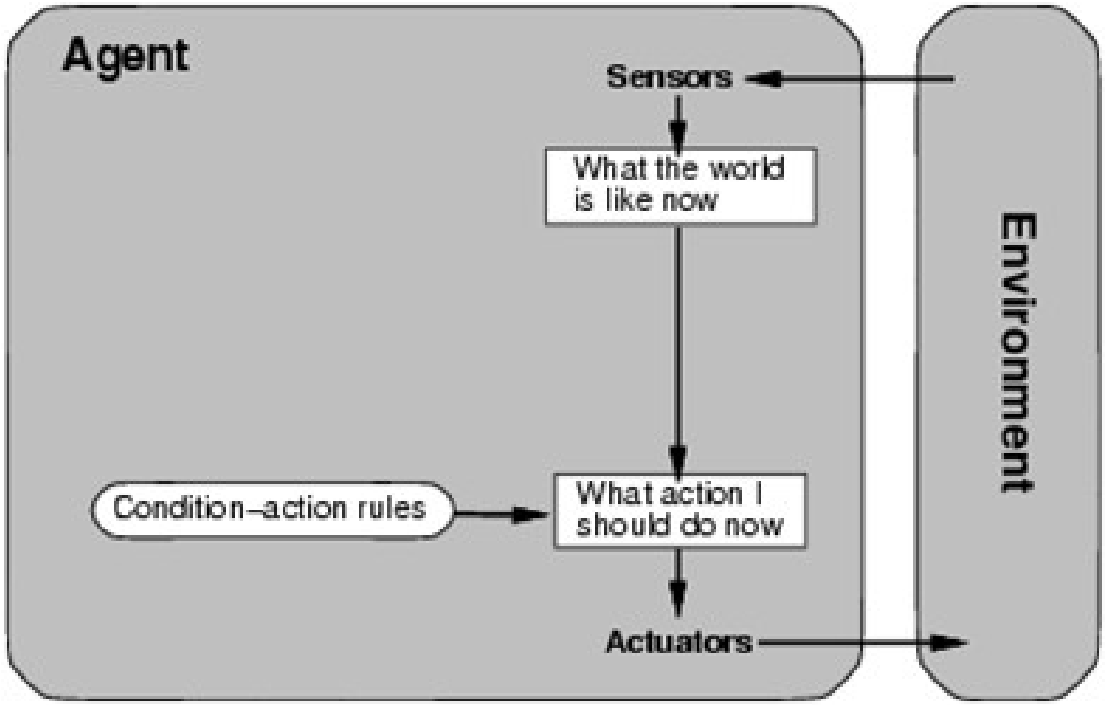
- **Materializa** o comportamento do agente;
- Internamente, **implementa** o *agent function*;
- **Uma arquitetura** de agente que é capaz de por em funcionamento o agente;

Arquiteturas de Agentes

- Agentes reativos
- Agentes baseados em modelos
- Agentes baseados em objetivos
- Agentes baseados em utilidade

[Russel e Norvig 2009]

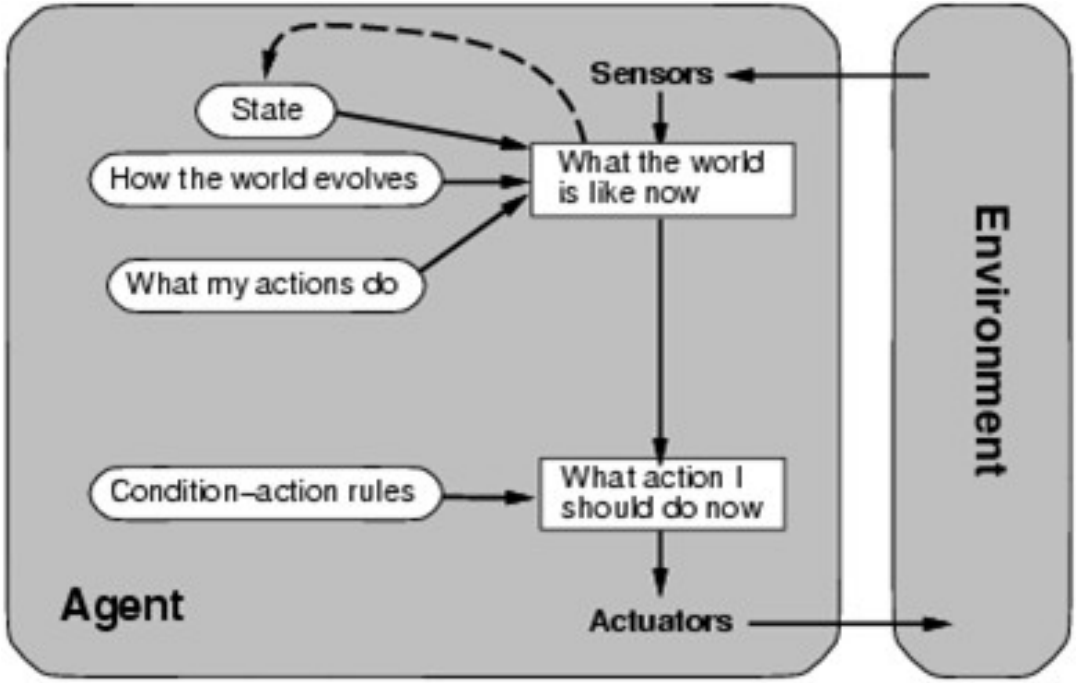
Agentes reativos



Características

- Simplicidade!
- Inteligência limitada
- Percebe => Seleciona ação
- É um problema sério caso o agente não perceba nada.

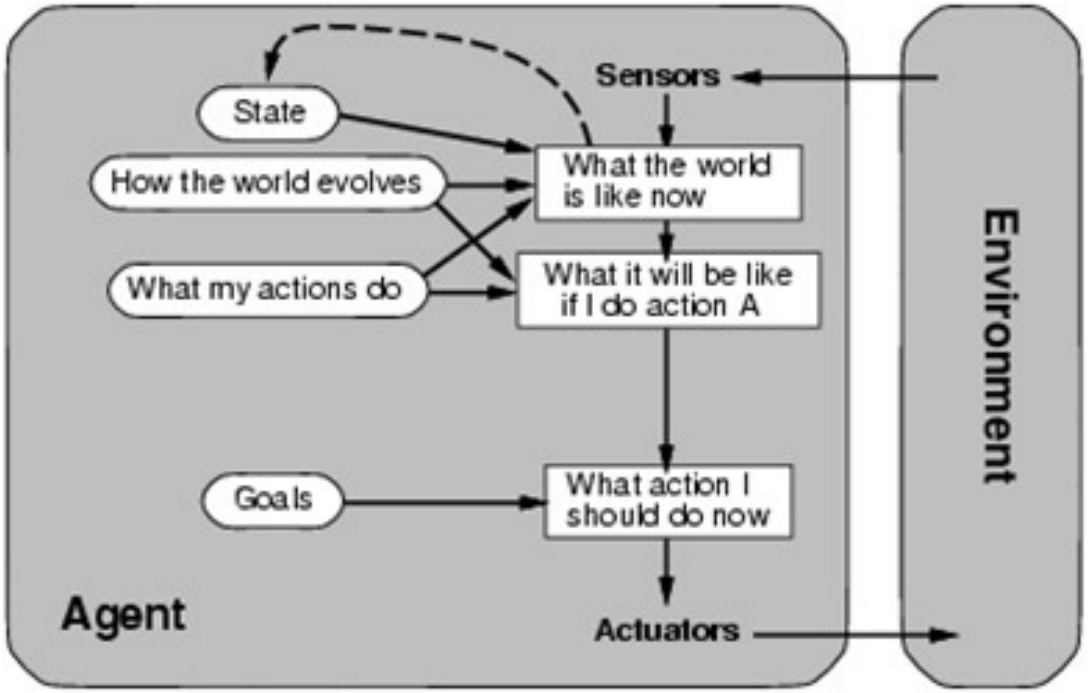
Agente baseado em modelo



Características

- Memória de percepções (estado interno)
- Possui conhecimento sobre o funcionamento do mundo

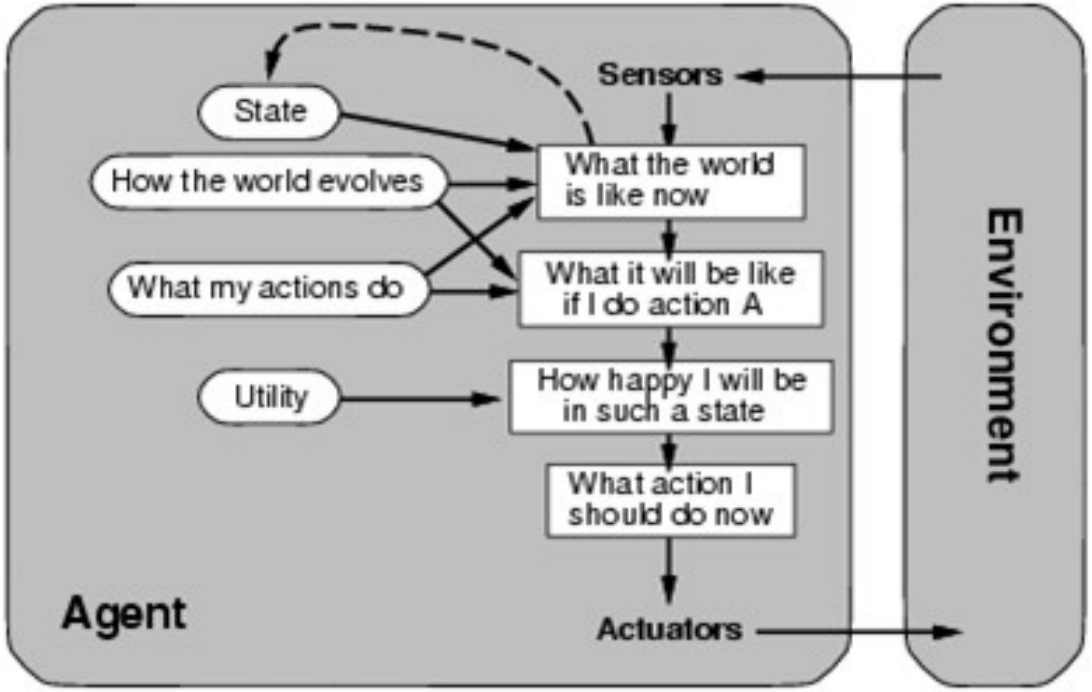
Agente baseado em objetivo



Características

- Possui um estado desejável (objetivo)
- Considera o que pode acontecer no futuro
- Pode selecionar um plano ao invés de uma ação

Agente baseado em utilidade



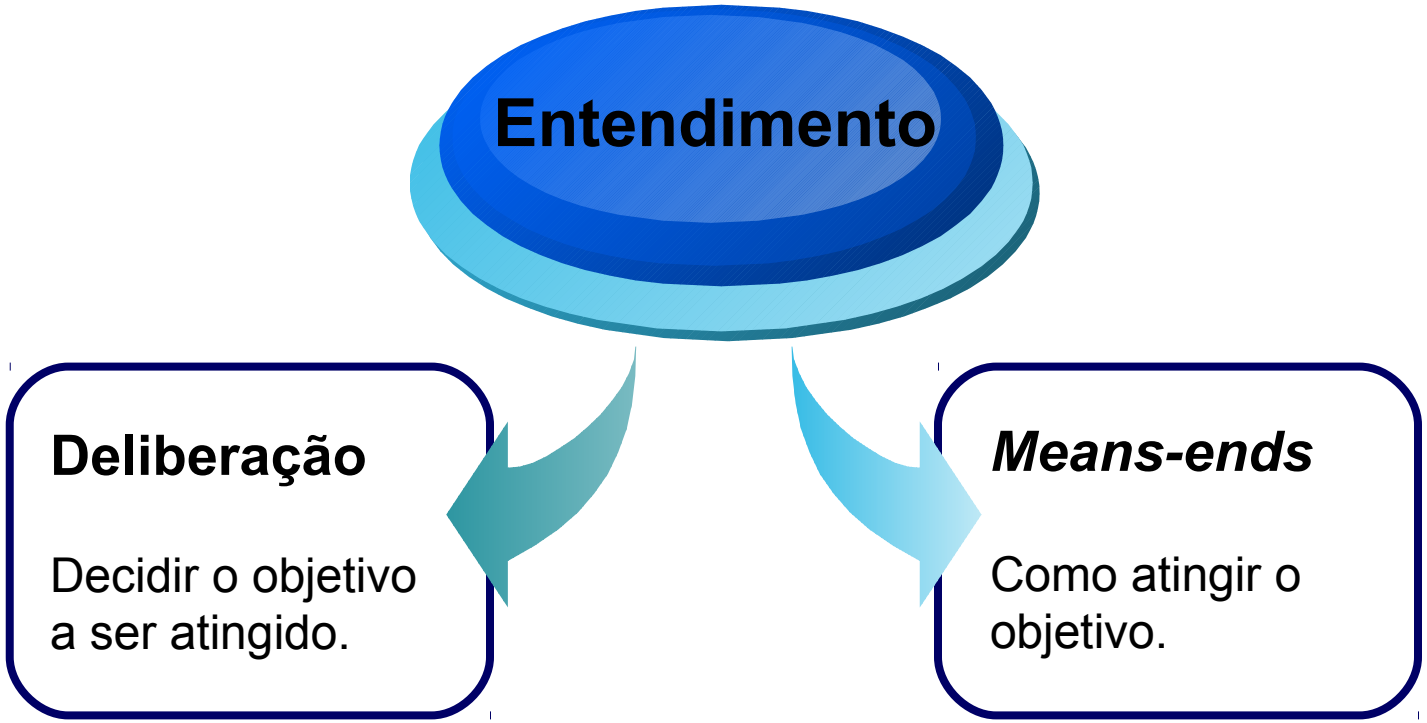
Características

- Mensuração de objetividade
- Possui uma função de utilidade
- Pode fazer balanceamento entre objetivos conflitantes
- Mapeia os estados em números reais

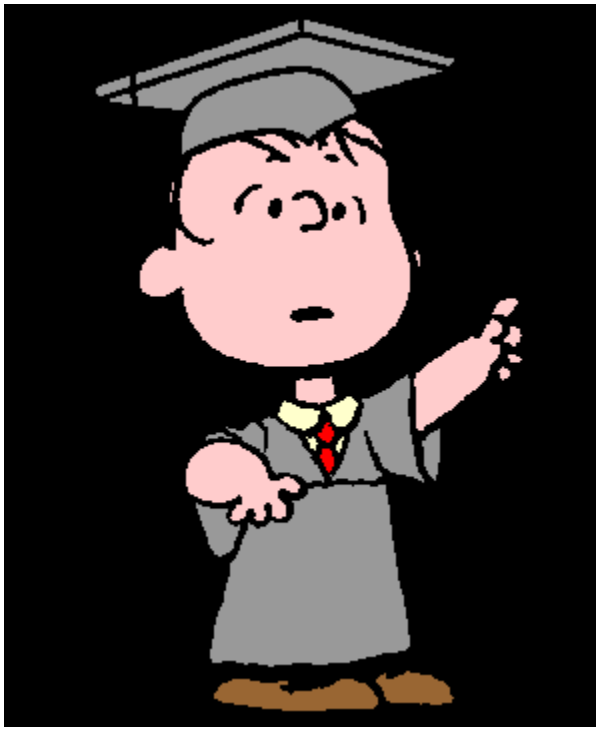
O modelo BDI

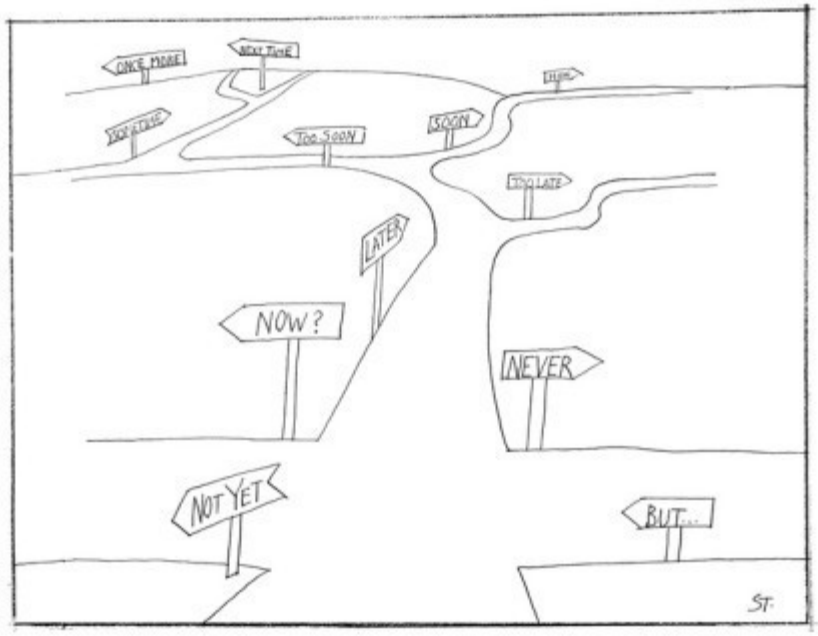
- É uma das mais importantes arquiteturas de agentes deliberativos;
- É baseada em um modelo cognitivo fundamentado em três atitudes mentais:
  - ***Beliefs – Desires– Intentions* (B-D-I)**
- Toda a fundamentação filosófica advém dos trabalhos:
  - Daniel C. **Dennett**. The Intentional Stance (1987)
  - Michael E. **Bratman**. Intentions, Plans and Practical Reason (1987)

O raciocínio prático



Raciocínio lógico (crenças)

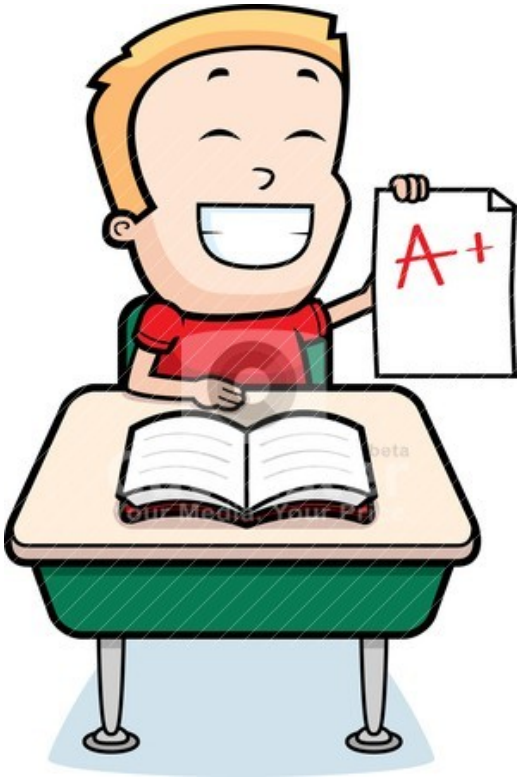




O que fazer depois de ter terminado um curso superior?
Quais opções tenho em função do conhecimento que possuo?

Esse conhecimento é a sua **CRENÇA**.

Raciocínio lógico (desejos)



Existirão grupos de alternativas (estados do mundo) que irão motivar o agente. Chamamos isso de **DESEJOS**.

Raciocínio lógico (Intenções)

- O agente optará por uma intenção e se comprometerá por ela:
  - Elas alimentarão o raciocínio lógico futuro do agente.
  - O agente deverá designar tempo e esforço para realizar a sua intenção.

Exemplo de intenção

- **Ingressar na academia.**
- Possíveis ações:
  - Inscrever-se em programas de mestrado;
  - Persistir!
  - Detectadas falhas (não aceitação em várias universidades) então seria racional aumentar as horas de estudo;
  - No entanto, persistir em muitas falhas é irracional.
- Assim, uma intenção está relacionada com crenças sobre o futuro.

Intenções e o raciocínio prático

- Intenções **guiam** como o raciocínio será atingido (means-ends)
  - "Tentar entrar em um programa de mestrado e, caso não consiga, tentar em outra universidade";
- Intenções **restringem** deliberações futuras
  - "Ações conflitantes com minha intenção não devem ser investidas, como por exemplo, ser rico e ser universitário";
- Intenções **persistem**
  - "Não se pode desistir de uma intenção sem uma boa razão para isso. Caso contrário, jamais uma intenção será atingida";
- Intenções **influenciam** crenças sobre as quais os futuros raciocínios práticos serão baseados
  - "Se a intenção é tornar-se um acadêmico, então deve-se acreditar que em breve isso será verdade. Se, simultaneamente, acredita-se que nunca será um acadêmico então o agente está sendo irracional".

Dilema de um agente BDI

Serei um agente corajoso ou cuidadoso?

Como fazer um balanceamento pró-ativo?

O dilema

- Não parar para reconsiderar suas intenções (**bold agents**):
  - Poderá gerar **trabalho inútil**, pois ele tentará atingir algo que talvez nunca possa ser atingido;
- Parar constantemente para reconsiderar suas intenções (**cautious agents**):
  - Poderá fazer com que ele **jamais atinga uma intenção** (por conta do tempo que seria insuficiente);

λ = Taxa de evolução do mundo

- λ baixo (ambiente não muda rapidamente)
  - Os agentes corajosos são mais eficientes, pois eles estarão sempre ocupados trabalhando nos seus objetivos (e atingindo suas intenções)
- λ alto (ambientes mudam frequentemente)
  - Os agentes cautelosos são normalmente melhores, pois eles são capazes de reconhecer quando uma intenção está condenada (ou novas oportunidades em situações casuais)
- **Lição**: diferentes tipos de ambientes requerem diferentes tipos de estratégias de decisão;

Commitment and effectiveness of situated agents, Kinny e Georgeff (1991)

A arquitetura BDI

Revisor de crenças:
Pode determinar um novo conjunto de crenças

Determina o conjunto de opções disponíveis para os desejos do agente

Representa o processo de deliberação

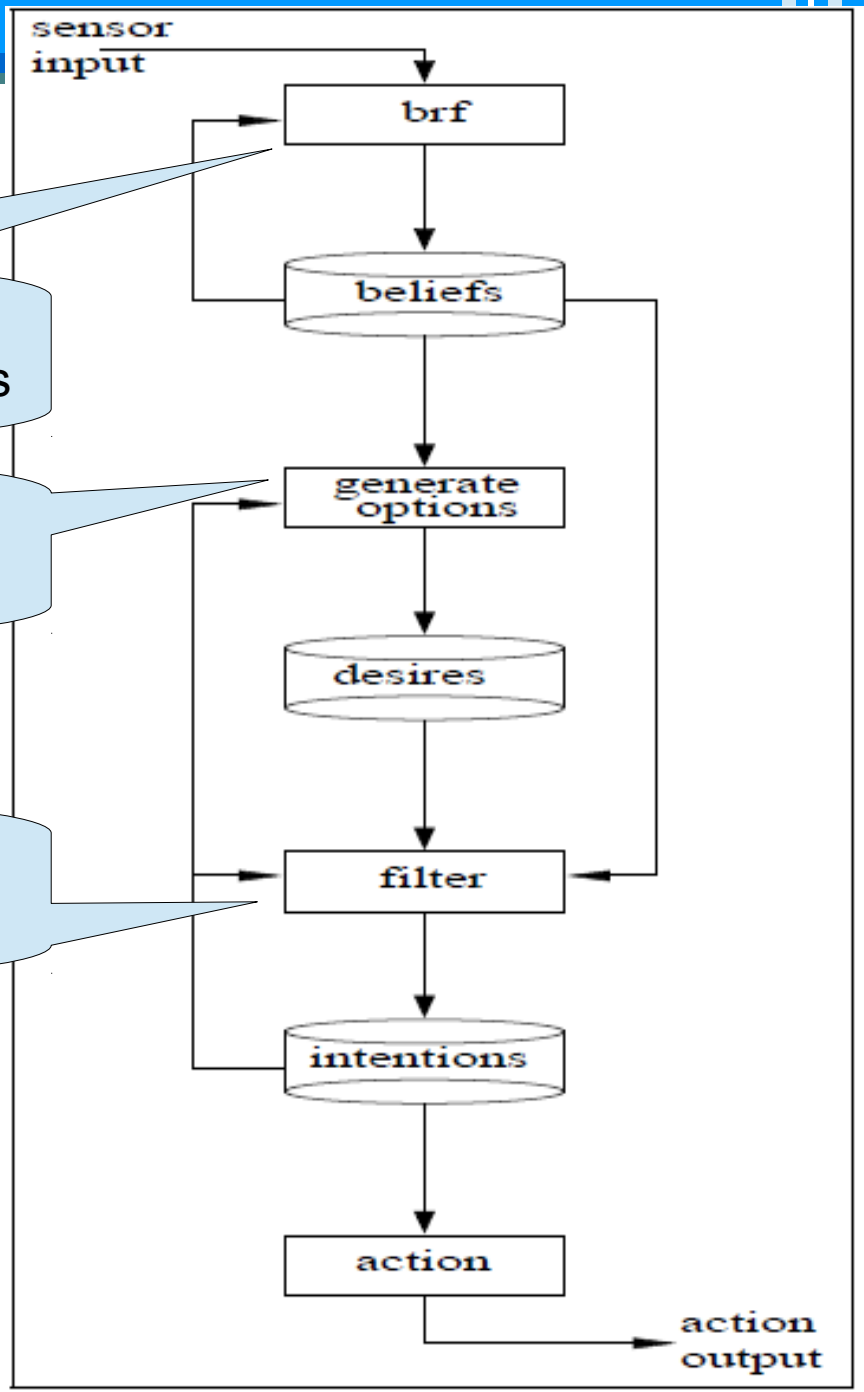


Diagrama esquemático da arquitetura BDI

Um agente BDI em pseudocódigo

1. function $action(p : P) : A$
2. begin
3. $B := brf(B, p)$
4. $D := options(D, I)$
5. $I := filter(B, D, I)$
6. return $execute(I)$
7. end function $action$

Pode-se ainda associar prioridade para cada uma das intenções, indicando grau de importância.

Plataforma de desenvolvimento de agentes

- **A Linguagem AgentSpeak**
- **Jason**

A linguagem AgentSpeak

- Proposta por Anand S. Rao (1996) no trabalho *AgentSpeak(L): BDI agents speak out in a logical computable language;*

Inspirada na arquitetura BDI (fiel à proposta filosófica)

Sintaxe **elegante**

Extensão natural do paradigma de programação lógica (como o Prolog)

Noções gerais

- Projetada para programação de agentes BDI na forma ***reactive planning systems***;

...estão permanentemente em execução

...reagem a eventos que aconteçam no ambiente

...executam planos depositados em uma biblioteca

- É constituída por:

Átomo de crença (predicado de primeira ordem)

Base de crença (coleção de átomos)

Literais de crença (átomos ou suas negações)

Noções gerais

- Existem dois tipos de **objetivos** (predicados):
  - De realização (!)
  - De teste (?)
- Na prática, indicam a execução de ***subplanos***;
- Existem **eventos ativadores** (*triggers events*) que podem iniciar a execução de um plano;
- Um **evento** pode ser de dois tipos:
  - Interno: gerado por um plano que precisa de um subobjetivo;
  - Externo: gerado por atualizações de crenças vindas do ambiente;
- Os E.A. estão relacionados com adição ('+') ou remoção ('-') de atitudes mentais;

Noções gerais

- Os **planos** são ações básicas que um agente executa no ambiente;

Exemplo de um plano

```
+concert(A,V) : likes(A)
   ← !book_tickets(A,V).

+!book_tickets(A,V) : ¬busy(phone)
   ← call(V);
      ...;
      !choose_seats(A,V).
```

Um concerto a ser realizado pelo artista A no local V corresponde a adição de uma crença concert(A,V) como conseqüência da percepção do ambiente.
Caso o agente goste do artista A então ele irá reservar ingressos para essa combinação.

Ao adotar o objetivo de reservar , se o telefone não estiver ocupado, então o agente irá executar o plano indicado pela ligação para o local do evento seguindo por um protocolo de compra "..." e finalizado pela escolha dos acentos para o concerto do artista.

Sintaxe abstrata

ag	$::=$	bs ps						
bs	$::=$	$b_1 \ldots b_n$						$(n \geq 0)$
at	$::=$	$p(t_1, \ldots, t_n)$						$(n \geq 0)$
ps	$::=$	$p_1 \ldots p_n$						$(n \geq 0)$
p	$::=$	$te : ct \leftarrow h$						
te	$::=$	$+at$	$\mid$	$-at$	$\mid$	$+g$	$\mid$	$-g$
ct	$::=$	at	$\mid$	$\neg at$	$\mid$	$ct \wedge ct$	$\mid$	T
h	$::=$	a	$\mid$	g	$\mid$	u	$\mid$	$h;h$
a	$::=$	$A(t_1, \ldots, t_n)$						$(n \geq 0)$
g	$::=$	$!at$	$\mid$	$?at$				
u	$::=$	$+at$	$\mid$	$-at$				

Fonte: Hübner, Bordini e Vieira (2004, p. 10).

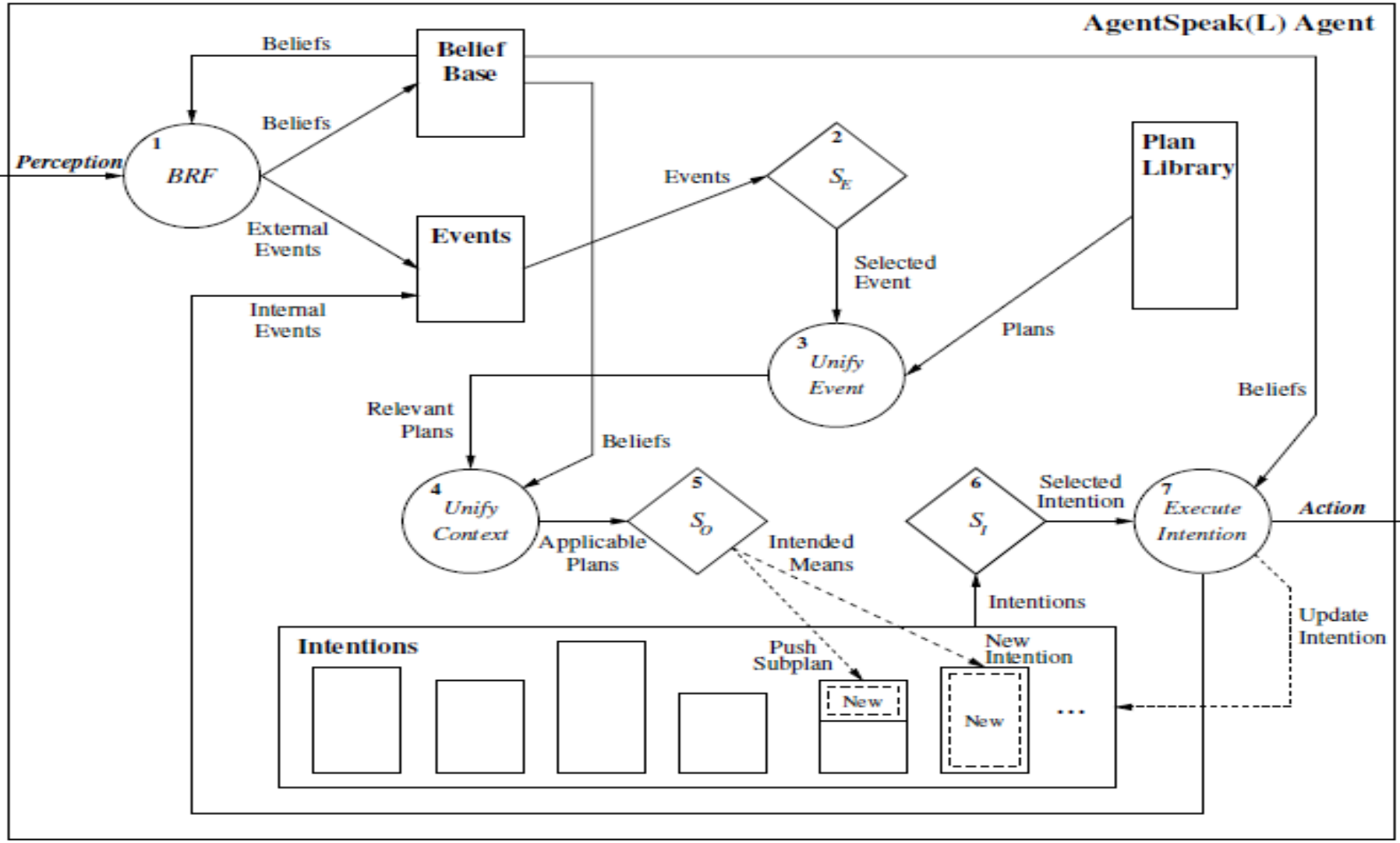
Onde:

ag: é um agente especificado por um conjunto de crenças **bs** e um conjunto de planos **ps**	**ct**: é o contexto do plano **h**: é uma seqüência de ações, objetivos ou atualizações de crenças
at: são fórmulas atômicas da linguagem, que são predicados onde **p** é um símbolo predicativo, ou seja, **p**: é um plano **te**: é o evento ativador	**te**: adição/remoção de crenças da base de crenças do agente **(+at e -at)** **te**: adição/remoção de objetivos **(+g e -g)**;
u: corresponde a atualização da base de crenças	**g**: são os objetivos e podem ser de realização **(!at)** ou de teste **(?at)**;

Um interpretador abstrato para AgentSpeak(L)

- Necessita...
  - ...ter acesso à **base de crença** e à **biblioteca de planos**;
  - ...gerenciar o conjunto de **eventos** e **intenções**;
- Seu funcionamento requer **três funções de seleção** para selecionar...
  - ...um evento;
  - ...um plano aplicável;
  - ...uma intenção.
- Devem ser **flexíveis** ao projetista.

Ciclo de raciocínio de um programa agente em AgentSpeak(L)



Jason

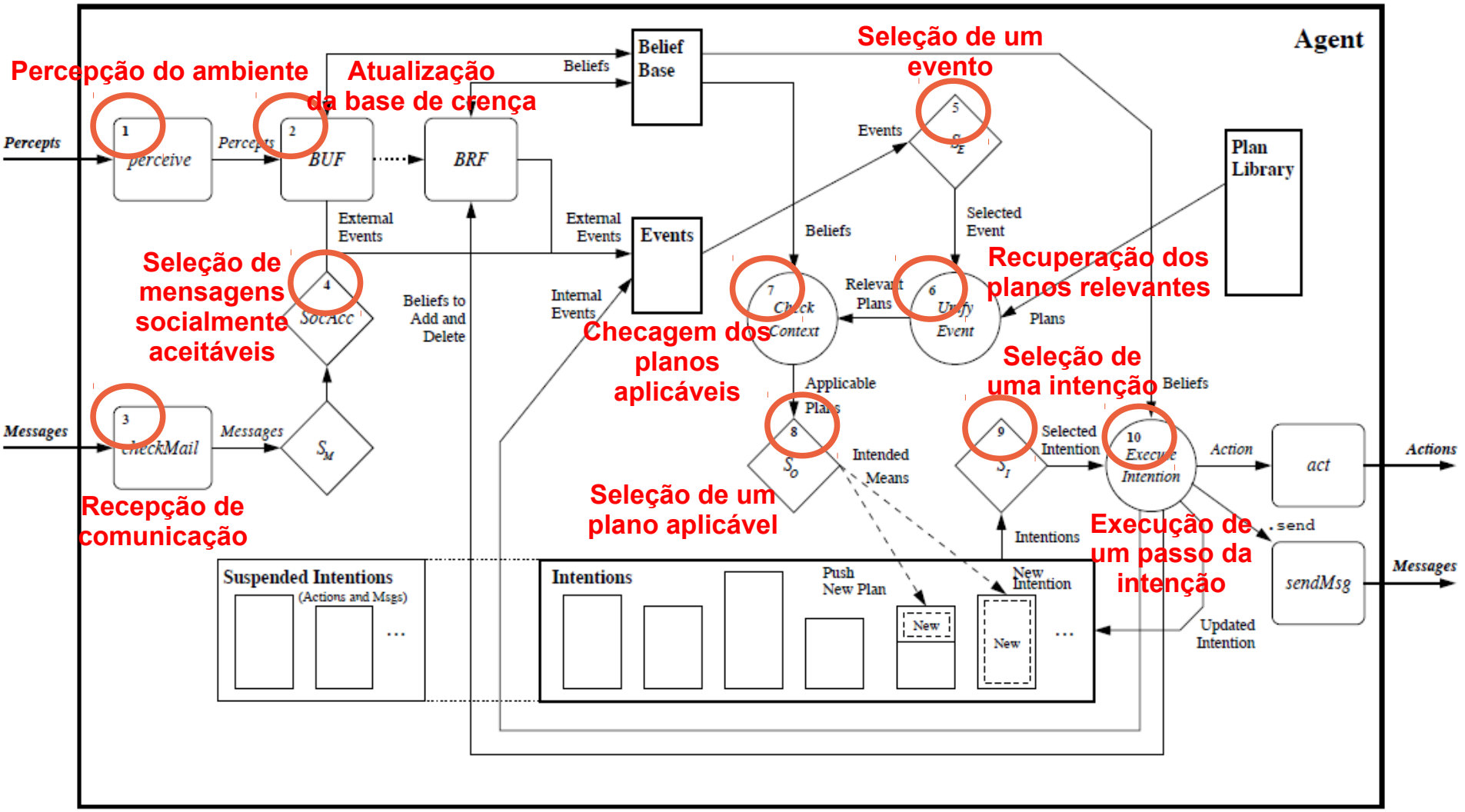
- Desenvolvido por:
  - Jomi F. Hübner e Rafael H. Bordini
    UFSC PUCRS

Baseado em diversos trabalhos anteriores, especialmente: Michael Fisher, Joyce Martins, Álvaro Moreira, Renata Vieira, Willem Visser, Mike Wooldridge

Jason ....

- ...é um **interpretador** para uma versão estendida da linguagem AgentSpeak(L);
- ...implementa uma semântica operacional da linguagem;
- ...provê uma **plataforma para desenvolvimento** de sistemas multi-agentes;
- ...possui muitos **recursos de customização** de agentes;
- ...é livre!!! \o/ => http://jason.sourceforge.net/
- ...é implementado em Java;

Ciclo de raciocínio do Jason



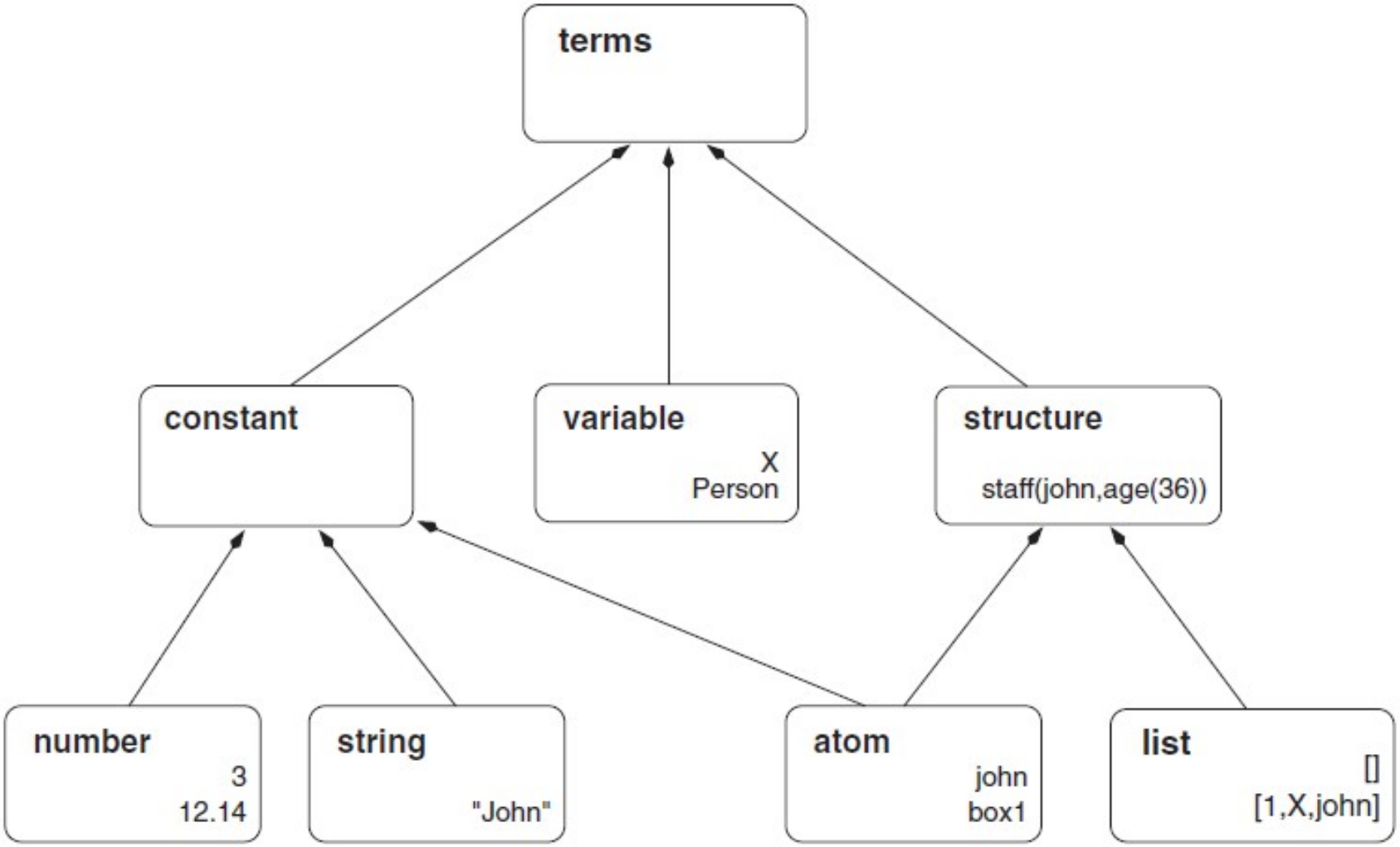
AgentSpeak + Jason

- Crenças:
  - Representada por literais de lógica de primeira ordem;
  - Descreve o que o agente **sabe sobre** o ambiente;
  - Expressa a crença do agente e não a verdade pura;
  - Exemplo de uma base de crença do agente Tom:

```
red(box1) [source(percept)].
tall (bob).
friend(bob,alice) [source(bob)].
lier(alice) [source(self),source(bob)].
~lier(bob) [source(self)].
```

Anotações

AgentSpeak + Jason: Termos Reconhecidos



AgentSpeak + Jason: Alterações na base de crença

Percepções

[source (percept)]

Automático

Intenções

[source(self)]

Exemplos:
+lier(alice);
-lier(jonh);

Comunicações

[source(bob)]

Exemplo:
.send(tom,tell,lier(alice));

AgentSpeak + Jason

Not
(negação como falha)

É true se o interpretador falhar para derivar o argumento.

Exemplo:
not likes(john, music).

~
(negação forte)

É true se o agente tem uma crença explícita que o argumento é falso.

Exemplo:
~likes(john,music).

AgentSpeak + Jason

- **Objetivos** (! = realização e ? = teste):
  - Exemplo: !write(book).
- Novos objetivos são adicionados:
  - **Por intenções**

```
!write(book);
?publisher(P);
```

  - **Por comunicações**

```
.send(tom,achieve,write(book)); // sent by Bob
// adds new goal write(book)[source(bob)] for Tom
```

AgentSpeak + Jason

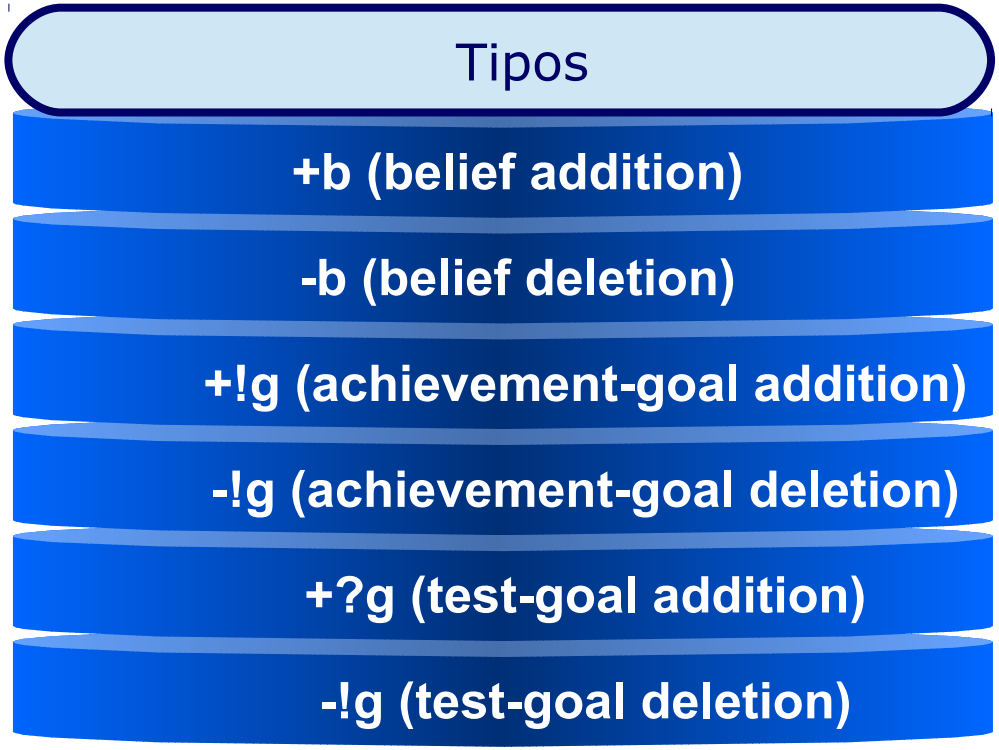
- **Por comunicações** (outros exemplos)

```
.send(tom,unachieve,write(book)); // sent by Bob
// removes goal write(book)[source(bob)] for Tom
```

```
.send(tom,askOne,published(P),Answer);
// sent by Bob
// adds new goal ?publisher(P)[source(bob)] for Tom
// the response of Tom will unify with Answer
```

AgentSpeak + Jason: Eventos

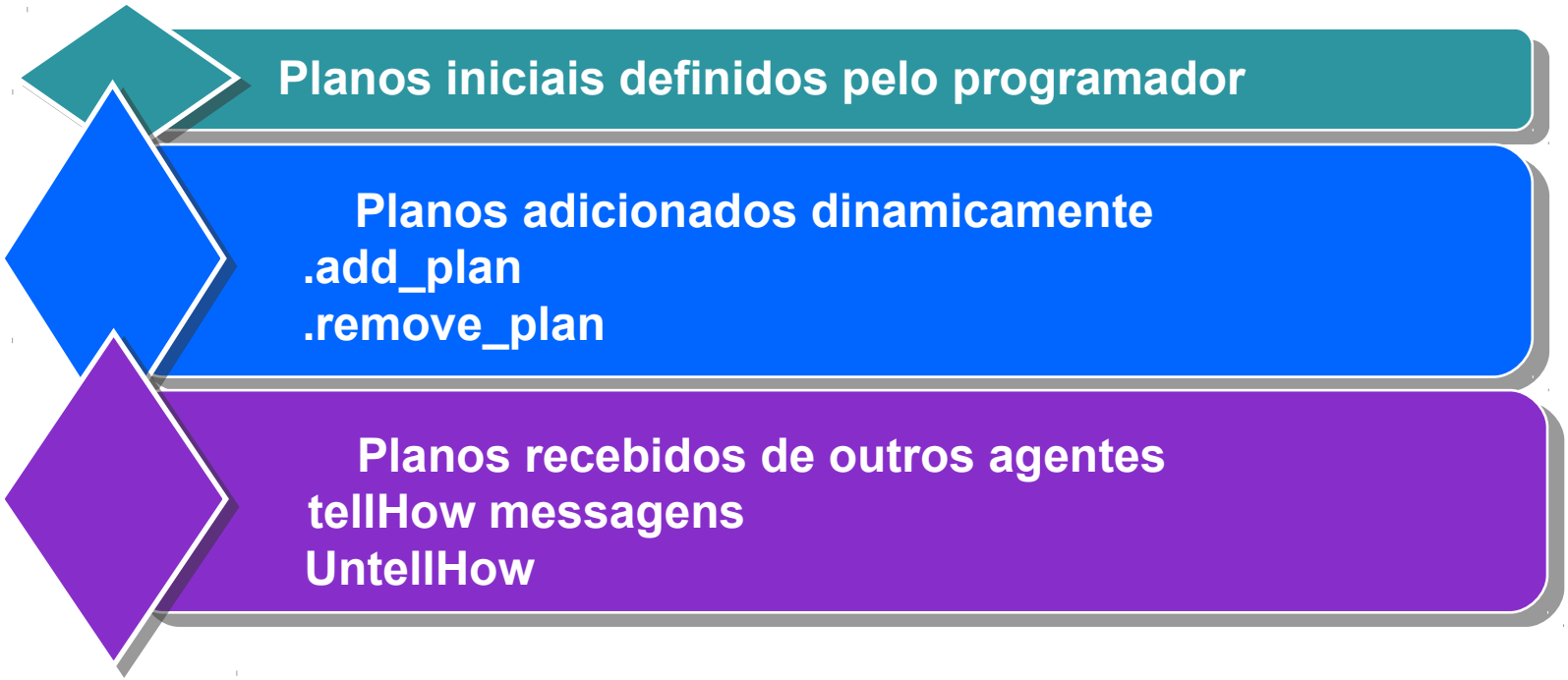
- Ocorrem como consequências das mudanças no agente (crenças ou objetivos);



- O agente reage a evento por meio da execução de planos;

AgentSpeak + Jason

- Uma **biblioteca de planos** de um agente é formada por:



- Um plano possui a seguinte estrutura:

triggering event : context <- body

AgentSpeak + Jason

- Exemplo de um plano inicialmente definido:

```
+!prepare(Something) : number_of_people(N) &
stock(Something,S) & S > N <-

... .
```

Jason

- **Ações internas** do agente:
  - Jason traz um gama de comportamento pré-programados inerentes aos agentes;
  - São identificados por um "." precedendo seu nome;
  - Exemplos:
    - .send
    - .print
    - .member
    - .list

  - É possível **criar as próprias** ações internas;
    - Ver mais detalhes em: http://www.slideshare.net/necioveras/jason-componentes-personalizados

Ambientes de agentes

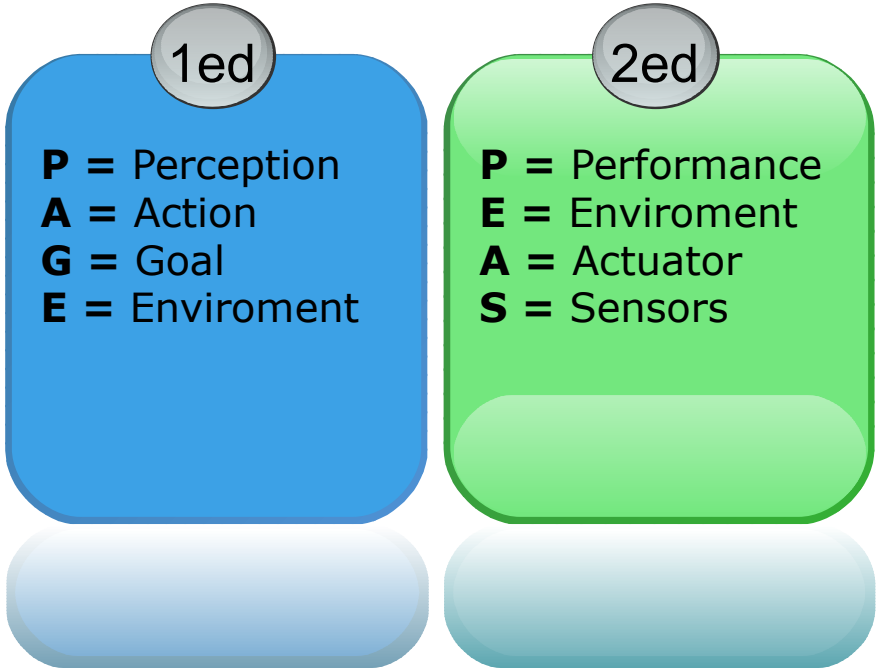
- **Conceitos**
- **Ambientes sob a perspectiva da ESOA**
- **Ambientes baseado em Artefatos**
- **Ambientes CartAgO**

Conceitos

- A noção de ambientes é um conceito primário em agentes e SMA, sendo o **lugar computacional** ou **físico** que os agentes serão situados;
  - Este lugar irá prover e definir as noções de **percepções**, **ações** e **interações** dos agentes;
- Os recursos fundamentais da **abstração de agentes estão** direta ou indiretamente ligados com o ambiente.

Ambiente de agente

- Basicamente um ambiente é o **problema** que o agente deverá ser a **solução**;
- Um ambiente de tarefas é definido por Russel e Norvig da seguinte forma:



Exemplo PEAS

Tipo de Agente	Medidas de Performance (Performance Measure)	Ambiente (Environment)	Atuadores (Actuators)	Sensores (Sensors)
Sistema de diagnóstico médico	Saúde do paciente, minimizar custos, causas	Paciente, hospital, equipe médica	Mostrar questões, testes, diagnósticos, tratamentos, orientação	Entrada dos sintomas por teclado, respostas do paciente, pesquisa
Sistema de análise de imagem de satélite	Categorização correta da imagem	Downlink a partir do satélite em órbita	Mostrar categorização da cena	Arrays de pixels de cores
Robô coletor de partes	Percentual de partes na caixa correta	Esteira rolante com partes; caixas	Braço articulado e mão	Câmera, sensores de ângulo articulado
Controlador de refinaria	Maximizar pureza, produção, segurança	Refinaria, operadores	Válvulas, bombas de água, aquecedores, telas	Temperatura, pressão, sensores químicos
Tutor de inglês interativo	Maximizar a pontuação de estudantes em testes	Conjunto de estudantes, agência de testes	Mostrar exercícios, sugestões, correções	Entrada pelo teclado

[Russel e Norvig 2009]

Propriedades dos ambientes

Totalmente observável

- Os sensores detectam todos os aspectos que são **relevantes** para a escolha da ação;
- Os agente não precisa manter nenhum estado interno para controlar o mundo;

Parcialmente observável

- Ou os sensores são imprecisos e possuem ruídos;
- Ou os estados estão ausentes nos dados dos sensores;

Propriedades dos ambientes

Propriedades dos ambientes

Estocástico

- Quando a completa determinação do próximo estado não for possível;

Determinístico

•O próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada;

Episódico

• Experiência do agente divida em episódios atômicos que geram um única ação independente;

Observações durante as mudanças

Sequencial

• Uma ação afeta todas as ações futuras;
• O agente precisa "pensar" à frente;

Propriedades dos ambientes

Dinâmico

- Quando um ambiente pode se alterar enquanto um agente estiver deliberando então esse ambiente é dinâmico para esse agente;

Semidinâmico

- É quando o ambiente não muda com o tempo, mas o agente sobre alterações no seu desempenho;

Estático

- São mais simples, pois o ambiente não muda durante as deliberações do agente;

Propriedades dos ambientes

O ambiente mais hostil

- Parcialmente observável
- Estocástico
- Sequencial
- Dinâmico
- Contínuo
- Multiagente

"As situações reais são tão complexas que para finalidades práticas devem ser tratadas como estocásticas".

Abordagens sobre o ambiente

Há duas principais perspectivas para definição de ambiente, oriundas...

...das raízes clássicas da I.A.

Existe um mundo externo que é percebido e atuado através dos agentes de modo a cumprir com suas tarefas.

...da Engenharia de Software Orientada a Agentes

Trabalhos recentes introduziram a ideia de abstração de primeira classe.

Ambientes sob o contexto da ESOA

- **Abstração de primeira classe**

Existe um lugar adequado para encapsular funcionalidades e serviços que suportam atividades de agentes

Trabalhos sobre o assunto:

1. Weyns, D., & Parunak, H. V. D. (Eds.). (2007). **Special issue on environments for multi-agent systems**. Journal of Autonomous Agents and Multi-Agent Systems, 14(1), 1–116

2. Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., & Ferber, J. (2005). **Environments for multiagent systems: State-of-the-art and research challenges**. In D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, & J. Ferber (Eds.), Environment for multi-agent systems, volume 3374 (pp. 1–47). Berlin/Heidelberg: Springer.

Ambientes sob o contexto da ESOA

- Nesta visão o ambiente não é mais apenas o alvo das ações do agente, nem um container ou gerador de percepções para o agente, mas agora ele é parte do SMA e pode ser desenhado para melhorar o desenvolvimento global do sistema;
- O ambiente pode ser definido sob o ponto de vista da engenharia de um SMA como **endógeno**, fazendo parte do sistema para o qual foi desenhado;
- A contrário, a visão clássica da IA, pode ser definida como **exógena**;

Programando Ambiente em SMA

- A ideia básica da programação de ambiente pode ser resumida como:

  prog. de SMA = prog. de agente + prog. de ambiente

- Onde implicitamente será referenciado softwares de SMA e **ambientes endógenos**;
- O ambiente é uma parte programável do sistema, **ortogonal** em relação à parte do agente;
  - Ortogonalidade significa separação de interesses com duas possibilidades;

Separação de interesses

1

Os **agentes são abstrações básicas** para projetar e programar partes autônomas do sistema.

São projetados para realizar algum objetivo, tanto individual ou coletivo, encapsulando a lógica e o controle de suas ações.

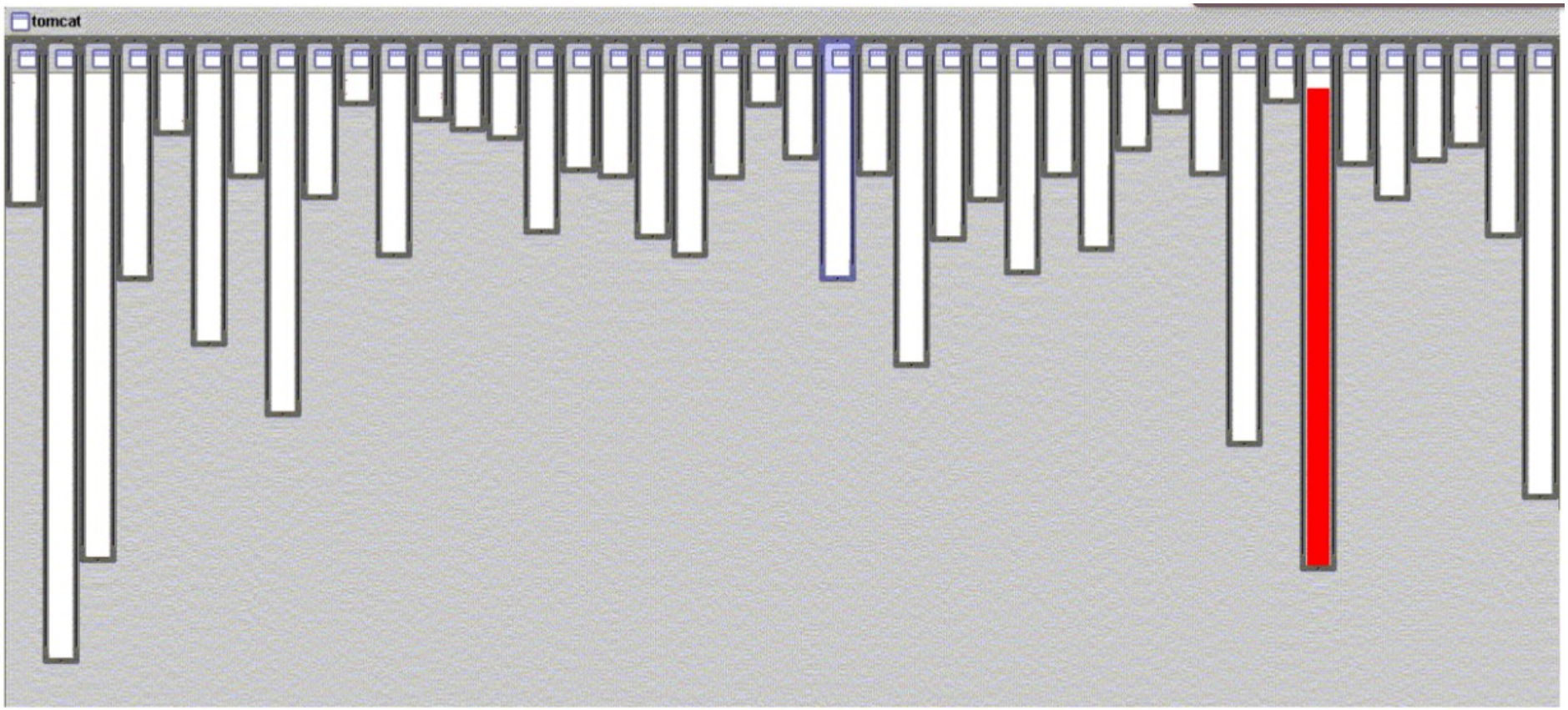
2

O **ambiente pode ser usado para projetar e programar a parte computacional** do sistema que é funcional para o trabalho dos agentes.

Agentes podem dinamicamente acessar e usar algumas tipos de funcionalidades para explorar, possibilitando adaptar-se para melhor atender suas atuais necessidades.

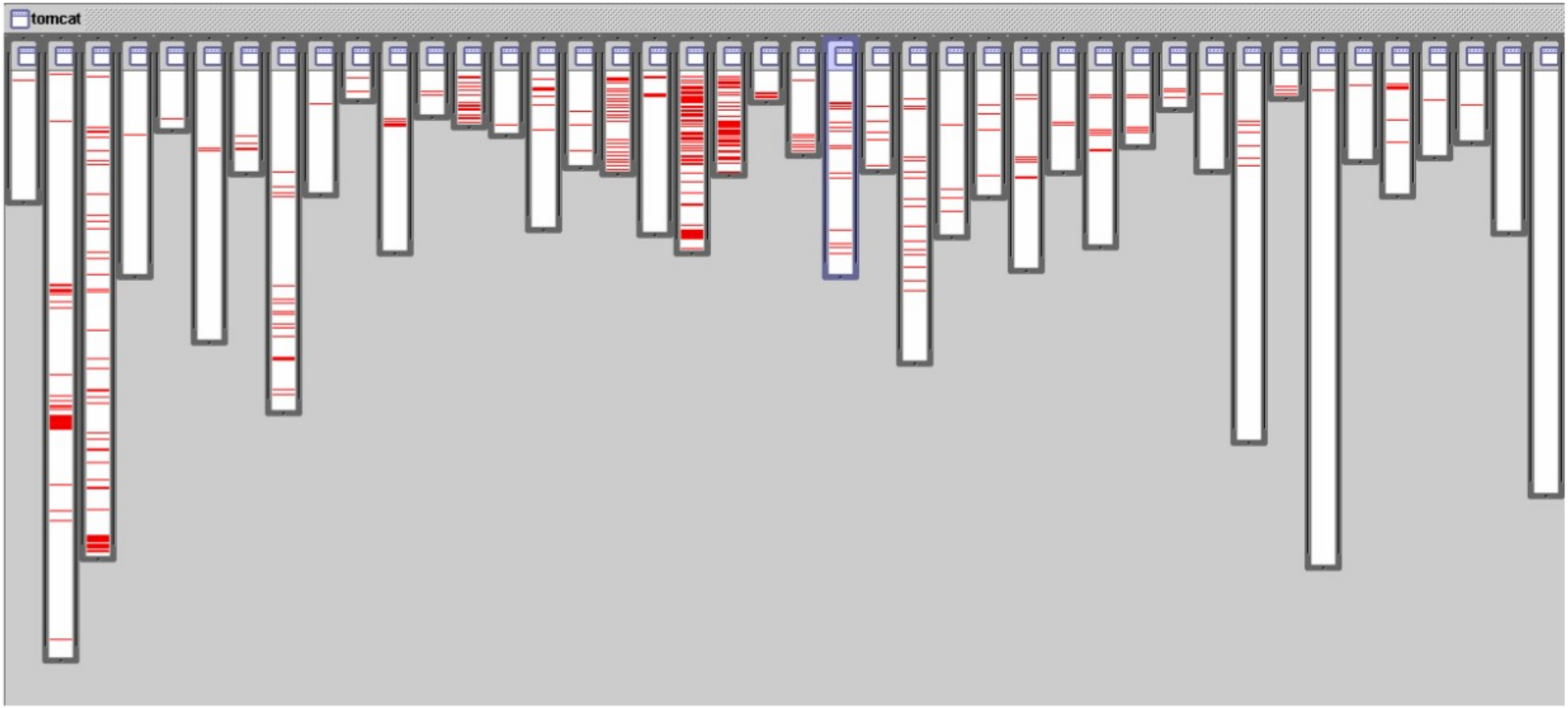
Exemplo de Ortogonalidade

- Uso do **XML Parse** do apache Tomcat

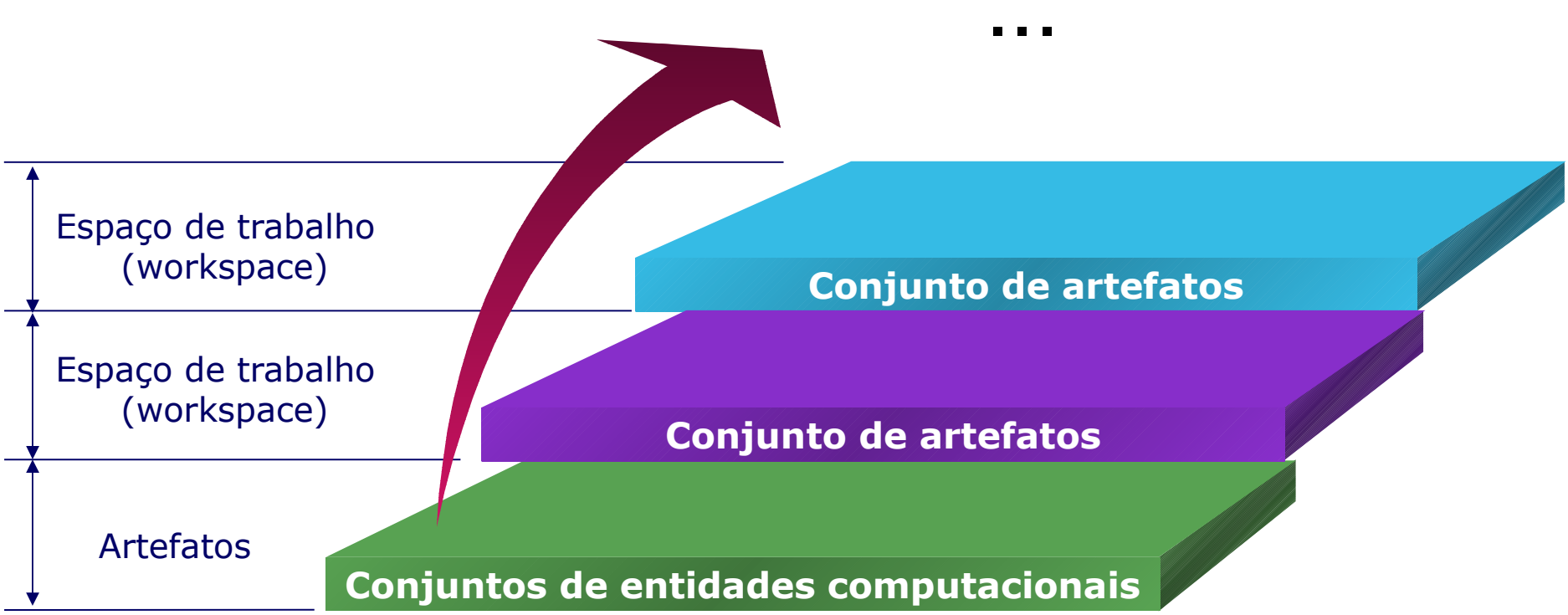


Exemplo de Ortogonalidade

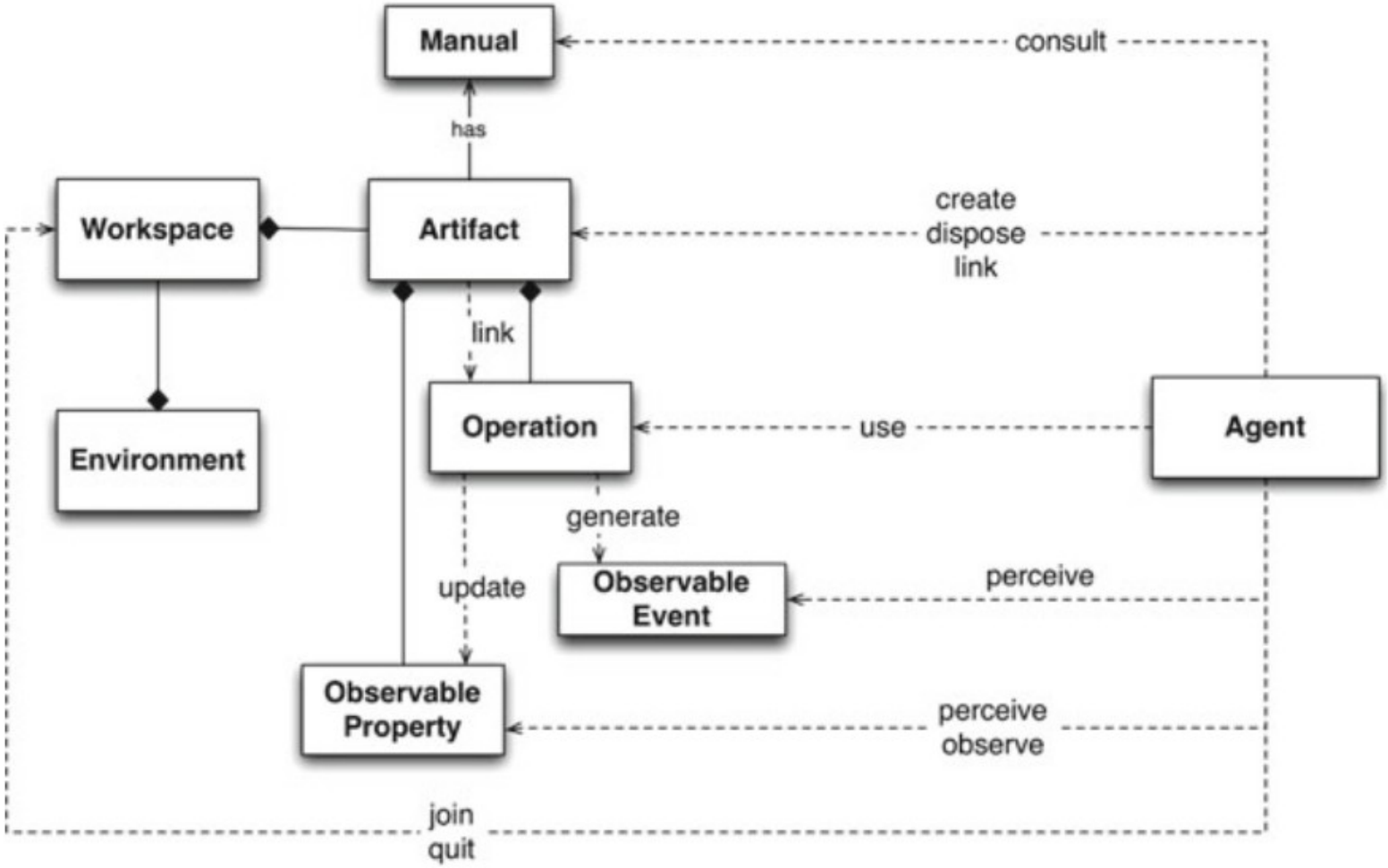
- Uso de **Logging** do apache Tomcat



Ambientes baseados em artefatos



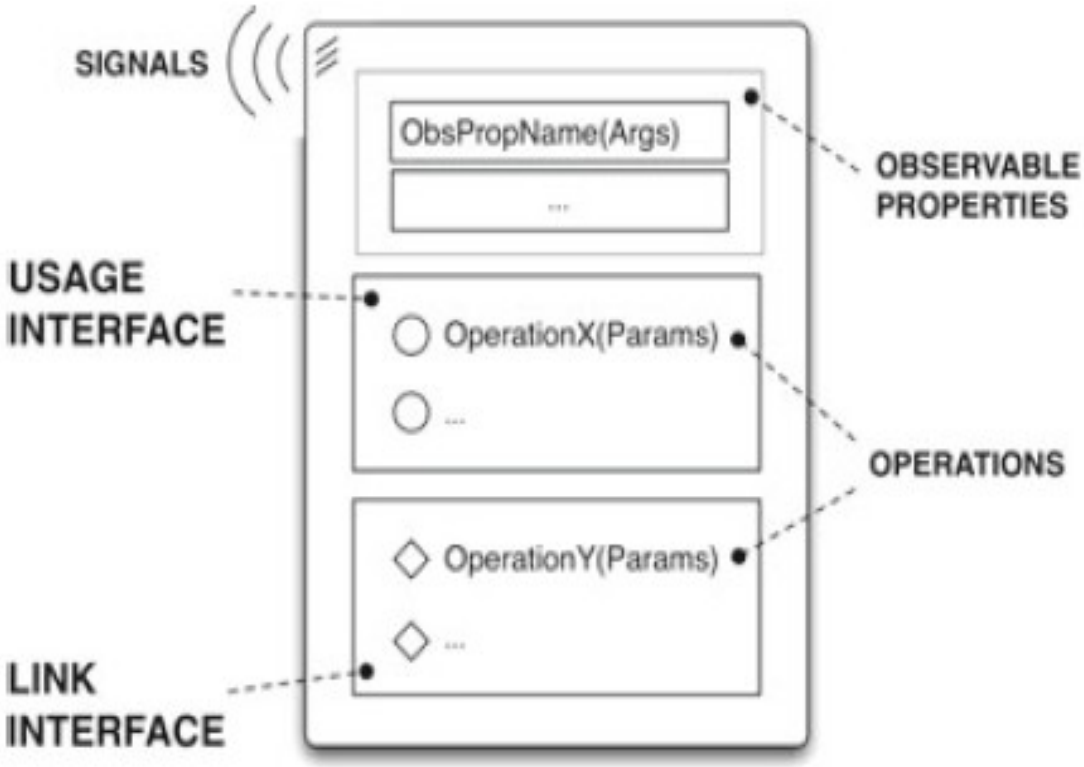
Ambientes baseados em artefatos



Uma visão geral dos principais conceitos que caracterizam ambientes baseados em artefato

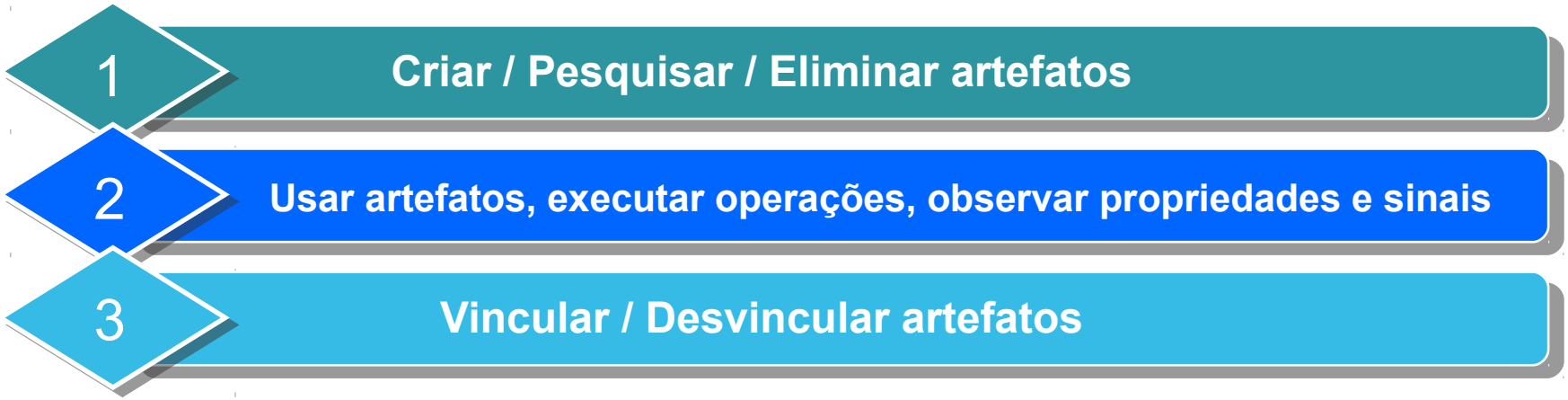
Ambientes baseados em artefatos

- Programadores de SMA definem os tipos de artefatos de forma análoga à POO, pois é definido **estrutura** e **comportamento**;



Ambientes baseados em artefatos

- Ações para trabalhar com artefatos
  - Um agente precisa **juntar-se a um workspace** para trabalhar com ele (ou vários ao mesmo tempo);
  - Também deve **sair o mais rápido possível** quando terminar o trabalho;
- Dentro de um workspace o agente **pode realizar ações** para trabalhar com artefatos categorizados em três grupos:



Ambientes baseados em artefatos

1 **Criar / Pesquisar / Eliminar artefatos**

- Os três tipos básicos são:
  - makeArtifact(ArName,ArTypeName,InitParams):ArId
  - disposeArtifact(ArId)
  - lookupArtifact(ArName):ArId

2 **Usar artefatos, executar operações, observar propriedades e sinais**

3 **Vincular / Desvincular artefatos**

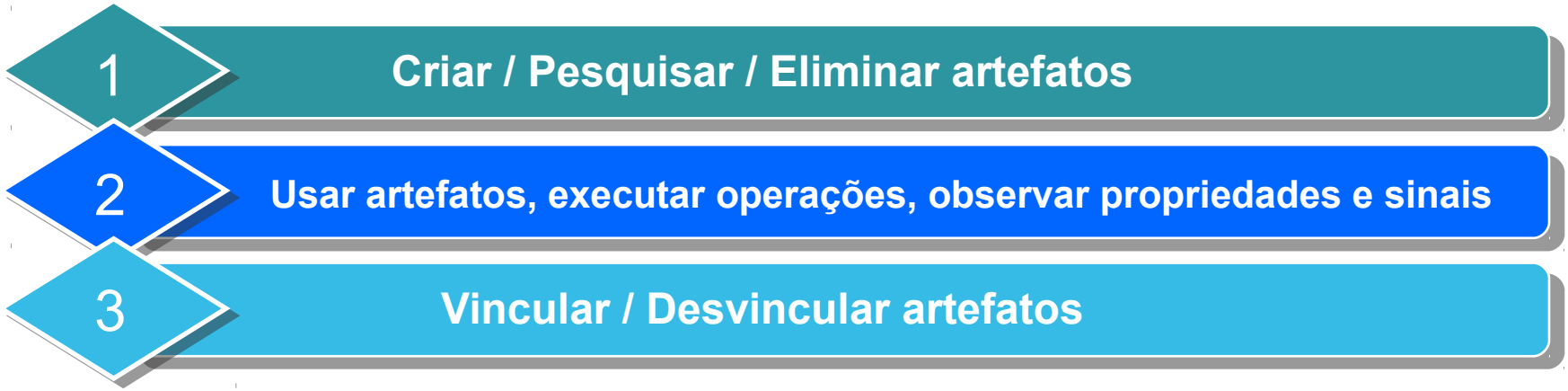
Ambientes baseados em artefatos

1. Criar / Pesquisar / Eliminar artefatos
2. Usar artefatos, executar operações, observar propriedades e sinais

- Envolve dois aspectos:
  - Ser **capaz de executar operações** listadas em ***usage interface*** do artefato;
  - Ser **capaz de perceber informações** observáveis do artefato em termos de propriedades e eventos observáveis;
- **focus(ArId,{Filter}) (ou stopFocus(ArId))**

3. Vincular / Desvincular artefatos

Ambientes baseados em artefatos



- Para isso existem duas ações básicas:
  - **linkArtifacts(LinkingArId,LinkedArId{,Port})**
  - **unlinkArtifacts(LinkingArId,LinkedArId)**

CArtAgO

(Common ARtifact infrastructure for AGent Open environments)

- É um **framework** e **infraestrutura** para **programação** e **execução** de ambientes baseados em artefatos;

Fornece enquanto framework:

1. Uma API em Java para programar artefatos e ambientes em tempo de execução para executar ambientes baseados em artefatos;
2. Uma biblioteca com um conjunto de tipos de artefatos de uso geral pré-definidos.

Provê enquanto infraestrutura:

Provê uma API e um mecanismo subjacente para estender Linguagens ou framework de programação de agentes assim como para que programa de agentes trabalhem dentro de ambientes Cartago.

CArtAgO

(Common ARtifact infrastructure for AGent Open environments)

- Cartago é **ortogonal** para a tecnologia específica adotada pela programação de agentes trabalhando em ambientes baseados em artefatos;
- Foi concebido **de modo a ser integrado com qualquer linguagem** de programação e plataforma de agentes, permitindo a criação sistemas heterogêneos;
- Um exemplo ilustrativo de **Cartago** com **Jason….**

Estudo de caso com Jason e Cartago

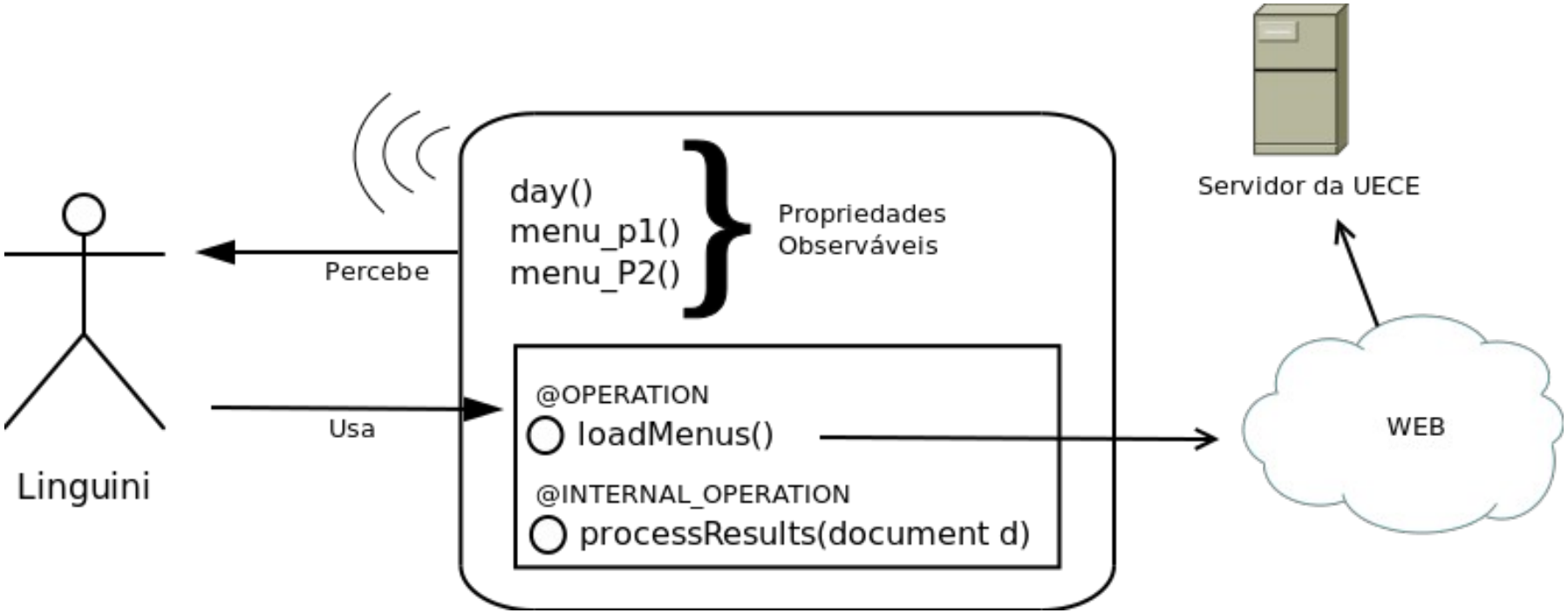
- **Estudo de caso I: Um agente monitor de cardápio para um restaurante universitário**
- **Estudo de caso II: Um simulador de um jogo de bingo**

Estudo de Caso I : Linguini

- Ambiente **computacional**;
- Propriedades:
  - Parcialmente observável;
  - Episódico;
  - Determinístico;
  - Contínuo;
  - Estático (na maioria do tempo);

- Os dados são providos por meio da WEB;
- **Problema**: monitorar o cardápio do RU da UECE e divulgar em uma rede social;

Arquitetura informal



Ambiente

```
1  package workspaces;
2
3  import java.io.BufferedReader;
15
16 public class EnvironmentWeb extends Artifact {  (1)
17   void init() {
18     defineObsProperty("day", "");
19     defineObsProperty("menu_p1", "");
20     defineObsProperty("menu_p2", "");
21   }
22
24   void loadMenus(){  (2)
62
63   @INTERNAL_OPERATION
64   void processResults(Document d){  (3)
65     for (Element e : d.getElementsByClass("ru_head_title")){
66       getObsProperty("day").updateValue(e.children().get(0).text());
67
68       if (e.nextElementSibling().children().get(0).text().length() > 139){
69         getObsProperty("menu_p2").updateValue(e.nextElementSibling().children().get(0).text().substring(139));
70         getObsProperty("menu_p1").updateValue(e.nextElementSibling().children().get(0).text().substring(0, 139));
71       }
72       else{
73         getObsProperty("menu_p1").updateValue(e.nextElementSibling().children().get(0).text().substring(0));
74         getObsProperty("menu_p2").updateValue("");
75       }
76
77       await_time(10);
78     }
79   }
80 }
```

Propriedades Observáveis

Atualizações do ambiente

O Agente

```
/* Initial beliefs and rules */
day("").
menu_p1("").
menu_p2("").

/* Initial goals */
!create.

/* Plans */
+!create: true <-
  ?setupArtifact(ID).

+?setupArtifact(E) : true <-
  makeArtifact("env_web", "workspaces.EnvironmentWeb", [], E);
  focus(E);
  !notification.

-?setupArtifact(E) : true <-
  .wait(30);
  !create.

+!notification: true <-
  loadMenus;
  !monitoring.

+!monitoring: true <-
  println("Esperando: ",(1000 * 60)/60000, " minutos");
  .wait(((1000 * 60) * 60 ) * 24);  //24 horas segundos
  !notification.
```

(a)

(b)

```
//Perceptions
+menu_p1(MenuP1): day(Day) & menu_p2(MenuP2)<-
  if (Day \== ""){
    internalActions.sendTwitter(Day);
    internalActions.sendTwitter(MenuP1);
    if (MenuP2 \== ""){
      internalActions.sendTwitter(MenuP2);
    };
  }.
```

(c)

Crenças iniciais

Objetivo inicial

Percepção

Ação interna

Planos

Considerações sobre o estudo

- Exemplo simples de demonstração de uso do Cartago com Jason;
- Recursos:
  - Comunicação Cartago <=> Web;
  - Integração agente <=> ambiente;
  - Ação interna de um agente (Java);
- Disponível em:

  http://www.github.com/necioveras/linguini
- Twitter: @linguini_uece

Estudo de caso II: Bingo

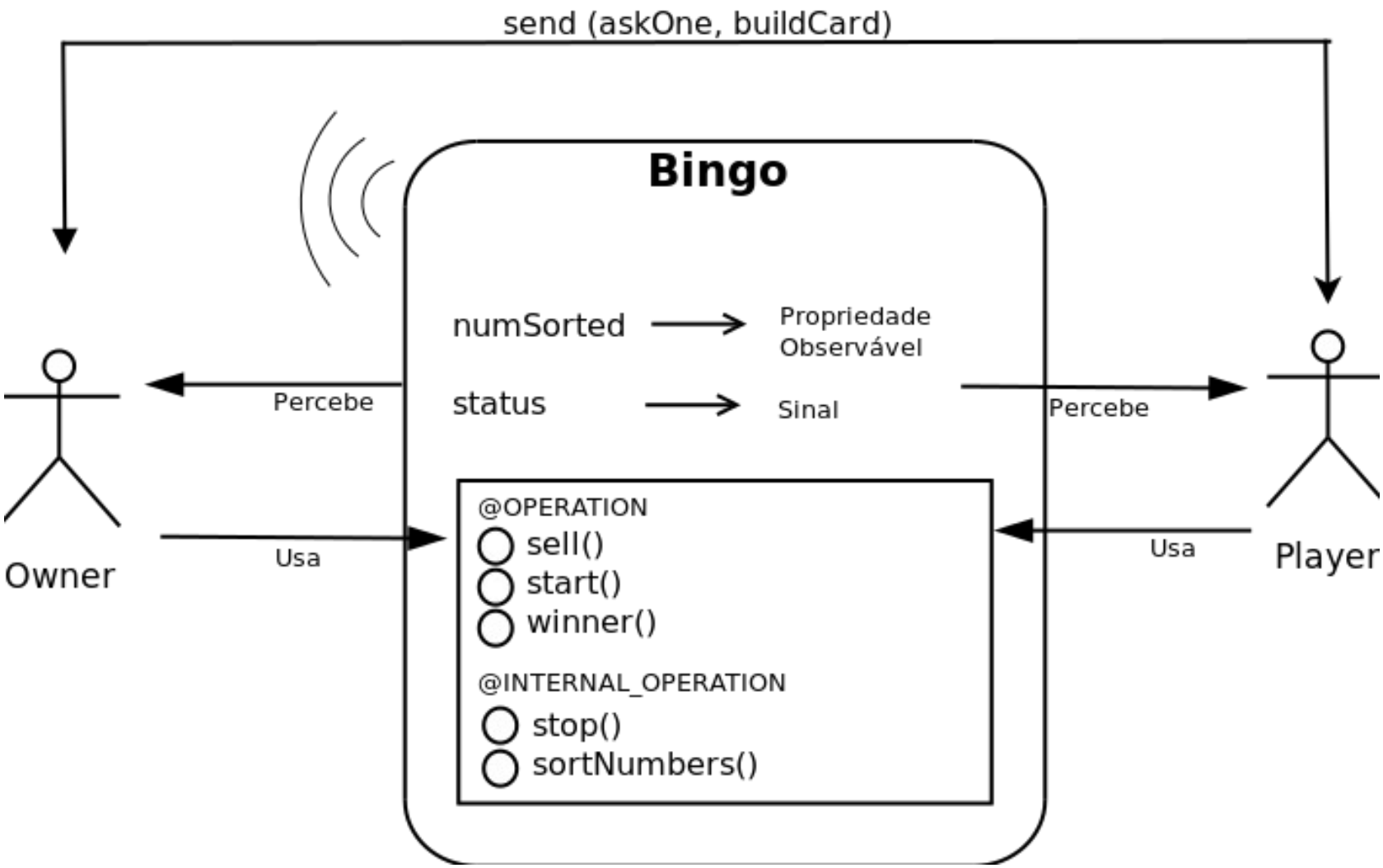
- O ambiente representa uma mesa de bingo do mundo real;
- Propriedades:
  - Parcialmente observável;
  - Sequencial;
  - Estocástico;
  - Contínuo;
  - Dinâmico;
- Existem dois agentes: Owner e Player

Estudo de caso II: Bingo

- Cria o artefato e o monitora a fim de perceber se um certo número de cartelas foram vendidas;
- Distribui cartelas;
- Delibera o início do sorteio;

- Fica “vagando” até encontrar o artefato “bingo”.
- Realiza uma operação de “compra” para participar do sorteio por meio da solicitação de uma cartela ao agente Owner.
- Após isso sua função é monitorar o ambiente esperando por números sorteados (informados por meio de sinais) e pelo final do sorteio.

Arquitetura informal



Ambiente

```
1  package tablet;
2  import java.util.HashSet;..
9  public class Bingo extends Artifact {
10
11   private Set <Integer> box = new HashSet <Integer>();
12
13   String internalStatus = "void";
14   int MAXNumberSorted = 40;
15   int MAXSold = 02;
16   int sold = 0;
17
18   void init() {
19     defineObsProperty("numSorted", 0);
20     signal("status", "selling");
21     internalStatus = "selling";
22   }
23
24   @OPERATION
25   void sell(){
26     sold++;
27     if (sold >= MAXSold){
28       signal("status", "ready");
29     }
30   }
31
32   @OPERATION
33   void start(){
34     signal("status", "started");
35     internalStatus = "started";
36     execInternalOp("sortNumbers");
37   }
```

```
39   @INTERNAL_OPERATION
40   void stop(){
41     internalStatus = "stop";
42     signal("status", "stoped");
43   }
44
45   @OPERATION
46   void winner(){
47     internalStatus = "stop";
48     signal("status", "stoped");
49   }
50
51   @INTERNAL_OPERATION
52   void sortNumbers(){
53     Random r = new Random();
54     int counter = 0;
55     while (!internalStatus.equals("stop")){
56       int x = r.nextInt(100);
57       if (box.add(x)){
58         counter++;
59         getObsProperty("numSorted").updateValue(x);
60         signal("status", "sorted");
61         await_time(3000);
62       }
63       if (counter >= MAXNumberSorted)
64         execInternalOp("stop");
65     }
66   }
67 }
```

Owner

Crenças

Percepção

Troca de mensagens

```
1  /* Initial beliefs and rules */
2
3  sizeCard(math.random(9)+1).
4  maxNumberToSort(math.random(99)+1).
5
6  /* Initial goals */
7
8  !create.
9
10 /* Plans */
11
12 +!create : true <-
14
15 +?setupBingo (C) : true <-
18
19 -?setuBingo(C) : true <-
22
23 //Signal status
24 +status(S) :  S == "ready" <-
25   start.
26
27 +!kqml_received(Sender, askOne, buildCard, Response) :
28   sizeCard(Size)  & maxNumberToSort(Max)
29   <-
30   ias.buildCard(Size, Max, Card);
31   .send(Sender, tell, Card, Response).
```

Player (planos)

Envio de mensagem

Atualização de Crença

```
1  /* Initial goals */
2
3  !participate.
4
5  /* Plans */
6
7  +!participate: true<-
8     ?myArtifact (ID);
9     focus(ID);
10    .send(owner,askOne, buildCard, Card);
11    +myCard(Card);
12    sell;  //buy   - informa que adquiriu uma cartela
13    println("Estou no bingo! Registrei minha cartela.").
14
15 +?myArtifact(C) : true <-
16    lookupArtifact("b0", C).
17
18 -?myArtifact(Art) : true <-
19    .wait(1000);
20    println("Esperando por um bingo.");
21    !participate.
```

Player (percepções)

```
23 //Perceptions of the signals
24
25 +myCard(Card) <-
26    .concat("Cartela:", Card,S);
27    println(S);
28    Hit = 0;
29    +myHits(Hit).                    //adiciona uma nova crença com o total de acertos
30
31 +status(S) : S == "sorted" & myCard(Card) & myHits(Hit) <-
32    ?numSorted(V);
33    println("Opa, percebi um numero sorteado ... ", V);
34    if (.member(V, Card) ) {
35       +myHits(Hit+1);
36       println("acertei:", V, " Ate agora acertei ", Hit+1, " numero(s) em um total de ", .length(Card));
37    }.
38
39 +myHits(Hit) : myCard(Card) <-
40    if (Hit == .length(Card)){
41       print("Gaaaaaaaaaaaaaaaaaaannnnnnnnnnnnhhhhhhhhhhheeeeeeeeiiiiiii!!");
42       winner;
43    }.
44
45
46 +status(S) :  S == "started" <-
47    println("Legal! Bingo inciado!").
48
49 +status(S) :  S == "stoped" <-
50    println("Ahhhh .. já acabou.").
51
52 //Percepctions of the Observable Properties
53 +numSorted(V).
```

Considerações sobre o estudo

- Exemplo mais apurado de demonstração de uso do Cartago com Jason;
- Recursos:
  - Definição explícita de crenças iniciais;
  - Multi-agentes;
  - Integração agentes <=> ambiente;
  - Comunicação entre agentes;
  - Ação interna de um agente (Java);
- Disponível em:
  - http://www.github.com/necioveras/Bingo_MAS_JaCa

Introdução aos Sistemas Multi-Agentes

- **Sociedades e Organizações**
- **Sistemas Multi-Agentes Normativos**
- **Modelo Organizacional MOISE+**

Sociedades e Organizações

- Organizações são uma poderosa ferramenta para construir sistemas complexos onde os agentes computacionais podem, autonomamente, desenvolver suas atividades **manifestando atitudes sociais**;
- A dimensão da organização é concebida em termos de **funcionalidades** e explorada pelos agentes;
  - Enquanto isso é mantido o controle das atividades sociais, monitorando e alterando essas funcionalidades em tempo de execução;
- Elas são concebidas em termos de **organizações humanas** (com normas, objetivos globais e papeis);
- Essa perspectiva assume uma camada organizacional que visa:
  - Promover uma **coordenação** desejada;
  - Melhorar o **controle**; e
  - Equilibrar as **dinâmicas sociais**.

Sociedades e Organizações

- É previsto um SMA integrado em termos de **sociedade de agentes, entidades ambientais e organizacionais**;
- Em particular pode-se referenciar o **Jason** como framework para desenvolvimento de agentes, **Cartago** para ambientes e **Moise** para organizações;

SMA Normativos

- As interações sociais são motivos de preocupação e por isso há trabalhos que usam **objetos normativos** como entidades reativas inspecionável por agentes atuando em "lugares normativos";
- Eles devem indicar **obrigações**, **proibições** e **direitos**;
- Os SMA Normativos são organizados por meio de **normas** que auxiliam na manutenção da ordem social e no comportamento do agente perante a sua sociedade;

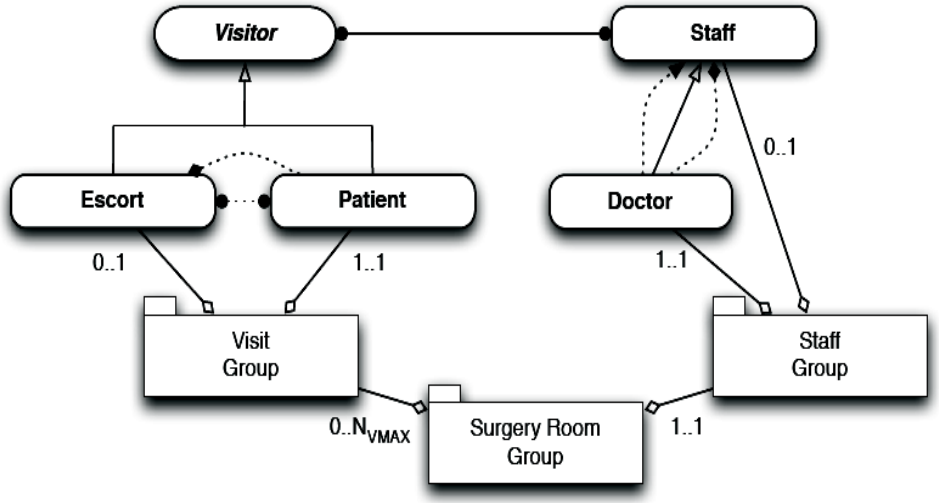
SMA Normativos

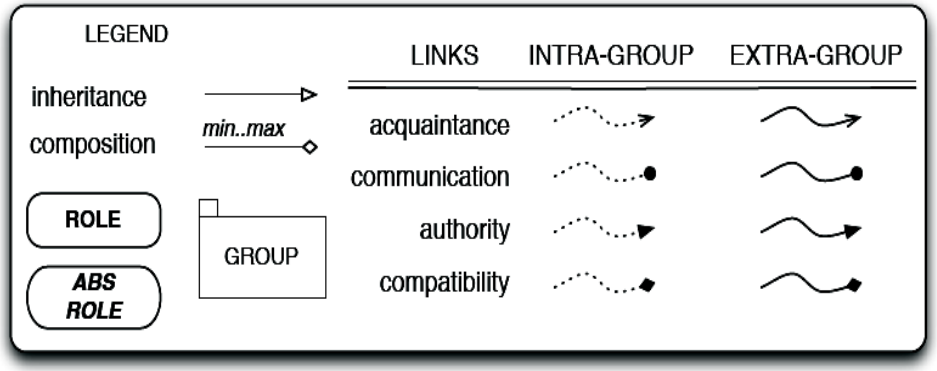
- Uma norma social trata-se de um **guia para conduta ou ação** a ser cumprido pelos membros de uma sociedade;
- Uma norma possui os seguintes aspectos:
  - **Regulação do comportamento;**
  - **Mecanismos de sanção;**
  - **Mecanismos de divulgação (líderes)**
- Problema: devida a autonomia dos agentes é possível que eles decidam por não seguir determinada norma;

Modelo Organizacional MOISE+

- MOISE+ é um modelo que estabelece componentes que formam uma organização e como eles podem contribuir para um SMA;
- O **Moise** especifica-os em três diferentes aspectos:
  - **Estrutural** (provê a estrutura organizacional em termos de grupos de agentes, papeis e relações funcionais entre os papeis);
  - **Funcional** (define o escopo comportamental dos agentes atuantes, padronizando comportamentos autônomos);
  - **Normativa** (refere-se aos papeis para as missões especificando um conjunto de normas em termos de permissões ou obrigações para completar uma missão).

Aspecto Estrutural





(a) Structural Specification

- Os **relacionamentos** podem ser especificados entre os papeis para definir autoridades, canais de comunicação e ligações conhecidas;
- A especificação permite a taxonomia de **grupos e ligações** intra-grupos;
- Grupos consistem em um conjunto de funções e propriedades relacionadas;
- As cardinalidades para funções dentro de um grupo indicam a quantidade máxima de agentes que podem desempenhar um **papel**;

Aspecto Funcional

- Oferece um conjunto de esquemas funcionais que, em conformidade com a SS, espera que vários agentes possa atingir os objetivos da organização;
- É vista como uma **árvore de decomposição de objetivos**, onde:
  - A raiz é o objetivo organizacional;
  - As folhas são os objetivos individuais de cada agente;
- Uma **missão** é o agrupamento de todos os objetivos que um agente se compromete;
- O exemplo a seguir mostra um cenário hospitalar;

Aspecto Funcional

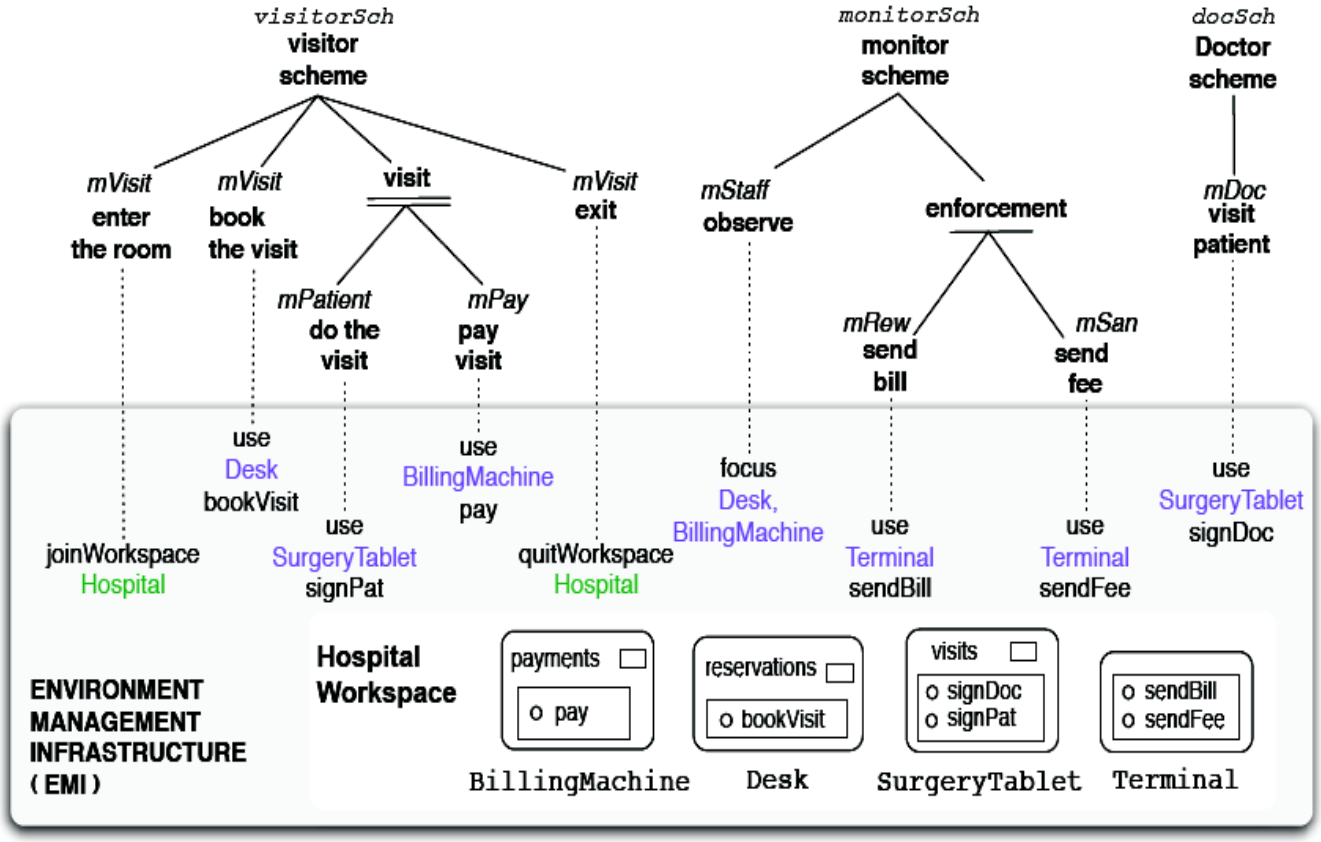


Fig. 2. (Above) Moise Functional Specification (FS) for the hospital scenario. Schemes are used to coordinate the behavior of autonomous agents. (Below) FS is used to find a set of environmental artifacts, and to map their functionalities in the EMI.

Aspecto Normativo

- Refere-se aos papeis (SS) para as missões (FS) especificando um **conjunto de normas**;
- Normas em Moise resultam em termos de **permissões ou obrigações** no comprometimento de uma missão;
- Isso permite criar **políticas específicas** para o comprometimento missionário;
- A figura a seguir mostra declarações de normas que regulam o cenário do hospital;
  - TTF (time to fulfill) refere-se ao tempo máximo que a organização irá esperar para que o agente cumpra a norma;

Especificação Normativa

id	*condition*	*role*	*type*	*mission*	*TTF*
*n*1		*Escort*	*obligation*	*mVisit*	–
*n*2		*Patient*	*obligation*	*mVisit*	–
*n*3		*Patient*	*obligation*	*mPatient*	–
*n*4		*Escort*	*permission*	*mPay*	5 minutes
*n*5	*unfulfilled*(*n*4)	*Patient*	*obligation*	*mPay*	5 minutes
*n*6		*Staff*	*obligation*	*mStaff*	–
*n*7		*Doctor*	*obligation*	*mDoc*	–
*n*8	*unfulfilled*(*n*5)∧ *unfulfilled*(*n*4)	*Staff*	*obligation*	*mSan*	1 day
*n*9	*fulfilled*(*n*4)∨ *fulfilled*(*n*5)	*Staff*	*obligation*	*mRew*	1 day
*n*10	*unfulfilled*(*n*6)	*Doctor*	*obligation*	*mStaff*	–

(b) Deontic Specification

Obrigado!

Dúvidas, sugestões, comentários?

IFPI - FMN - UESPI - CEEP

ENUCOMP

ENCONTRO UNIFICADO DE COMPUTAÇÃO EM PARNAÍBA