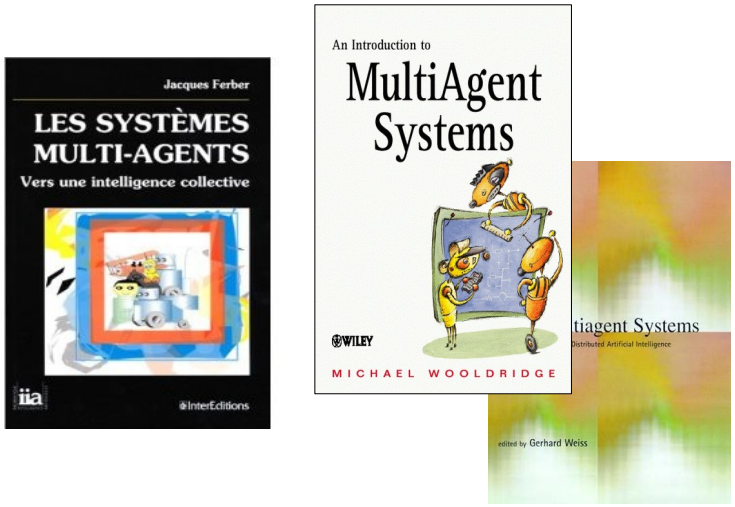
Advanced Human Machine Interactions

Introduction to multi-agent systems and autonomous agents

Alexandre Pauchet
INSA Rouen – Département ASI
BO.B.RC.18, alexandre.pauchet@insa-rouen.fr

Introduction



The field of Multi-Agent Systems (MAS) appeared in the 80's, thanks to the fusion of several disciplines

It is not (only)

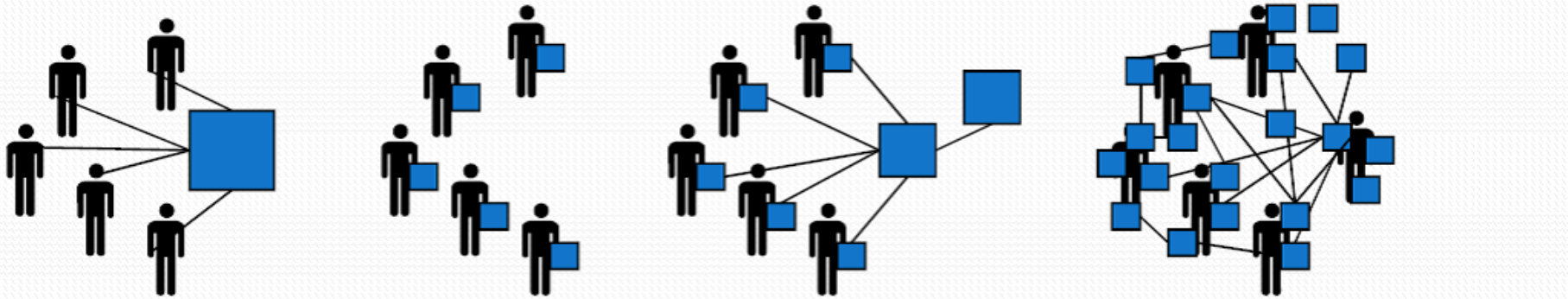
- a language
- standards
- an architecture
- a method...

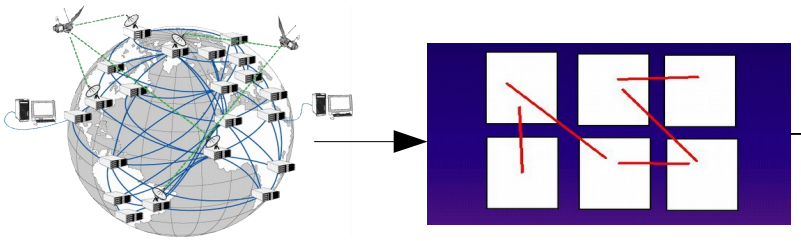
…but a group of features and characteristics allowing the development of systems with some properties (distributed, adaptive, flexible, 'smart', ...)

In an effort to follow modern IT evolution...

Systems and communication

- Évolution de l'informatique vers le « Pervasive Computing » (informatique diffuse)





Internet

web

- Cloud computing
- Connected objects
- Ubiquitous/pervasive computing
- Internet of things

Large scale systems that must **adapt** to **dynamic** environments, **interacting** with humans on a regular basis.

Artificial Intelligence

Statistical approach

- Formal neural nets
- Multi-layer perceptron

Symbolic approach

- Turing test, ELIZA
- Lisp, Prolog, …
- Expert systems (Mycin, ...)

Rise of **distributed** AI and **collective intelligence** to solve more and more **complex** problems, subject to **uncertainties** and **incomplete knowledge**

What is an agent? (according to Sycara)

An **agent** is "*a software system whose main features are its situated nature, autonomy, adaptativeness and sociability.*"

- **Situated nature** : sensory input => actions modifying the environment
- **Autonomy** : controls its actions and its internal state
- **Adaptability** : allows agents to
  - React in a flexible way
  - Take initiatives towards a goal
  - Learn
- **Sociability** : interaction with other agents

MAS definition (according to Ferber)

A **MAS** is made up of:

- an **environment** *E*
- a **set of situated objects** *O* (*a position into E is associated with them*)
- **a set of agents** *A* (*A included into O ; they are active compared to other passive objects of O*)
- **a set of relations** *R* uniting objects
- **a set of operations** *Op* allowing agents *A* to perceive and manipulate objects in *O*
- **operators** representing the application of those operations and the world reaction to the resulting modification attempts (called the universe laws)

Computer science, AI and MAS

A few definitions

- A ***software agent*** is computer system capable of acting **autonomously** to achieve its **own goal**
- A ***multi-agent system*** is made up of software agents **interacting** in a physical or virtual **environment**
- An ***heterogeneous MAS*** (or a ***mixed community***) is a system made up of software agents interacting with human agents.

Application fields

- Distributed solving of problems
- Modelling, individual-centered simulation
- Distributed software applications
- Human-Machine interaction and communication within mixed communities

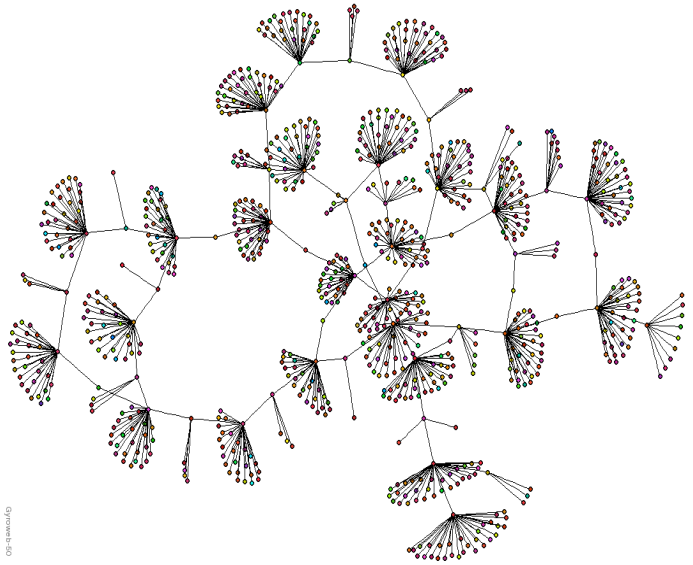
Multi-agent application examples

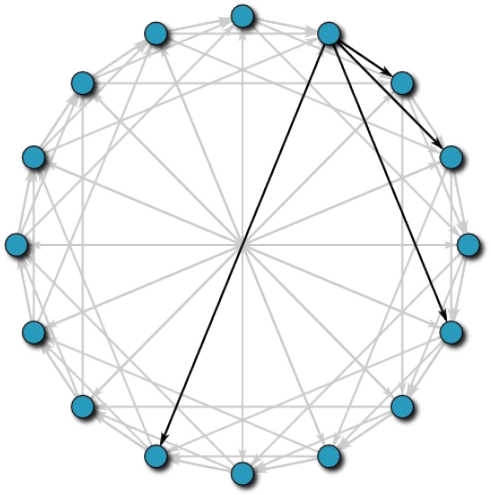
Graphical simulation

Example: MASSIVE system

- Large scale simulation
- Autonomous decision making
- Local/limited perceptions or actions
- Interactions thanks to message sequences
- Behaviour modelling
- Heterogeneity of agents
- No overall control, but *emergence* of global behaviour during realistic simulations

Peer-to-peer networks





- *Aim*: giving access to resources located on the network nodes
- *Working principle*: distributed indexing or discovery algorithms, and/or repository necessitating a collaborative activity of agents/nodes
- *Multi-agent feature*: protocols implemented by interaction rules, open systems, autonomous agents (sometimes heterogeneous)

Swarm robotics

Robocup

Robocup rescue

- *Aim*: coordinate a mobile robots fleet
- *Working principle*: local decision-making contributing to an overall goal achievement
- *Multi-agent feature*: locally noised and limited perception, located environment, distributed planning

Fields of development for a multi-agent application

Vowels breakdown

A MAS is made up of [Demazeau, 95]

- **A**gents (2)
- **E**nvironements (1)
- **I**nteractions (3)
- **O**rganizations (4)

AEIO 1/4: Environment

Physical environment

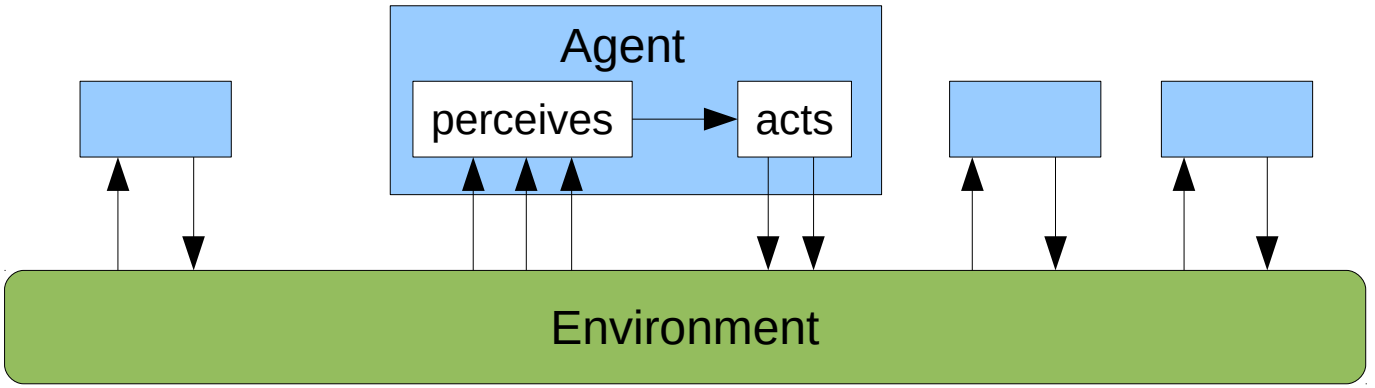
- Modelling of the **world states** and their dynamics
- Definition of **action** and **perception** primitives
- Especially present in embedded systems (mobile robotics, intelligent ambiance)

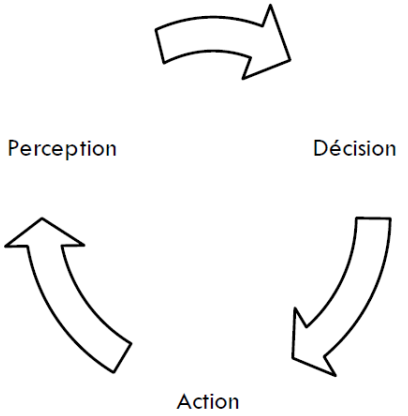
Situated environments

- Determine an agent **position** along with perception and action **constraints**
- Mostly used in simulation

AEIO 2.1/4: Agent architectures
Reactive agents

- Simple architecture implementing a behaviour responding to events *(stimuli)*
- No environment representation
- Indirect communications (*via* the environment)
- No events or behaviours history
- Intelligence is embedded into the MAS organisation and appears by *emergence*





Difficulties: MAS + settings

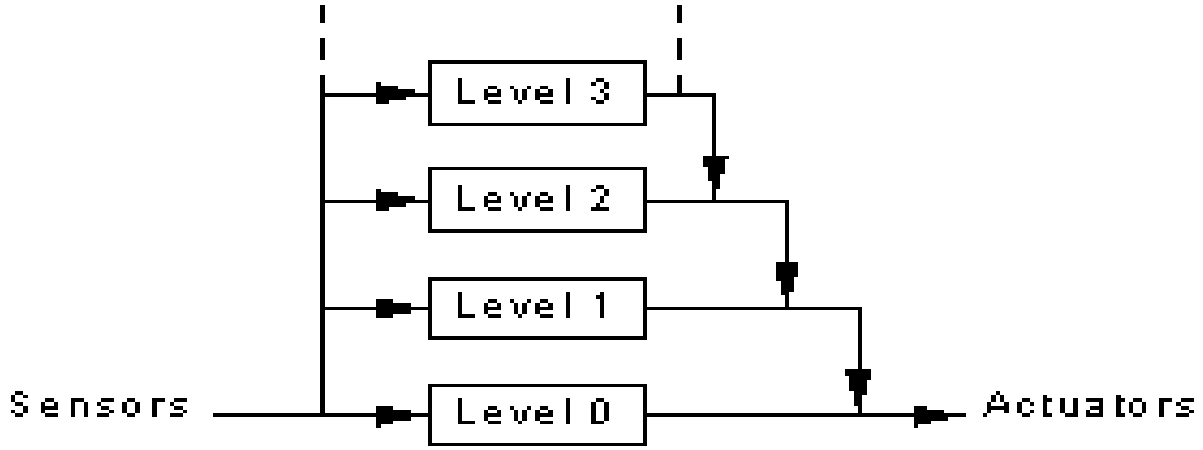
Reactive control cycle

Data:

- Rules “condition => action”
- Set of percepts

```
while (true) {
    percepts := see();
    state := interpret(percepts);
    rule := match(state,rules);
    execute(rule[action]);
}
```

Example: Subsomption architecture [Brooks, 96]



- Each layer interprets its inputs and creates a response
- Possibility to delete inputs and inhibit outputs

AEIO 2.2/4: Agent architectures
Cognitive agents

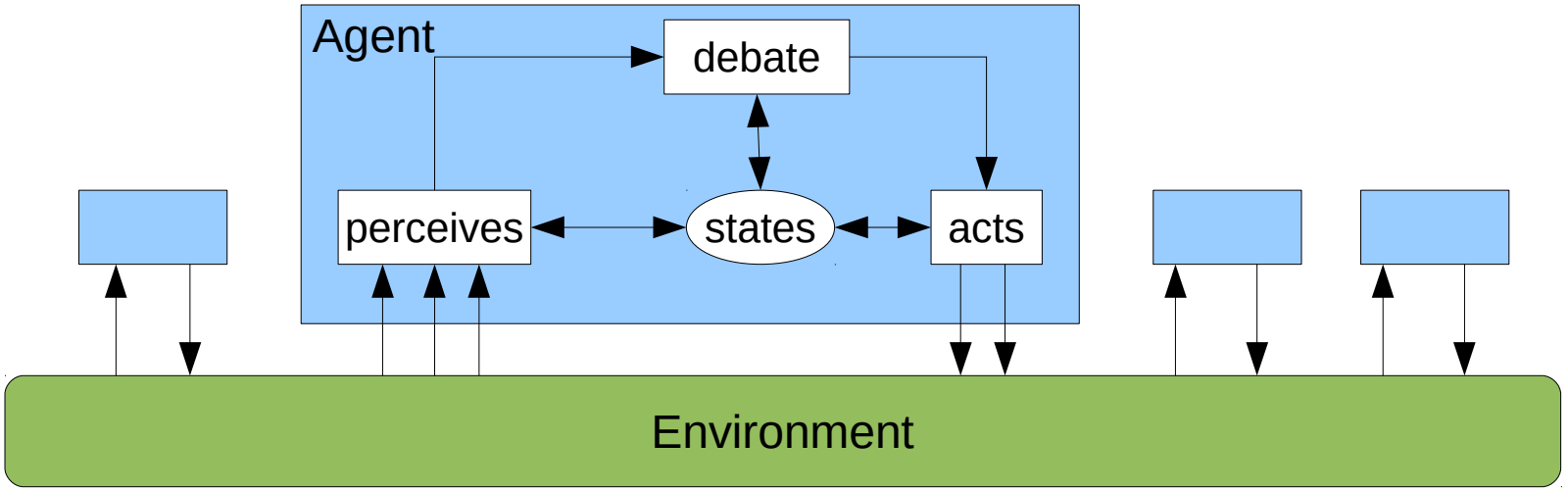
- Agents = intelligent entities, able to solve problems by themselves
- Intentionality: agents have goals and explicit plans allowing them to reach their goals
- MAS tries to organise the cooperation of traditional expert systems, through communications
- Intelligence is within the agents

  Difficulties: agent architecture + communications

- AHMI: Human - cognitive agent interaction

Cognitive agents

- Action choice is subject to **deliberation**
- Explicit **representation** of the environment, the agents' goals and their abilities
- An history can be used to take decision, learn or plan a sequence of actions



Deliberative control system

Data

- Set of states
- Set of percepts
- Set of actions

```
states := initialise_state();
while (true) {
    percepts := see();
    states := update_states(percepts)
    action := deliberate(states);
    execute(action);
}
```

AEIO 3/4: Interactions

Indirect interaction

- Mainly for reactive agents
- Communication between agents by **actions/perceptions on the environment**
- Example: pheromone deposits

Direct interaction

- Mainly for cognitive agents
- Sending/reception of structured **messages**
- Sequencing into interaction **protocols**
- Interaction is sometimes considered as an action

AEIO 4/4: Organisations

Top-down approach

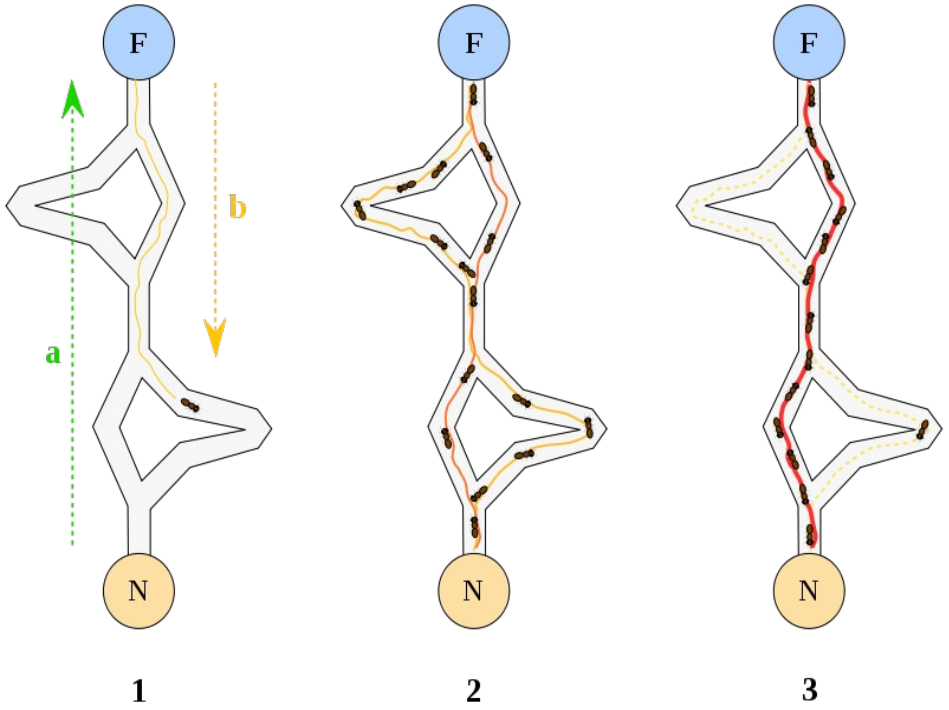
- Formal definition, **explicit** of an organisation
- **Prescriptive** or **restrictive** use for agents
- MAS readapts *via* re-organisation

Bottom-up approach

- **Implicit** organisation resulting from local agents' behaviour
- **Emergence** of some organisation
- MAS readapts *via* auto-organisation

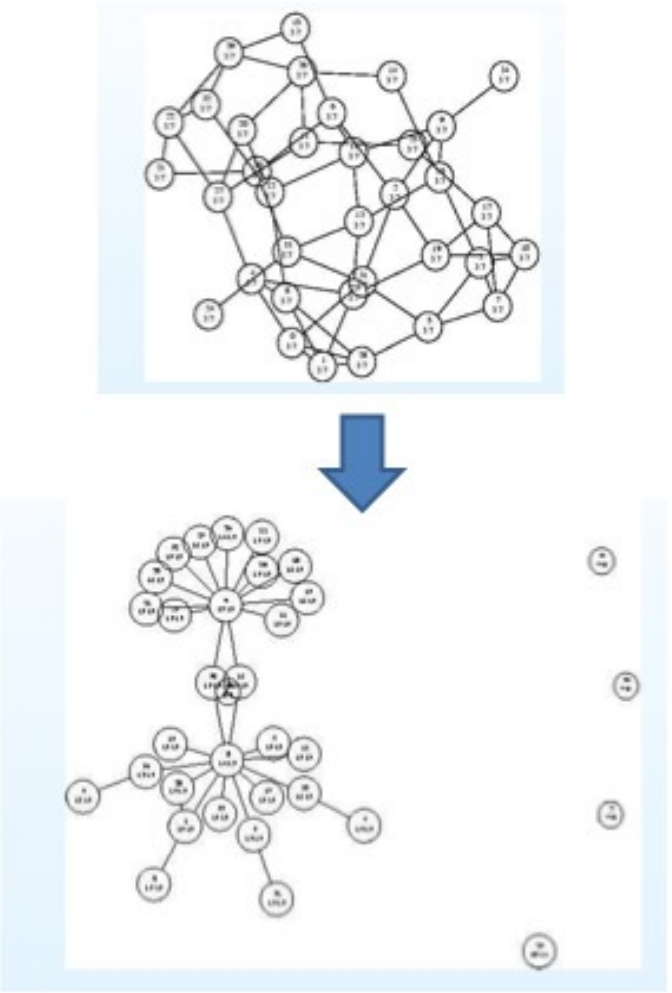
Auto-organisation

Ant colony



https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

Peer-to-peer network regulation [Grizard, 06]



To sum up, at agent level

A software agent has several features

- It is **autonomous**
- It has its own decision ability (**pro-activity**)
- It **interacts** with other agents
- **It can take action** on and **perceive** its environment

To sum up, at MAS level

A multi-agent system

- is an **open** system
- that works in a **decentralised** manner
- is a **compromise** between **control/regulation** and **emergence**

Its properties allow to develop **flexible large scale** systems

A responsive architecture:
Eco Problem Solving (EPS)

Problem solving (1/5)
Formalisation

Definition

A *problem* is defined by :

- an initial state,
- a goal (final state),
- a set of operators allowing to pass from one state to another.

$\Longrightarrow$ To solve a problem : give a sequence of operators allowing to pass from the initial state to the final one.

Representation

The states of the problem are represented by an oriented graph whose vertices are states.
There is an edge between nodes u and v iff there is an operator that turns u into v.

Problem solving (2/5)
Examples of addressed problems

Exemples

- Proving theorems, solving equations
- Cube world, tower of Hanoi
- Eight queens puzzle
- River crossing puzzle (wolf, goat and cabbage problem, Seven Bridges of Koenigsberg)
- Timetables
- Path search in a maze
- Magic squares, sudoku, crosswords
- Travelling salesman problem

Remarque

Most of "real" problems can be identified as one of the above formalisation.

Problem solving (3/5)
Exact solving

Principle

- Calculation of all possible paths going from the initial state to the final one.
- Exponential computational complexity
  $\Rightarrow$ Combinatorial explosion (memory and/or time).

Example : chess

- State space : all authorized configurations of the board
- Initial state : initial configuration of chess
- Final state : all configurations of the board implying checkmate
- Rules such as : "*if(white pawn is in square(i,2), and square(i,3) is empty, and square(i,4) is empty) then move white pawn from square(i,2) to square(i,4)*"

Problem solving (4/5)
Heuristic methods

Principle

Localized search view and stop after a certain amount of time.
⇒ Approximation of solution
⇒ Admissible quality solution, obtained in minimal time

Exemples

- Simulated annealing,
- Tabu search
- Evolutionary algorithms (or genetic algorithms)
- **Multi-Agent systems (eco-problem solving)**

Problem solving (5/5)

Problem solving and Human-Machine Interactions

Principle

Problem solving

- Dynamic
  - human intervention,
  - open or evolving environment
- with human interaction
  - cooperation,
  - collaboration,
  - competition

Exemple

- Timetable bargaining
- Games

Eco-problem solving (1/5)
Principles

- Used in problem solving
- Based on a set of reactive agents sharing the same behaviour, called *eco-agents*
- As for any MAS, decentralised solving, and emergence of a large-scale behaviour
- Also interesting for dynamic problem solving
- **Goal : reach a stable state (the problem's solution)**

Eco-problem solving (2/5)

Eco-agents

- Their behaviour can be compared to a sequence of perception/action instructions
- Localised environment perception : network of dependencies made up of neighbour agents
- Each agent has a goal : *satisfaction*
- An agent is dissatisfied when *obstructed* by other agents
- If an agent is obstructed, it can *attack* another one
- An attacked agent is dissatisfied and can *flee*

Representation with finite-state automata

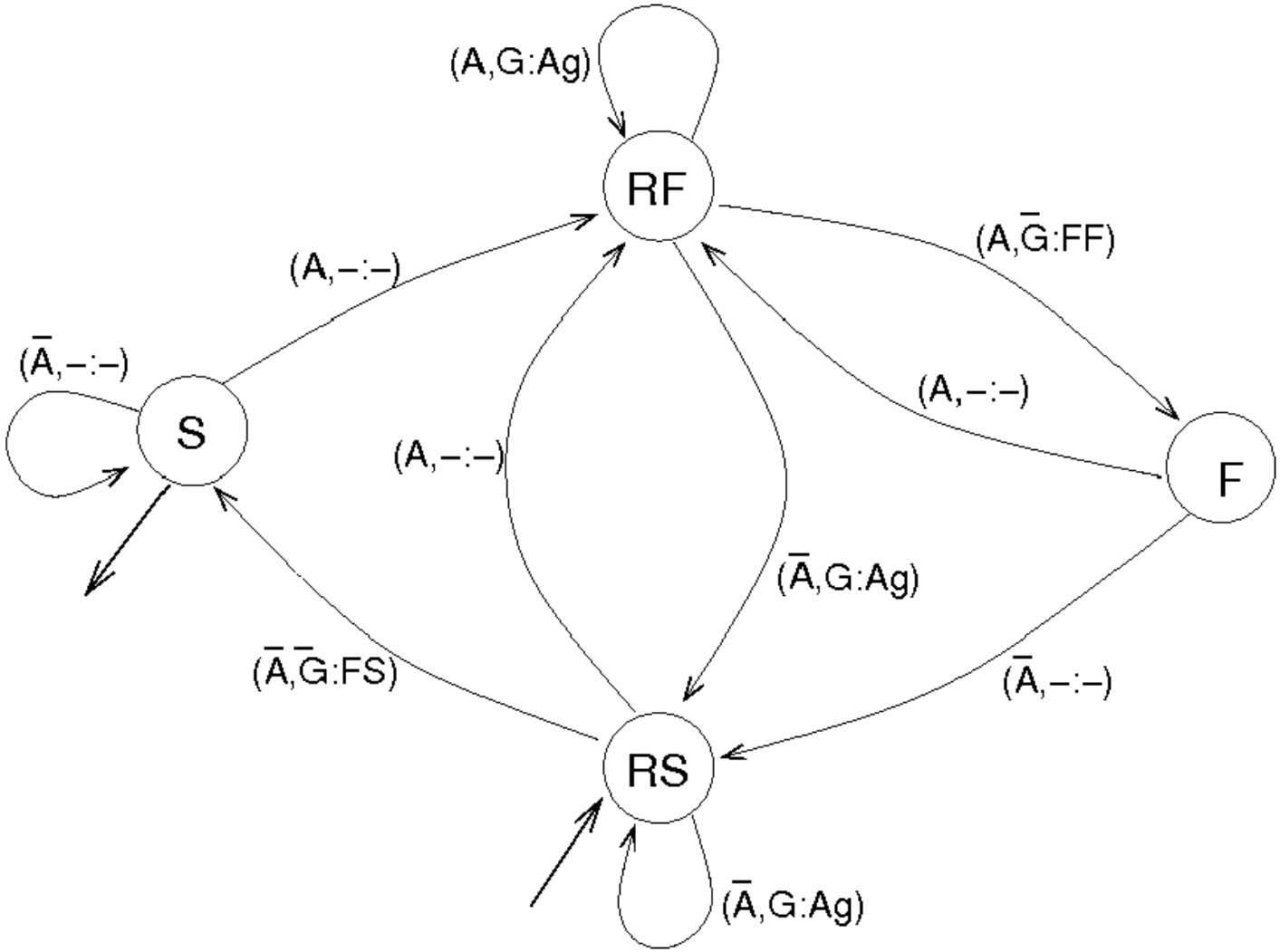
Eco-problem solving (3/5)

Eco-agents architecture

- 4 internal states
  - S Satisfaction (final state)
  - SS Seeking satisfaction (initial state)
  - SE Seeking escape
  - E Escape state
- Perceptions
  - $A/\overline{A}$ attacked/not attacked by an other agent
  - $G/\overline{G}$ obstructed/not obstructed by an other agent
- Actions
  - FS Do satisfaction (action achieving its goal)
  - AT Attack an agent
  - FF Flee

Eco-solving problem (4/5)

Eco-agents behaviour automaton



Eco-problem solving (5/5)

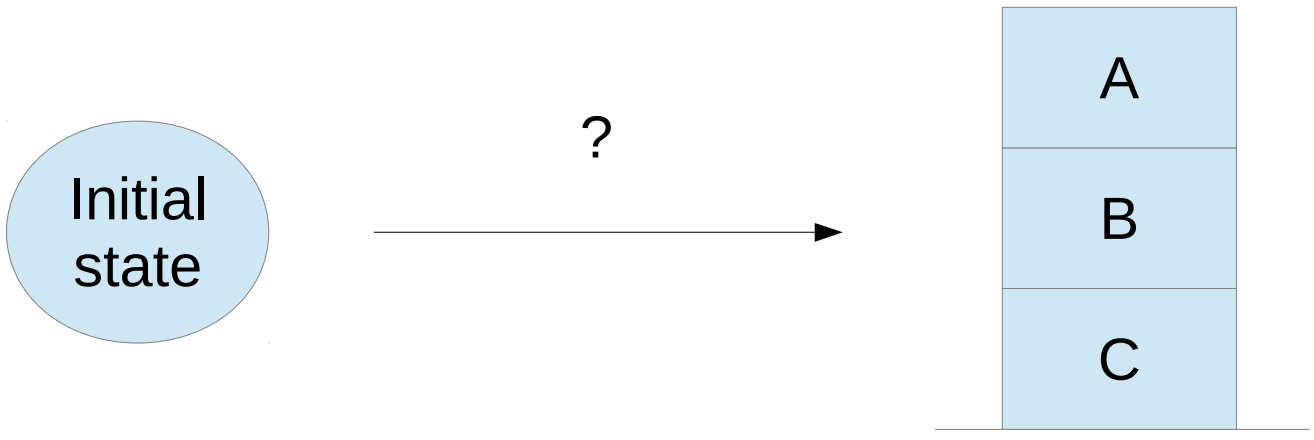
Solving eco-problems

- An eco-problem is defined as
  - A set of agents, featuring a *goal* and a *behaviour* (automaton + basic actions depending upon application)
  - An initial configuration, agents set into their initial state
  - An ending criterion (most of the time : all agents are satisfied)

- Solving a problem through eco-problem solving
  1. Identify the different types of eco-agents
  2. Identify the satisfaction conditions of each eco-agent
  3. Associate a structure describing eco-agents
  4. Specify methods `doEscape`, `doSatisfaction` and `Attack(X)`

Example : cube world

Problem description



From an initial state (a set of cubes on the table), find a sequence of actions to execute in order to reach a final state (a specific configuration of the cubes)

Example : cubes world

Solving through eco-problem solving

- 2 types of eco-agents : the table and the cubes
- Final state : cube A would be satisfied if on the table, cube B if on top of cube A, cube C if on top of cube B
  {A.on(Table), B.on(A), C.on(B)}
- To a cube, fleeing consists in being put on the table

Example : cubes world

Table eco-agent

Attributes

Methods

```
Method satisfied?():boolean {
  return(True)
}
```

Example : cubes world

Cube eco-agent

```
Attributes
  over: Cube
  under: EcoAgent (Cube or Table)
  goal: EcoAgent (Cube or Table)

Methods
  Method doSatisfaction() {
    this.goal.over := this
    this.under.over := null
    this.under := goal
  }

  Method doEscape(p: EcoAgent) {
    this.under.over := null
    this.under := p
  }

...
```

Example : cubes world

Cube eco-agent

```
...

  Method satisfied?(): boolean {
    if this.under = this.goal then return(True)
    else return(False)
  }

  Method findPlaceToFlee() {
    return(Table)
  }

  Method obstructorsFlee() {
    if this.over <> null then // return the obstructors list
      return(this.over)
    else
      return(null) // else nothing
  }

...
```

Example : cubes world

Cube eco-agent

```
...

  Method obstructorsSatisfaction() {
    EcoAgent r := null
    if this.over <> null then
      r := this.over
    else if this.goal.over <> null then
      r := this.goal.over
    return(r)
  }
```

Interaction with eco-agents

- Heterogeneous MAS
  - Reactive MAS, easily configurable
  - Mixed community : human / artificial agents
  - Indirect interactions
  - Seeking stable state, human disturbance

- Eco-problem solving through heterogeneous MAS
  - Features identical to a heterogeneous MAS made up of eco-agents and humans
  - Agents and humans cooperate to solve a problem

References

Links

- Link to lesson by C. Bertelle
- Link A. Drogoul

Books and articles

- J. Ferber, "*Les Systèmes Multi-Agents : vers une intelligence collective*", InterEditions, 1995
- K.P. Sycara, "*The many faces of agents*", AI Magazine, 19(2), pp. 11-12, 1998

Cognitive architecture examples

Problem to solve: cube world
States description with predicates

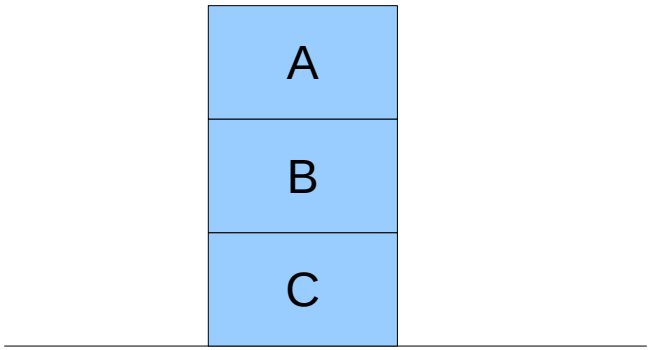
- Robotic hand status

  HANDEMPTY, HOLDING(x)

- State of a cube

  Above: CLEAR(x)

  Below: ONTABLE(x), ON(x,y)



Example:

HANDEMPTY, CLEAR(A), ON(A,B), ON(B,C), ONTABLE(C)

Cube world operations

- pickup(x)

  PRE & DEL: ONTABLE(x), CLEAR(x), HANDEMPTY

  ADD: HOLDING(x)

- putdown(x)

  PRE & DEL: HOLDING(x)

  ADD: ONTABLE(x), CLEAR(x), HANDEMPTY

- stack(x,y)

  PRE & DEL: HOLDING(x), CLEAR(y)

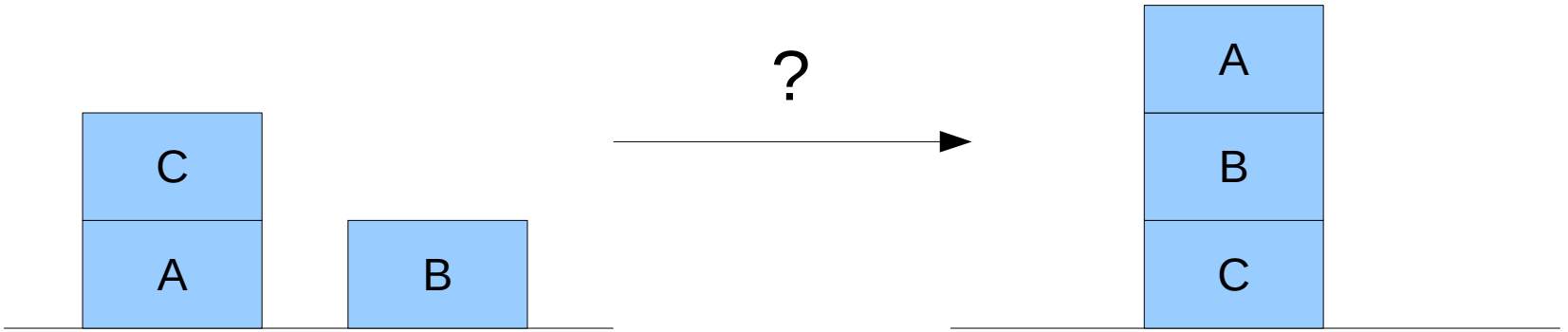
  ADD: HANDEMPTY, ON(x,y), CLEAR(x)

- unstack(x,y)

  PRE & DEL: HANDEMPTY, ON(x,y), CLEAR(x)

  ADD: HOLDING(x), CLEAR(y)

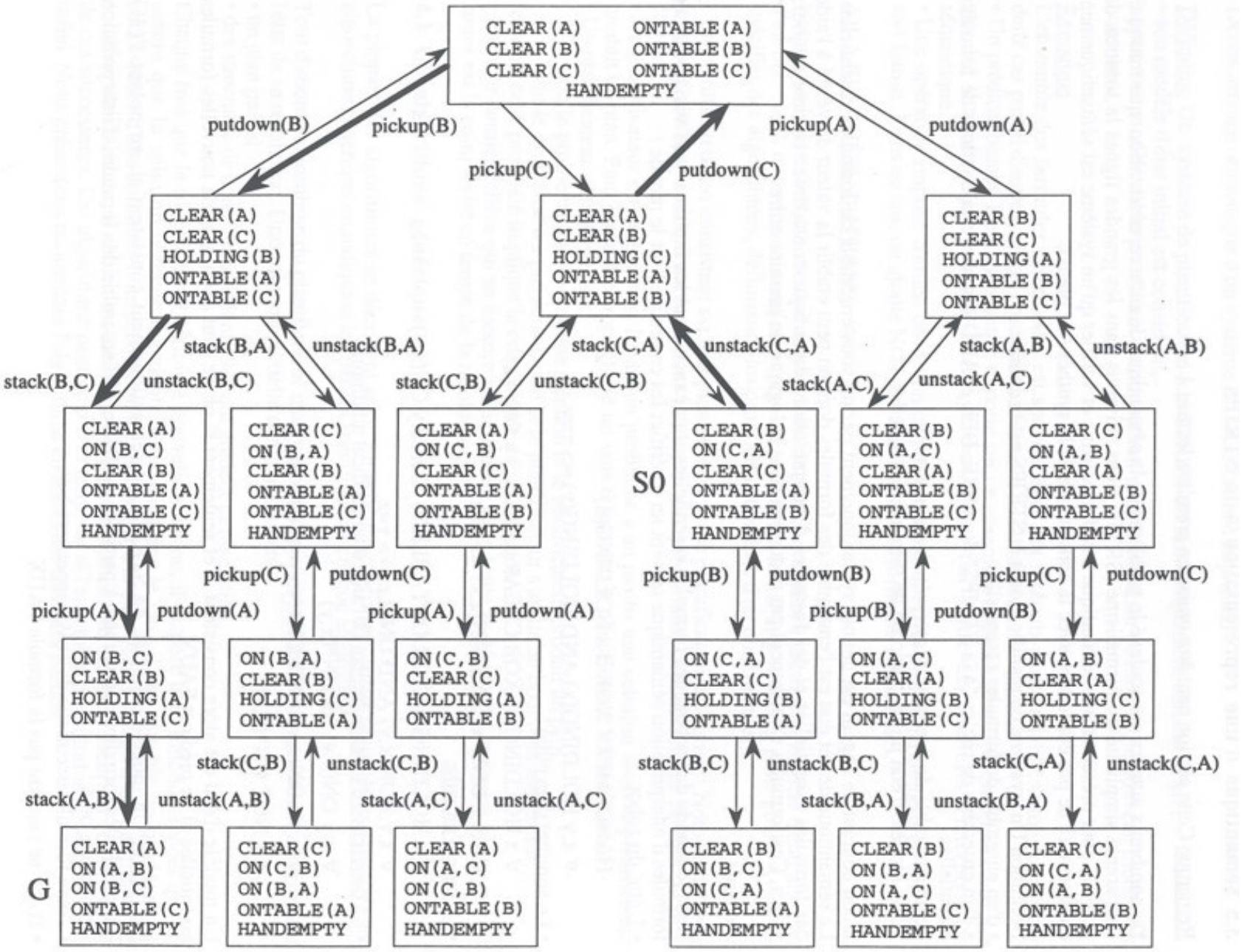
Initial and final states



Possible sequence:

unstack(C,A), putdown(C), pickup(B), stack(B,C), pickup(A), stack(A,B)

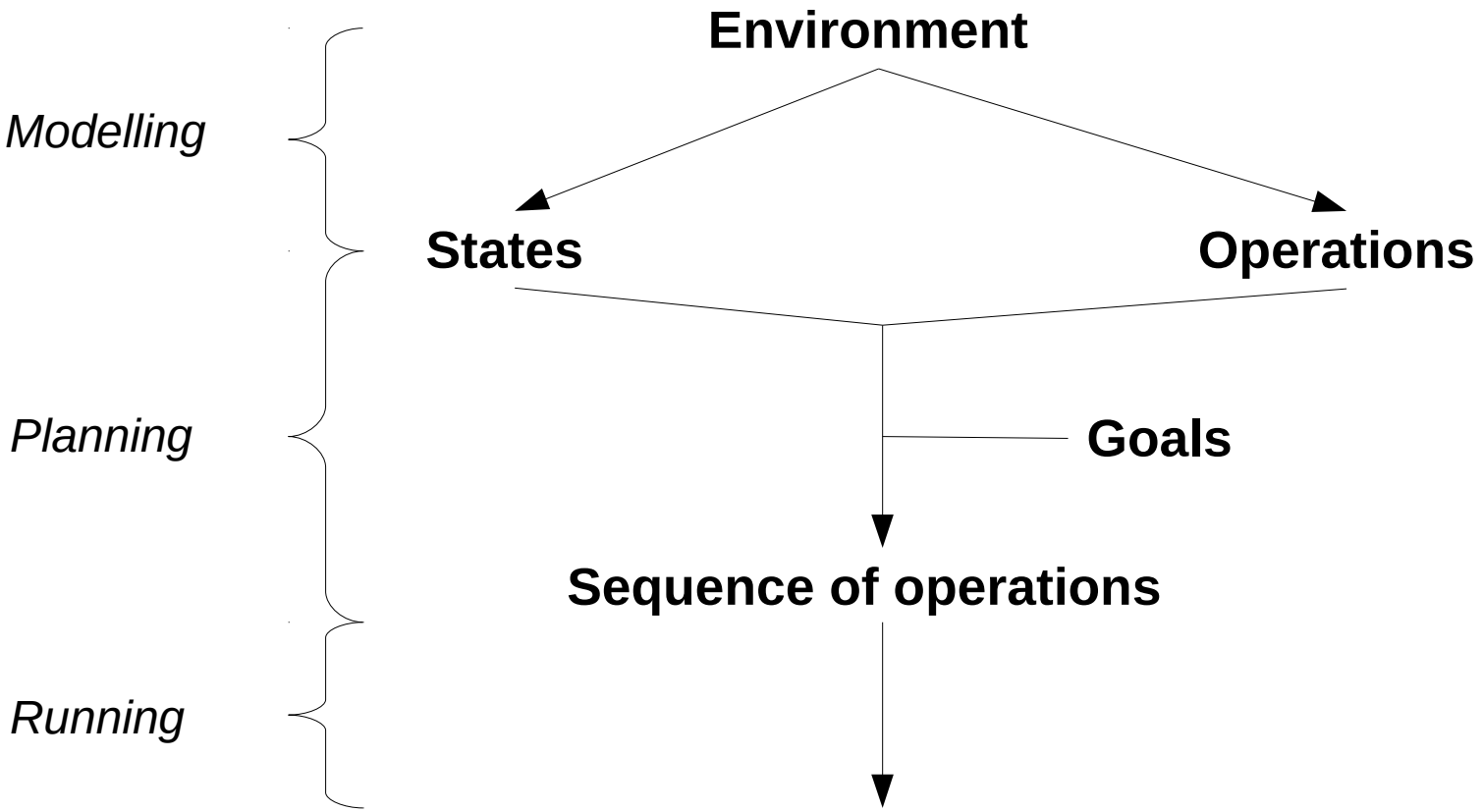
State graph


CLEAR(A) ONTABLE(A)
CLEAR(B) ONTABLE(B)
CLEAR(C) ONTABLE(C)
HANDEMPTY
putdown(B)
pickup(B)
pickup(C)
putdown(C)
pickup(A)
putdown(A)
CLEAR(A) CLEAR(C) HOLDING(B) ONTABLE(A) ONTABLE(C)
CLEAR(A) CLEAR(B) HOLDING(C) ONTABLE(A) ONTABLE(B)
CLEAR(B) CLEAR(C) HOLDING(A) ONTABLE(B) ONTABLE(C)
stack(B,A)
unstack(B,A)
stack(C,A)
unstack(C,A)
stack(A,B)
unstack(A,B)
stack(B,C)
unstack(B,C)
stack(C,B)
unstack(C,B)
stack(A,C)
unstack(A,C)
S0
CLEAR(A) ON(B,C) CLEAR(B) ONTABLE(A) ONTABLE(C) HANDEMPTY
CLEAR(C) ON(B,A) CLEAR(B) ONTABLE(A) ONTABLE(C) HANDEMPTY
CLEAR(A) ON(C,B) CLEAR(C) ONTABLE(A) ONTABLE(B) HANDEMPTY
CLEAR(B) ON(C,A) CLEAR(C) ONTABLE(A) ONTABLE(B) HANDEMPTY
CLEAR(B) ON(A,C) CLEAR(A) ONTABLE(B) ONTABLE(C) HANDEMPTY
CLEAR(C) ON(A,B) CLEAR(A) ONTABLE(B) ONTABLE(C) HANDEMPTY
pickup(A)
putdown(A)
pickup(C)
putdown(C)
pickup(B)
putdown(B)
ON(B,C) CLEAR(B) HOLDING(A) ONTABLE(C)
ON(B,A) CLEAR(B) HOLDING(C) ONTABLE(A)
ON(C,B) CLEAR(C) HOLDING(A) ONTABLE(B)
ON(C,A) CLEAR(C) HOLDING(B) ONTABLE(A)
ON(A,C) CLEAR(A) HOLDING(B) ONTABLE(C)
ON(A,B) CLEAR(A) HOLDING(C) ONTABLE(B)
stack(C,B)
unstack(C,B)
stack(A,B)
unstack(A,B)
stack(A,C)
unstack(A,C)
stack(B,C)
unstack(B,C)
stack(B,A)
unstack(B,A)
stack(C,A)
unstack(C,A)
G
CLEAR(A) ON(A,B) ON(B,C) ONTABLE(C) HANDEMPTY
CLEAR(C) ON(C,B) ON(B,A) ONTABLE(A) HANDEMPTY
CLEAR(A) ON(A,C) ON(C,B) ONTABLE(B) HANDEMPTY
CLEAR(B) ON(B,C) ON(C,A) ONTABLE(A) HANDEMPTY
CLEAR(B) ON(B,A) ON(A,C) ONTABLE(C) HANDEMPTY
CLEAR(C) ON(C,A) ON(A,B) ONTABLE(B) HANDEMPTY

Planning

Planning

Determine a sequence of actions to perform in order to reach a certain goal



Planning algorithm

```
If success(Initial state)
   Return (Initial state)
Else
   Scons = {Initial state}
   Stoex = {Initial state}
   While Stoex ≠ Ø do
      state = chose(Stoex)
      If (next_state = new_successor(state) ≠ Ø) then
         If next_state ∉ Scons then
            If success(next_state) then
               Return (next_state)
            EnfIf
            Scons = Scons ∪ next_state
            Stoex = Stoex ∪ next_state
         EndIf
      Else
         Stoex = Stoex – {state}
      EndIf
   EndWhile
   Return (failure)
EndIf
```

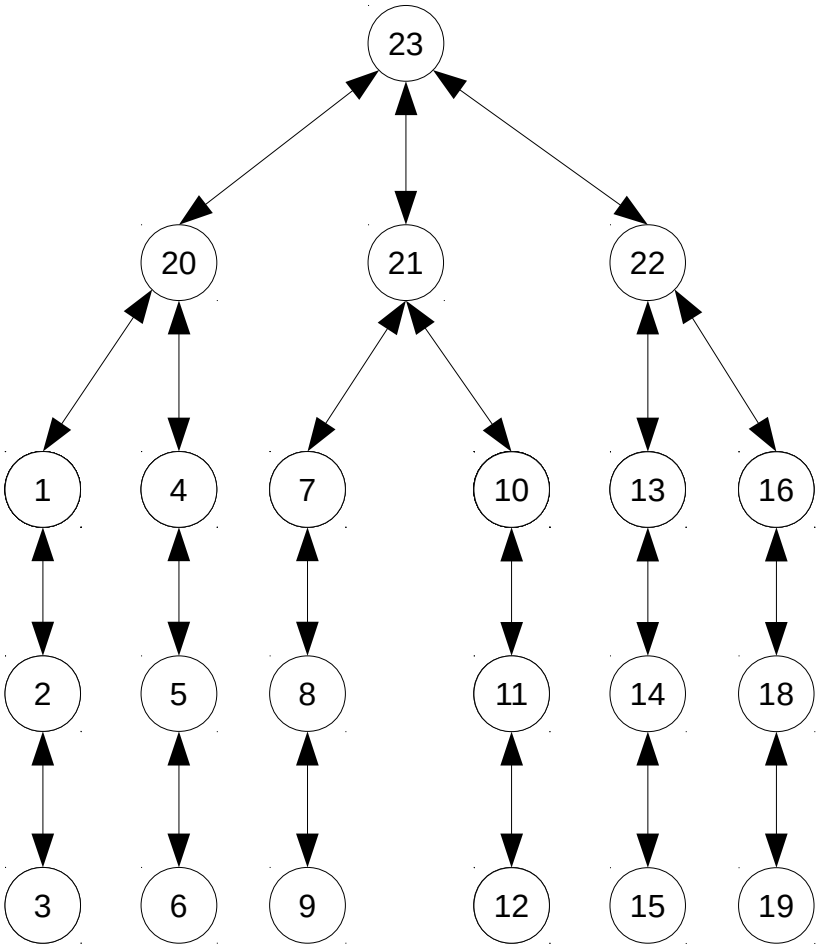
Solution searching strategy is implemented here

Forward linkage

Principle: begin from the initial state and explore the states produced by applying successive operations

- Complete approach that ends if the world models are finite
- Depth-first or breadth-first search
- Complexity proportional to the number of state models

Example



- Forward linkage
- Selection of the oldest state with the less successors

1) {10} ⇒ {11, 21}

- CLEAR(B) & ON(C,A) & CLEAR(C) & ONTABLE(A) & ONTABLE(B) & HANDEMPTY
- *Candidates* : pickup(B), unstack(C,A)

2) {10, 11} ⇒ {12, 21}

- ON(C,A) & CLEAR(C) & HOLDING(B) & ONTABLE(A)
- *Candidates* : putdown(B), stack(B,C)

3) {10, 11, 12} (failure) ⇒ {21}

4) {10, 21} ⇒ {7, 23}

...

Backward linkage

Principle: begin from the goal to find the initial state

The STRIPS planner generates intermediate solutions with 2 operators

- **Decomposition**: if the solution under consideration is composed, propose solutions in the appropriate order
- **Regression**: if the solution considered is elementary, choose an action that leads to it

Example

- CLEAR(A) & ON(A,B) & ON(B,C) & ONTABLE(C) & HANDEMPTY
  - *Candidates* : stack(A,B)
- ON(B,C) & CLEAR(B) & HOLDING(A) & ONTABLE(C)
  - *Candidates* : pickup(A), unstack(A,B)
- CLEAR(A) & ON(B,C) & CLEAR(B) & ONTABLE(A) & ONTABLE(C) & HANDEMPTY
  - *Candidates* : stack(B,C), putdown(A)
- CLEAR(A) & CLEAR(C) & HOLDING(B) & ONTABLE(A) & ONTABLE(C)
  - *Candidates* : pickup(B), unstack(B,A), unstack(B,C)
- ...

Example of cognitive agents architecture allowing planning: BDI model

BDI model

Based on a cognitive model of intentionality
[Georgeff, 83] [Bratman, 90]

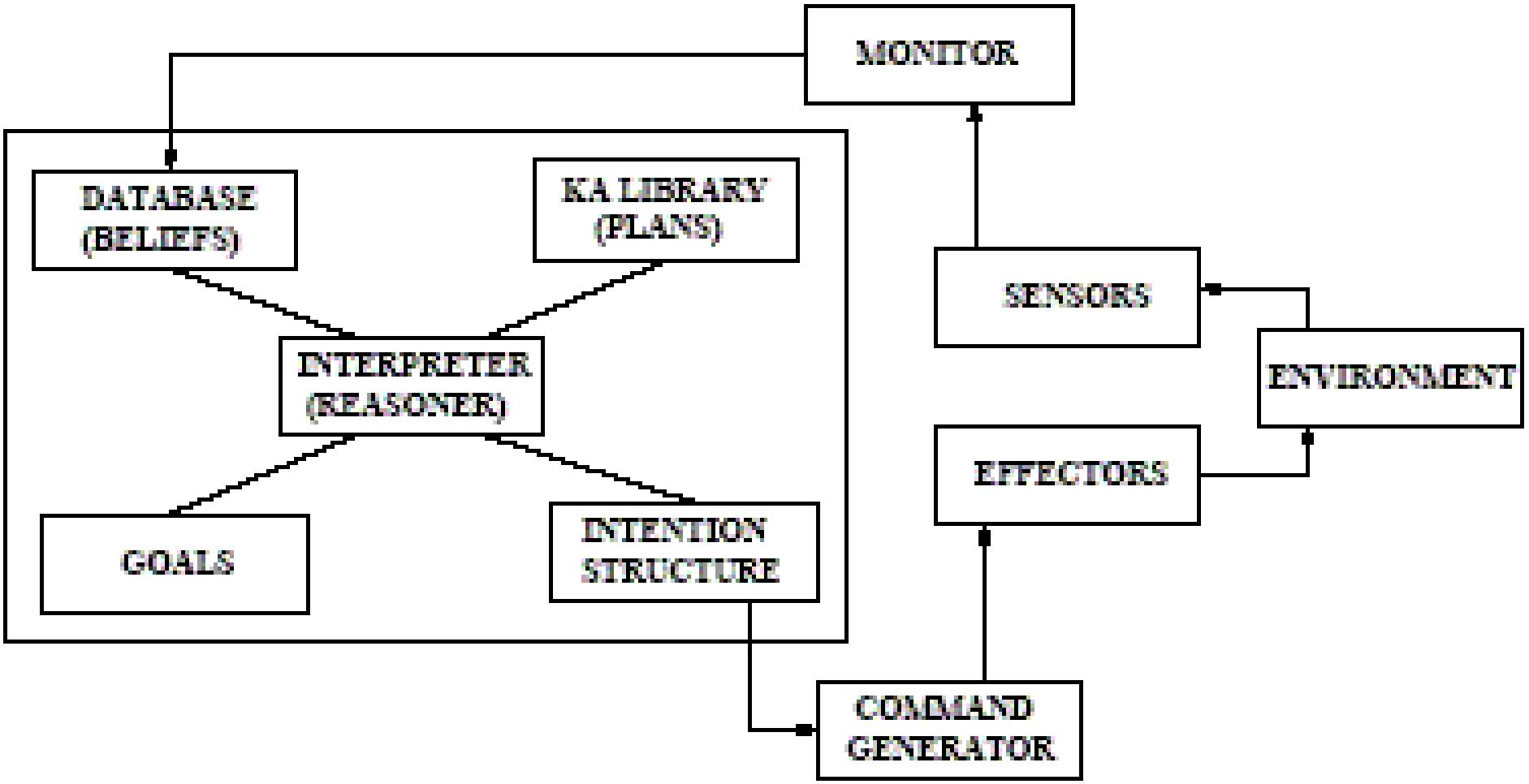
A BDI model is made up of

- A set of **Beliefs** upon itself and the world (modalities or predicates)
- A set of potentially conflictual **Desires**
- A set of consistent and not conflictual **Intentions**
- Reasoning mechanisms to update beliefs, chose desires and generate intentions

BDI model implementations

- Definition of an architecture based upon this model of reasoning (e.g.: PRS [Georgeff, 87])
- Formalisation in modal logic (e.g.: [Rao & Georgeff, 93])
- Agent programming languages (e.g.: Jason)

Procedural Reasoning System [Georgeff, 87]



BDI formal model [Rao & Georgeff, 93]

Preamble

- An agent perceives the current situation as a **world state**
- A world state is made up of *true*, *false* or *unknown* facts represented by predicates (e.g.: onTable(cube_A), not_clear(cube_B), ...)
- The representation of the world can be made in the hypothesis of a closed world or an open world

BDI formal model [Rao & Georgeff, 93]

BDI model formulas

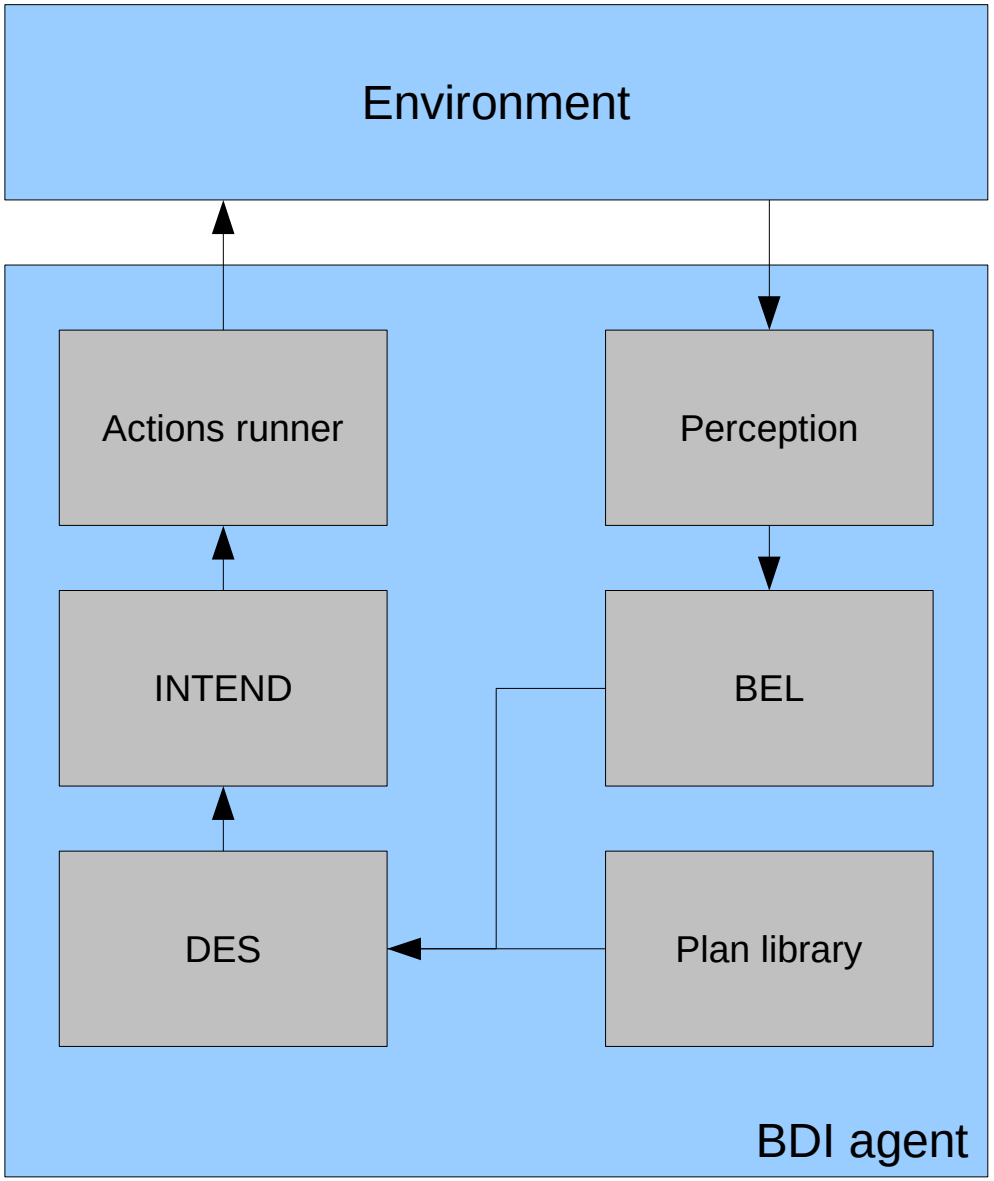
- A state formula *s* is
  - A proposition (a world state)
  - A conjunction or negation of state formulas (s1 AND s2, NOT s3)
  - BEL(*s*), DESIRE(*s*), INTEND(*s*)
- A path formula *p* is
  - A state formula
  - A conjunction or negation of path formulas
  - F *p*, G *p*, … timed operators (*p* will be true at least once, *p* will always be true, ...)

BDI formal model [Rao & Georgeff, 93]

BDI predicates examples

- BEL(onTable(cube_A))
- BEL(NOT onTable(cube_A))
- INTEND(onTable(cube_B))
- BEL(does(take(?X)) AND onTable(?X) → NOT onTable(?X))
- INTEND(clear(cube_B))

How it works



While (stop condition)

1. Get new perceptions
2. Update beliefs
3. Update the desire stack depending on "activatable" plans
4. Update the intention stack
5. Run the first intention of the intention stack

Plans library

- Plan library encodes a set of "activatable" sub-plans depending on beliefs
- A desire will be satisfied thanks to a set of intentions encoded by sub-plans

Planning is pre-wired !

Example: sub-plan « Paint a picture »

Pre: `BEL(picture-not-painted)`

Post: `DES(painted-picture)`

Plan: `INTEND(take-brush) AND INTEND(soak-brush) ...`

Multi-agent programming language

BDI programming language example: Jason

http://jason.sourceforge.net/Jason/Jason.html

- Developed in Java by R. Bordini and J. F. Hubner
- Inspired by AgentSpeak formalism

Basic concepts

- Goals: *!goal*
- Internal actions: *.action(...)*
- States addition (+) and substraction (-) operators
- Reaction plan to an event: *event* <- *actions*

Simple example of Jason code

```
// Agent tom in project greeting.mas2j
!start.

+!start : true <- .send(bob,tell,hello).

+hello[source(A)]
  <- .print("I receive an hello from ",A);
     .send(A,tell,hello).
```

Communication between agents (humans or software agents)

Communicating is taking action!

- Speech act theory (Austin, Searle), computational formalisation (Searle, Vanderveken)
  - Locutionary dimension
    - production of signs, creation of the communication action
  - Illocutionary dimension
    - intention expressed by the speaker
  - Perlocutionary dimension
    - effect on the speaker
- Dialogue acts (Bunt) distinguish between:
  - Form of the statement (e.g. : « *Does it rain ?* »)
  - Communicative function
  - Semantic content

BDI mental states can formalise those dimensions

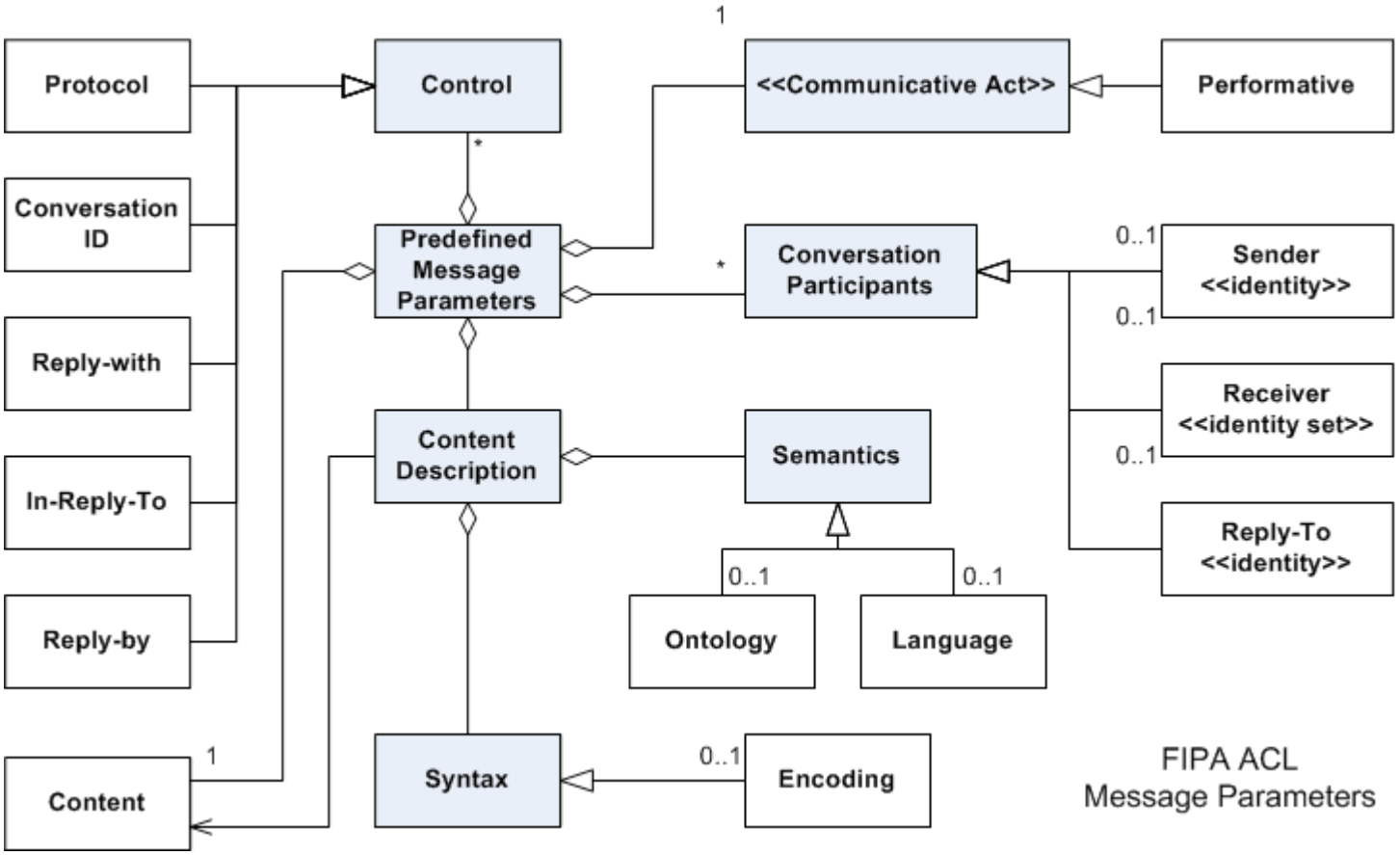
Examples

- Statement « *Does it rain ?* » associates communicative function « *closed question* » and proposal « *it rains* ». They add statement « *Does it rain ?* » to the linguistic context and add to the listener's beliefs that the speaker wants to know if proposal « *it rains* » is true.

Act	Example	Meaning
!	$!_x p$	Agent *x* confirms *p*.
?	$?_x p$	Agent *x* asked question *p*.
!?	$!?_x p$	Agent *x* confirms its ignorance about *p*.
$	$\$_x$	Agent *x* has nothing to say anymore (dialogue ends).

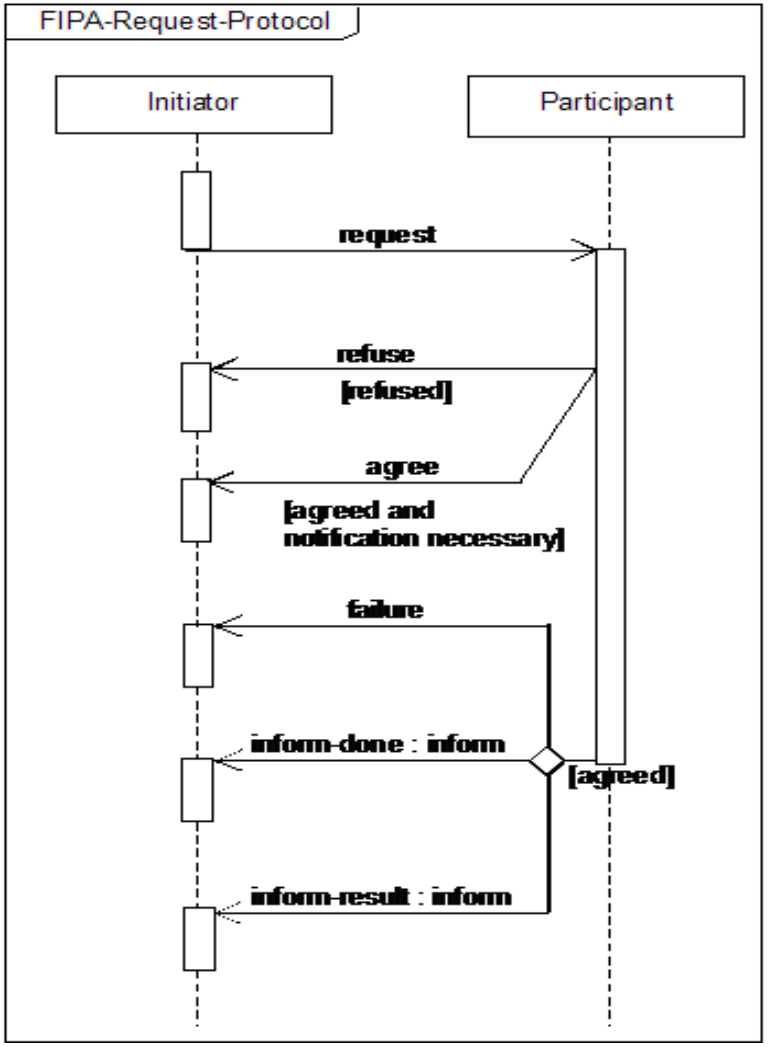
FIPA-ACL messages

Agent Communication Language



FIPA protocol

FIPA Request Interaction Protocol



FIPA-ACL examples

<i, inform(k,p)>

FP : $B_i p \wedge \neg B_i(B_k p \vee U_k p)$

RE : $B_k p$

Ex : Agent i informs agent j that (it is true that) it is raining today.

```
(inform
 :sender (agent-identifier :name i)
 :receiver (set (agent-identifier :name j))
 :content
  "weather (today, raining)"
 :language Prolog)
```

<i,query-if(j,X)>

FP : $\neg B_i X \wedge \neg B_i \neg X \wedge \neg U_i X \wedge \neg U_i \neg X$

RE : Done(<j,inform(i,X)> $\vee$ <j,inform(i,¬X)>)

To go further

Main conferences

- AAMAS (Autonomous Agents and Multi-Agent Systems)
- IAT (Intelligent Agent Technology)
- JFSMA (French-speaking days about Multi-Agents Systems)

Lessons

- J.P. Sansonnet (http://perso.limsi.fr/jps/)
- R. Courdier (http://personnel.univ-reunion.fr/courdier)