Artificial Intelligence Summer School

Multiagent Decision Making

Laurent.Vercouter@insa-rouen.fr

From Artificial Intelligence...

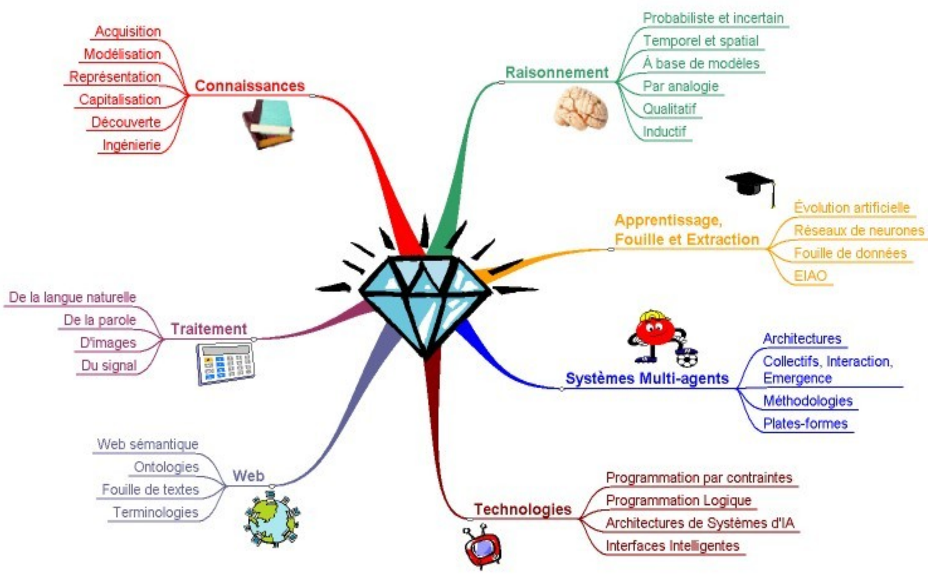
Creation of the field « Artificial Intelligence » at the Darmouth conference, 1956

2 approaches :

- Understanding and reproducing human intelligence
- Developping computer programs able to solve « complex problems »

AI today (def. AFIA)

« the construction of computer programs that perform tasks that are, for the moment, more satisfactorily accomplished by human beings, as they require high-level mental processes such as perceptual learning, memory organization and critical reasoning » (Marvin Minsky)



… to Distributed Artificial Intelligence...

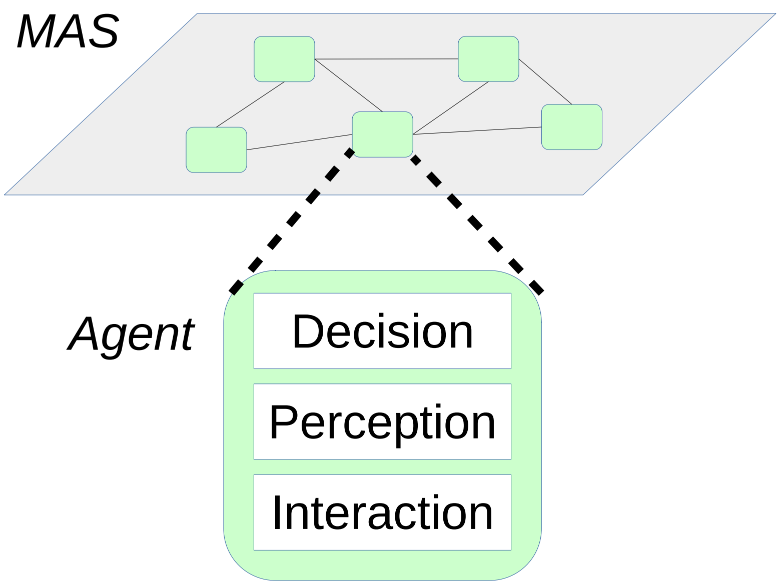
In the 90’s, the development of networks and multi-processor architectures paves the way for distributed and/or parallel systems.

Distributed Artificial Intelligence encompasses :

- Parallel A.I.
  - Improved computing performance
  - Languages for managing concurrency and parallelism
- Distributed problem solving
  - Problem decomposition for multiple solutions
  - Competing or complementary solutions
  - Need for synchronization of shared knowledge and partial solutions
- Multi-Agent Systems (MAS)

...to Multi-Agent Systems

A **MAS** is a set of agents that interact together and with a shared environment.



Macroscopic perspective: the MAS has a global behavior resulting from the actions and interactions of the agents that make it up.

Microscopic perspective: an agent is conceived as an intelligent and autonomous software entity.

- **Top-down** development: use expected macroscopic behavior as a starting point for developing agent models
- **Bottom-up** development: develop agent models and observe the resulting macroscopic effects

Agenda

- **Introduction to Multiagent systems**
  - Agent architectures
  - Main application domains
- Decision making
- Reactive agents
- Cognitive agents

Intelligent Agent (1)

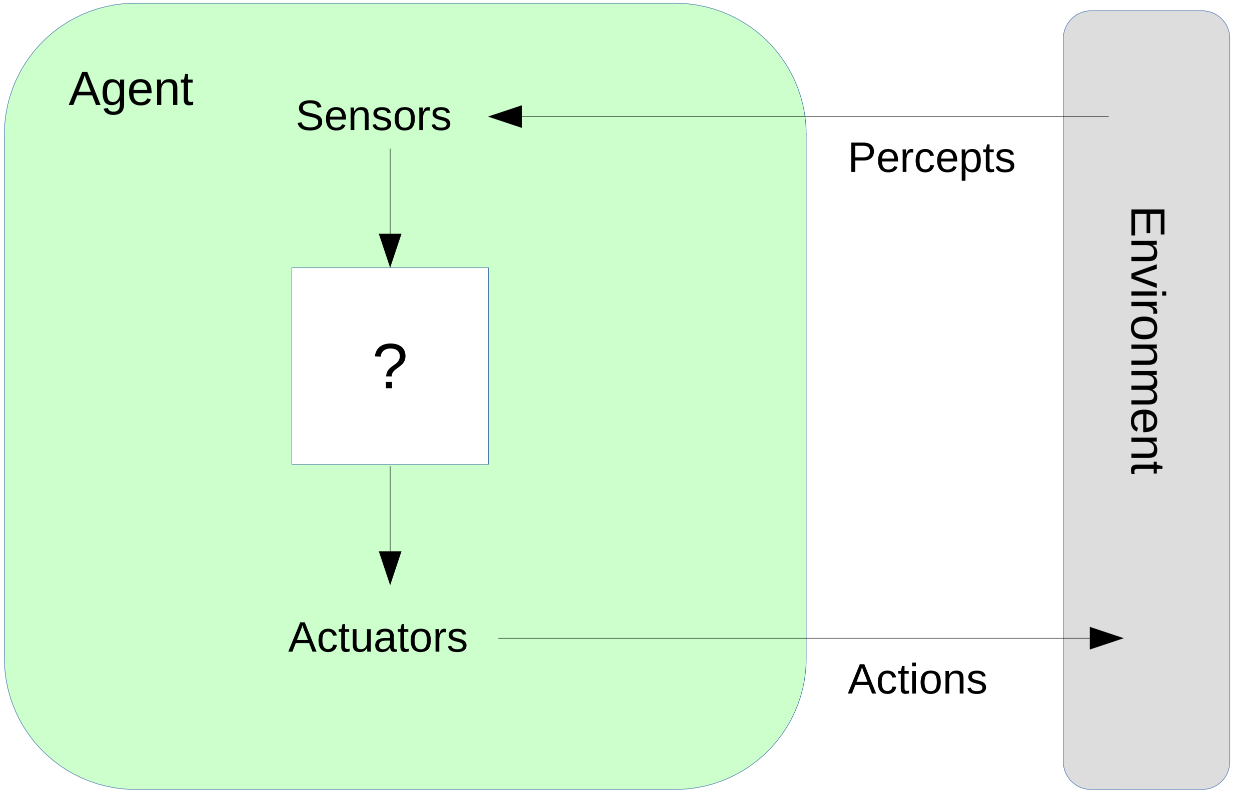
An agent is a physical or virtual entity (def. Ferber, 91) that

- is capable of acting in an environment,
- can communicate directly with other agents,
- is driven by a set of tendencies (in the form of individual objectives, etc.),
- has its own resources,
- is capable of perceiving its environment (albeit to a limited extent),
- possesses skills and offers services,
- ...

Intelligent Agent (2)

General scheme

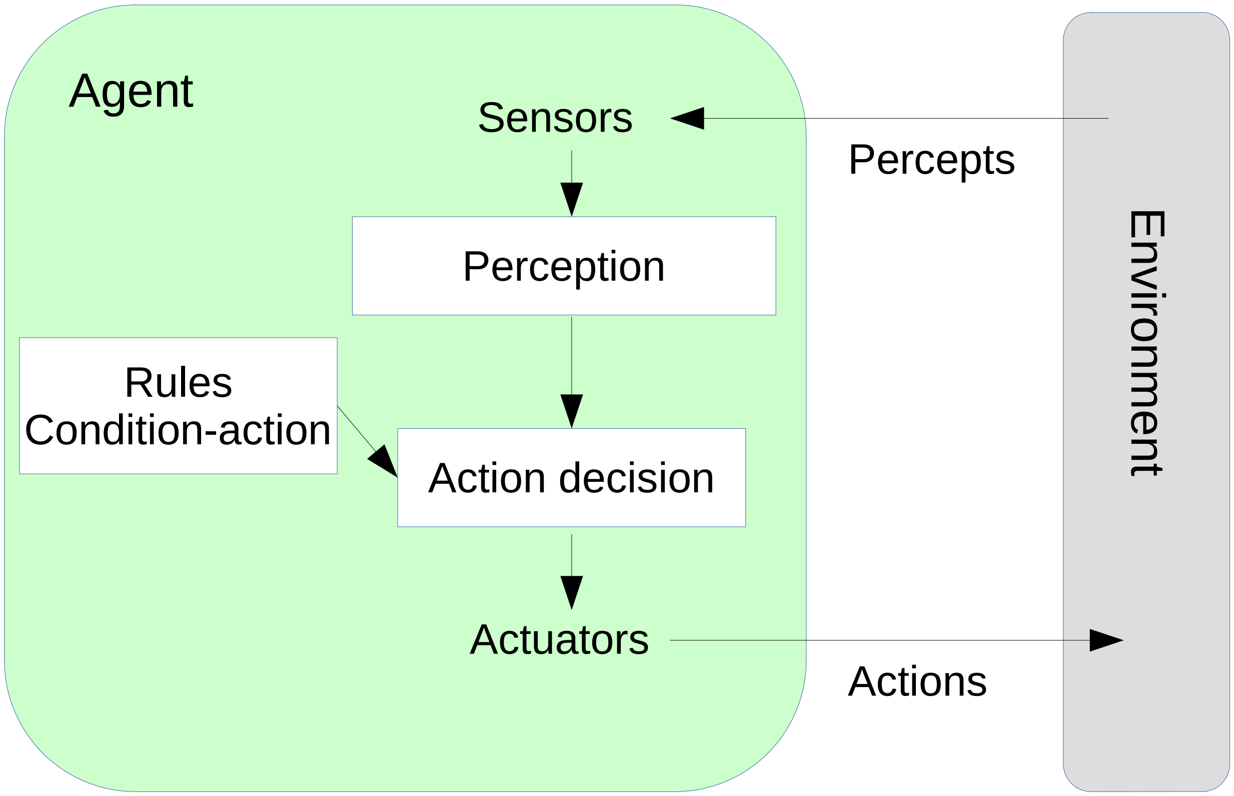
(source « Artificial Intelligence », Russel & Norvig, 3rd édition)



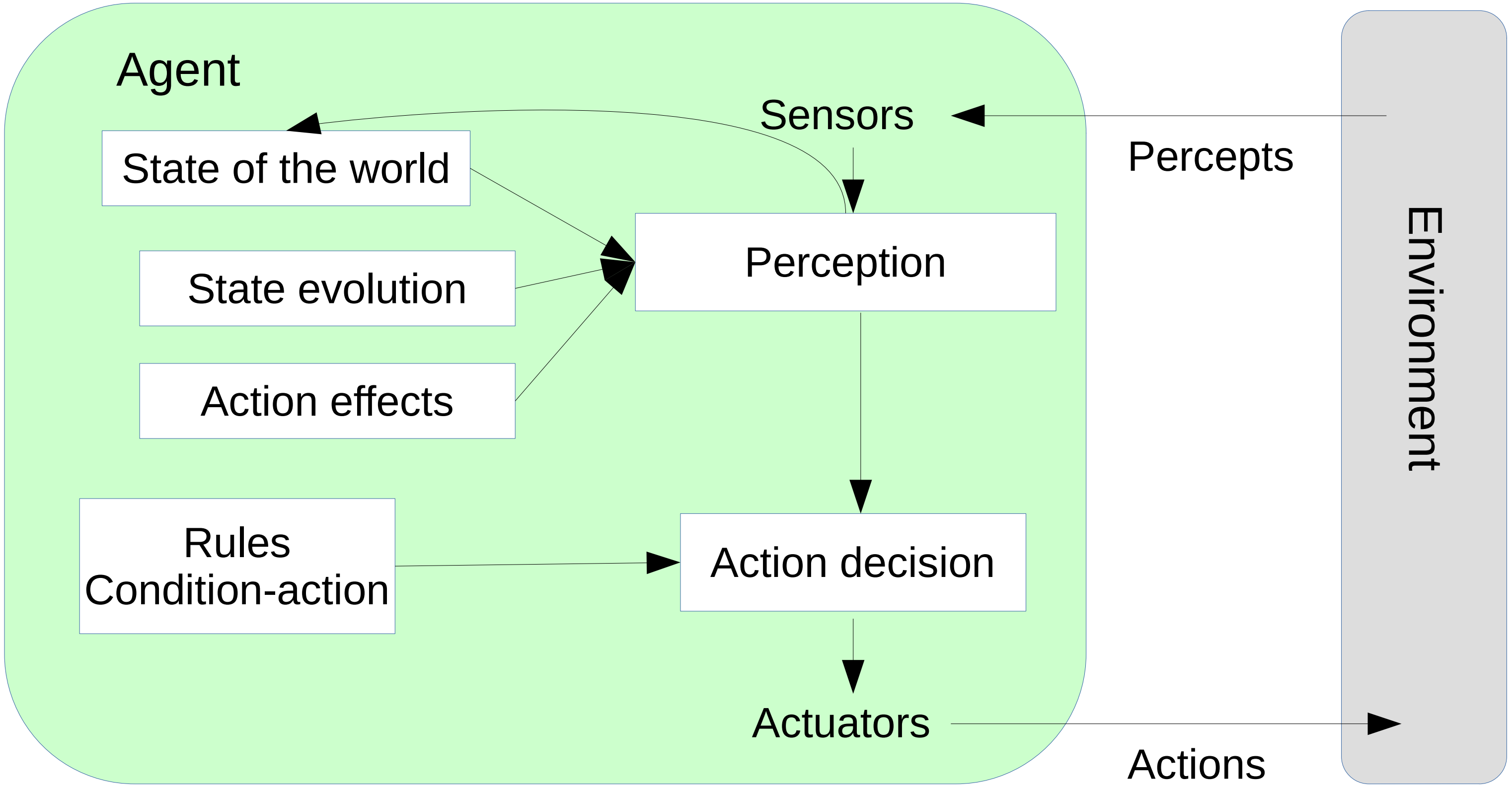
Simple reactive architecture

Example : thermostat reactive agent

- Perception: ***temperature*** = *X*
- Possible actions: ***heat***, ***turn off***
- Rules: if ***temperature*** *< 19 then* ***heat***, if ***temperature*** *≥ 19 then* ***turn off***



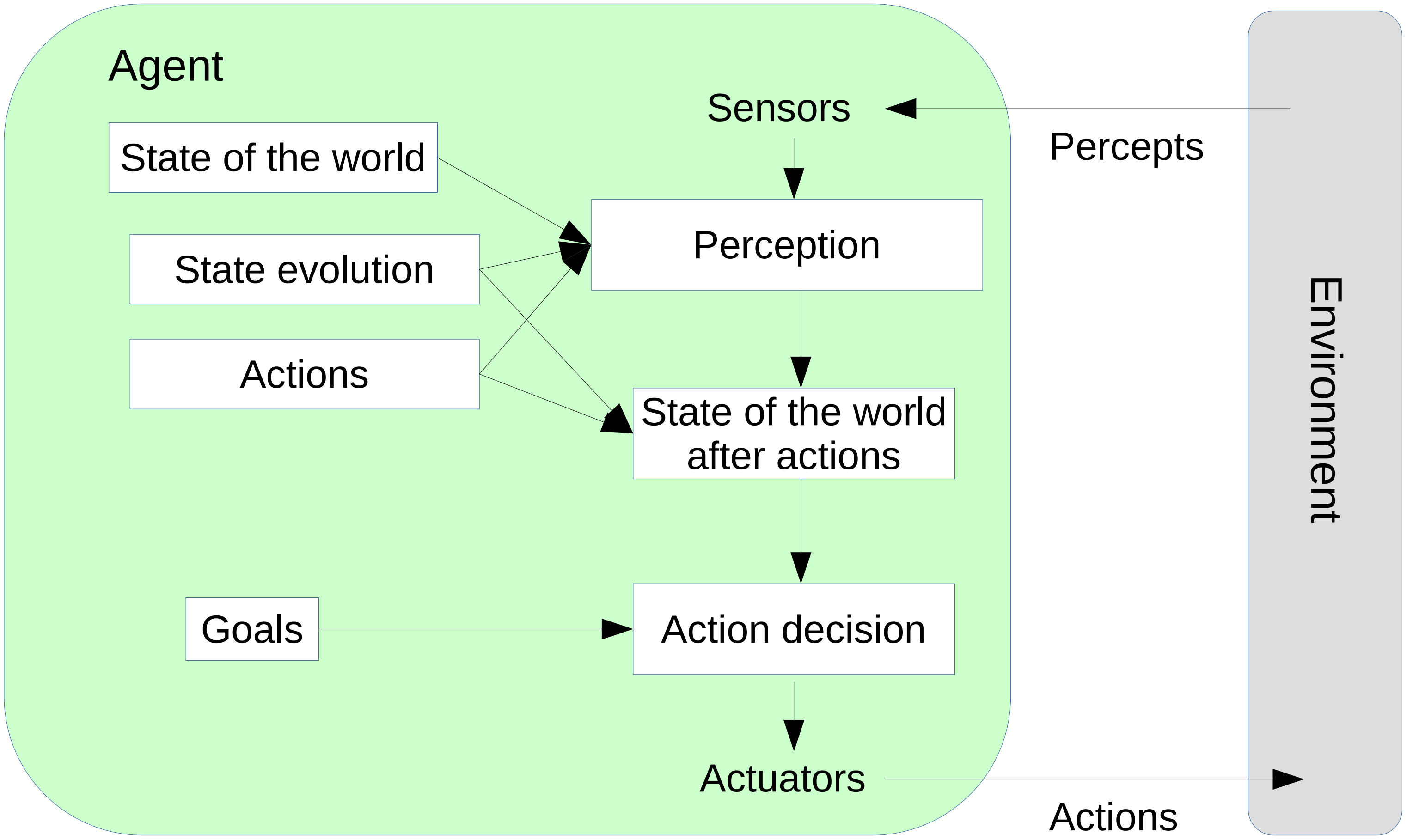
Reactive architecture with intern state



Cognitive architecture based on goals

Example : thermostat cognitive agent

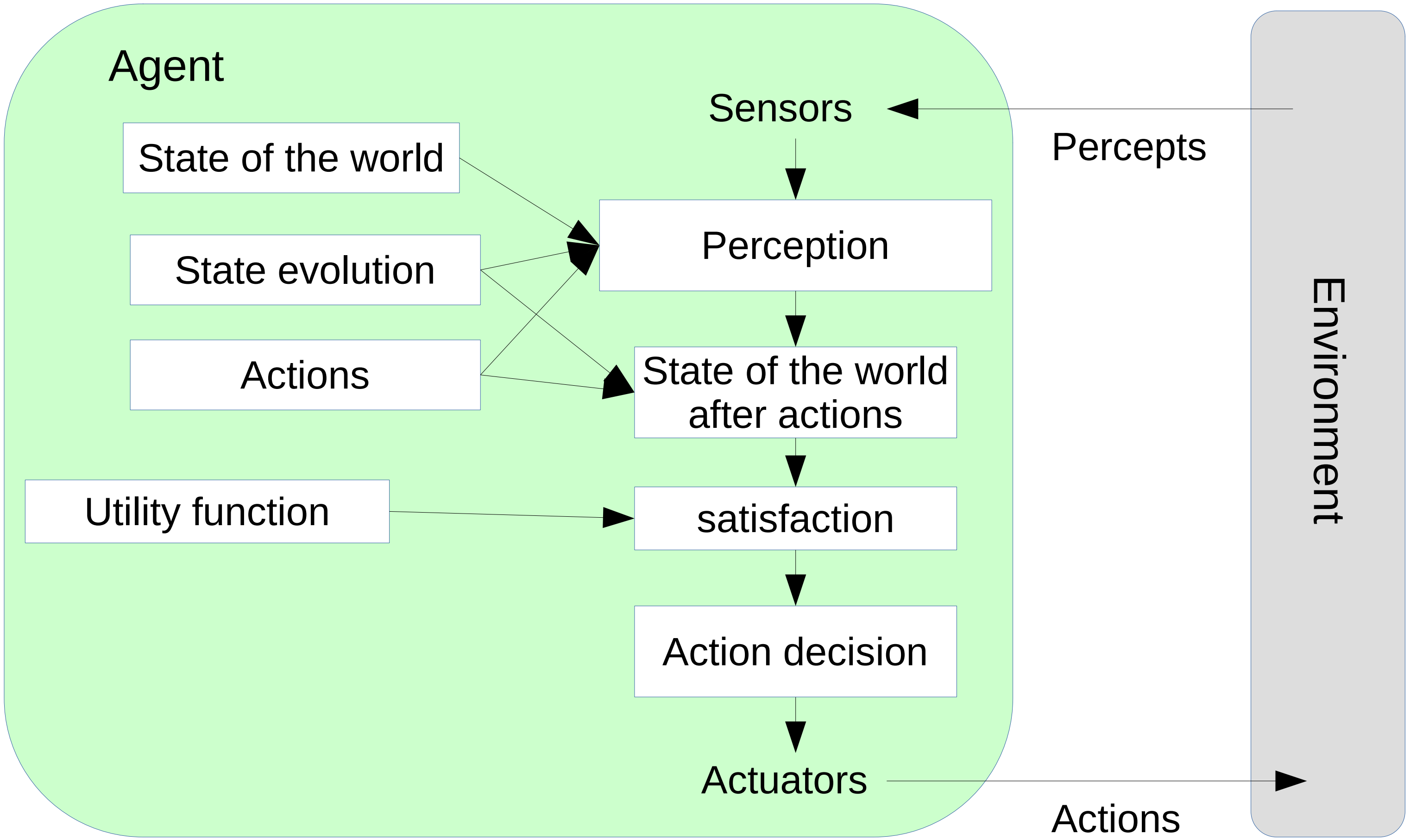
- State of the world: ***temperature*** = *X*, ***temperature increasing***
- Possible actions: ***heat*** *(effect = temperature++)*, ***turn off*** *(effect = Ø)*
- Goals: *temperature ≥ 19*



Cognitive architecture based on utility

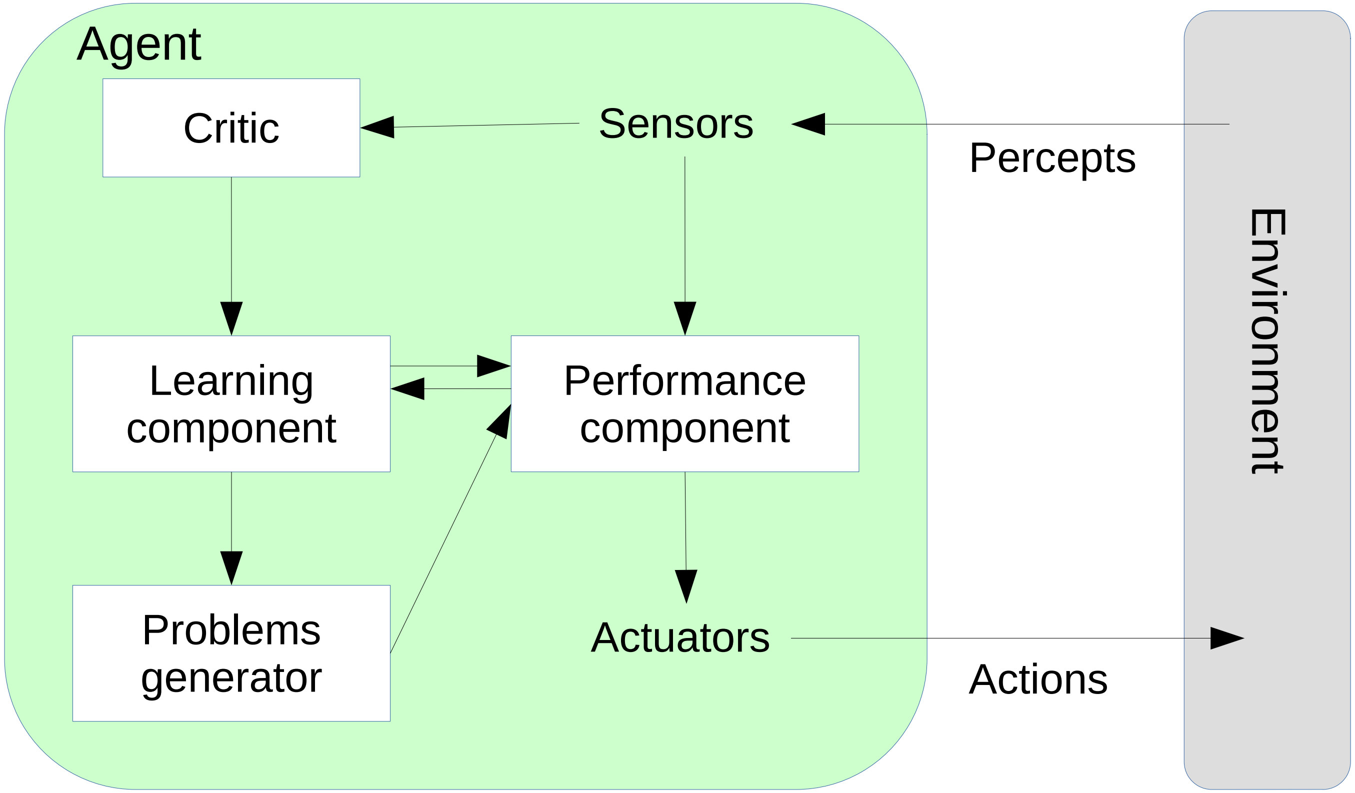
Example : thermostat cognitive agent

- State of the world: ***temperature*** = *X*
- Possible actions: ***heat*** *(effect = temperature++)*, ***turn off*** *(effect = Ø)*
- Utility function: *minimize(|temperature-21|)*



Learning agent

The performance component groups here the components architecture seen previously.

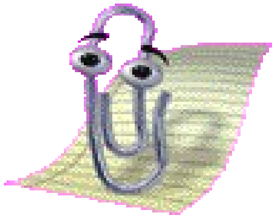


Examples of application of an autonomous agent

- Autonomous robots




- Software personal assistant




- Conversational agent
  - https://www.youtube.com/watch?v=K6kcv3zwoo8

Agent vs. Object

In many MAS programming languages, an agent is implemented as an object, but we are looking for the following properties:

- Autonomy
  - Internal: the agent's control mechanism is its own
  - External: no control over the behavior of other agents
- Pro-activity
  - Behavior directed by its own objectives (rules, goals, utility function, learning function)
- Sociability
  - Ability to interact with other agents

From one agent to a MAS

The co-existence of several agents in the same environment introduces a "social" dimension into the system.

- Dependencies between agents
  - Ex: A's actions can influence B's goals
- Interactions between agents
  - Ex: A communicates its goal to B to influence its behavior
- Social knowledge
  - Ex: A incorporates its perception of B's goals and knowledge into its own knowledge

Properties of a MAS

At the global level, we look for the following properties in the design of a MAS:

- System openness
  - During execution, agents can enter or leave the system
- Decentralized control
  - The control loop is implemented locally in each agent
- Agent heterogeneity
  - Agents with different motivations and functions can coexist

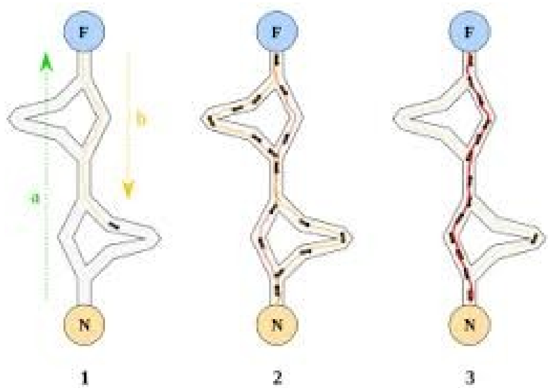
Application domains of MAS

- Distributed problem solving
  - Distributing AI algorithms
- Modeling and simulating autonomous entities
  - Reproduction or validation of individual and collective behavior models
- Distributed software applications
  - Service deployment in heterogeneous networks
- Interaction and communication with humans
  - Task delegation or assistance

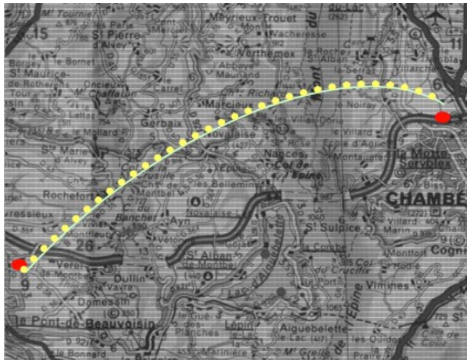
Distributed problem solving

- Examples of spatial problem solving

Path finding



Cartography



https://www.youtube.com/watch?v=8F6caMF7MXg

- Characteristics
  - Actions to be chosen according to a local representation of an agent's world
  - Sequence of coordinated interactions between agents
  - Actions constrained by a simulated environment and agent position

Distributed software application

- Examples from collective robotics

Robocup

Robocup rescue

- Characteristics
  - Actions to be chosen according to a local representation of an agent's world
  - Communications between agents may be limited in scope
  - Highly uncertain perceptions and effects of actions

Agent-based simulation

- Example from situated individual-centric simulation

Trafic simulation

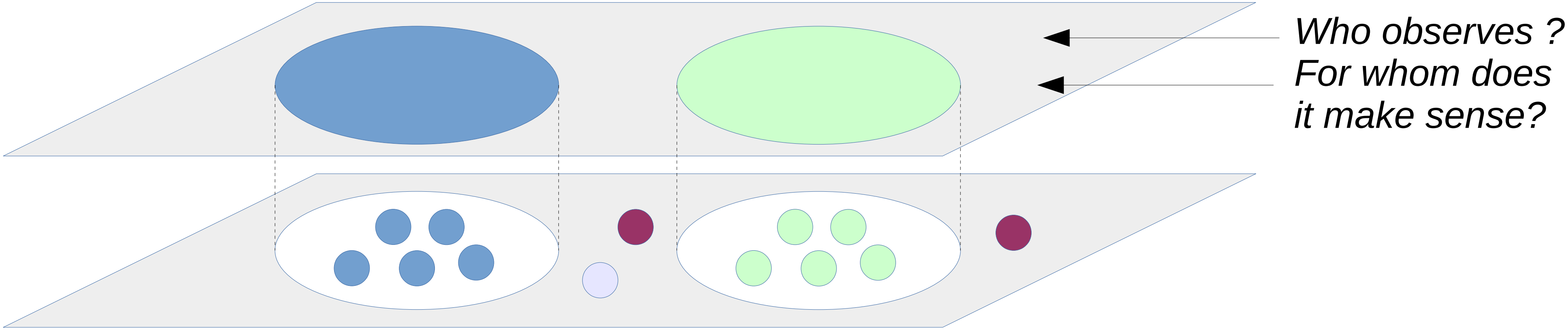
MASSIVE

- Characteristics
  - Actions to be chosen according to a local representation of an agent's world
  - Sequence of coordinated interactions between agents
  - Actions constrained by a simulated environment and agent position

Emergence and collective intelligence

Emergence

- In a hierarchical system of increasing complexity, a phenomenon or entity that originates at an earlier level is called emergent



Collective intelligence

- Inspiration from natural social systems (insect colonies, flocking, etc.)
- No overall coordination or supervision
- The fundamental principle of C.I. is that entities can produce functional behavioral patterns through direct or indirect interaction.

Agenda

- Introduction to Multiagent systems
- **Decision making**
  - Rationality
  - Basics of logic programming
- Reactive agents
- Cognitive agents

Rationality

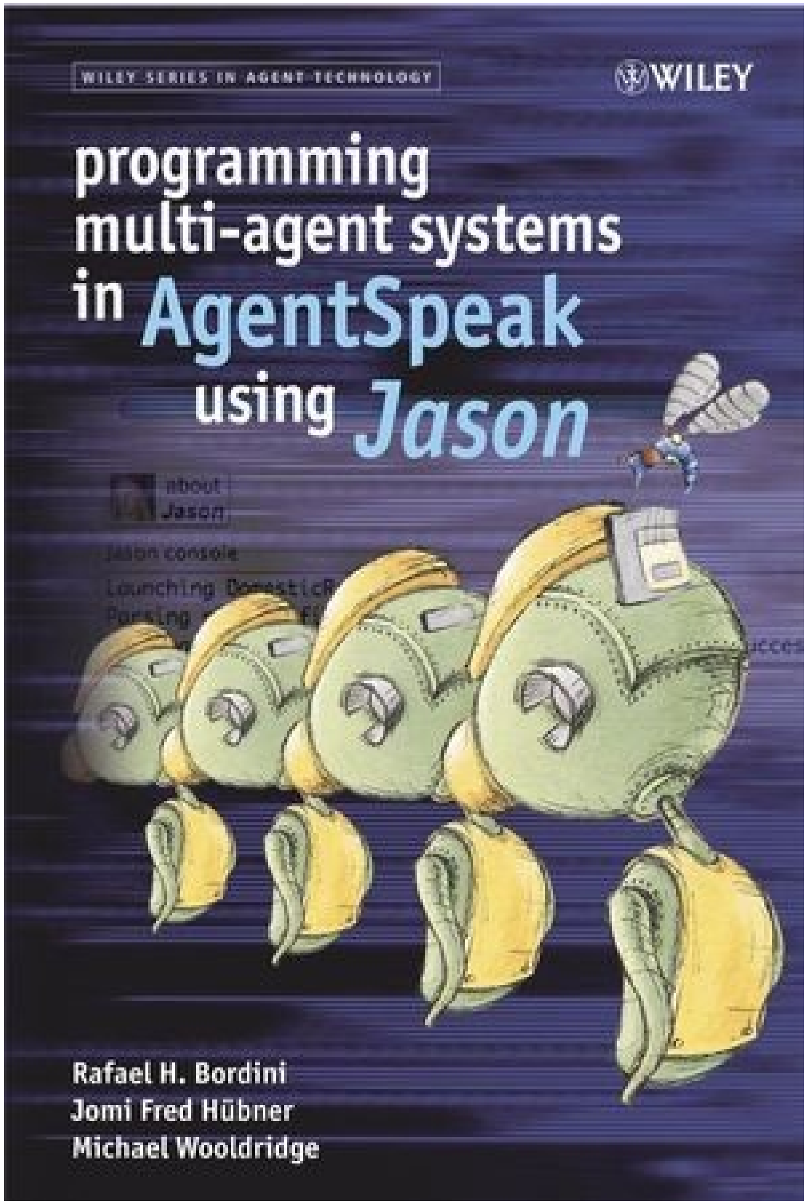
- Rational thinking according to Aristotle (syllogism)
- In the 19th century, logicians defined formal statements at the intersection of philosophy and mathematics
- Rational action
  - Automatically defining the right action to perform
  - "right action" → existence of a goal or function to be optimized
- Limited rationality (H. Simon) = taking into account limited capacities or information

Automatic rational decision making

For software agents the « good » action to perform is defined by

- associating actions and situations
  - Reactive agents
- Defining goals and actions effects
  - Logical cognitive agents
- Defining several possible effects of agents
  - Probabilistic cognitive agents

A langage for developing rational agents : Jason



- Interpreter fo an extended version of the AgentSpeak language
- Paradigm of logic programming
- One of the most used platform to develop cognitive agents
- Downloadable in open source : http://jason.sourceforge.net/

Basics of first-order logic

In first-order logic, terms are made up of **variables** (whose symbol begins with a capital letter) and **functions** (in lower case).

- A variable is a term
- A function of arity n is a term, *f(t1, ..., tn)* with *t1, ..., tn* being terms
- An arity 0 term is a constant, e.g. f

Example (to be used when simulating a vacuum cleaner robot)

- *pos(X)* a pos function designates the robot's position X
- *dirty* a constant indicating that the floor is dirty

Horn clauses

An Horn clause is written

$$A \text{ :- } B_1, \ldots, B_n$$

with A, $B_1, \ldots, B_n$ being terms

An Horn clause specifies that if $B_1, \ldots, B_n$ are true then A is true.

If n=0 the Horn clause is a fact (always true), otherwise it is a rule.

Example

- *dirty :- pos(X), dirty(X)*

Logic programming

A logic program is a sequence of Horn clauses

For logical deduction or constraint satisfaction

- Expression of domain semantics by rules and facts
- Expression of an initial knowledge base by facts
- Deduction of new facts by logical inference

For rational decision-making

- Same possibilities for inference from facts
- Expression in the form of rules for actions to be taken when certain facts are true
- Updating a knowledge base by perceiving the effect of actions

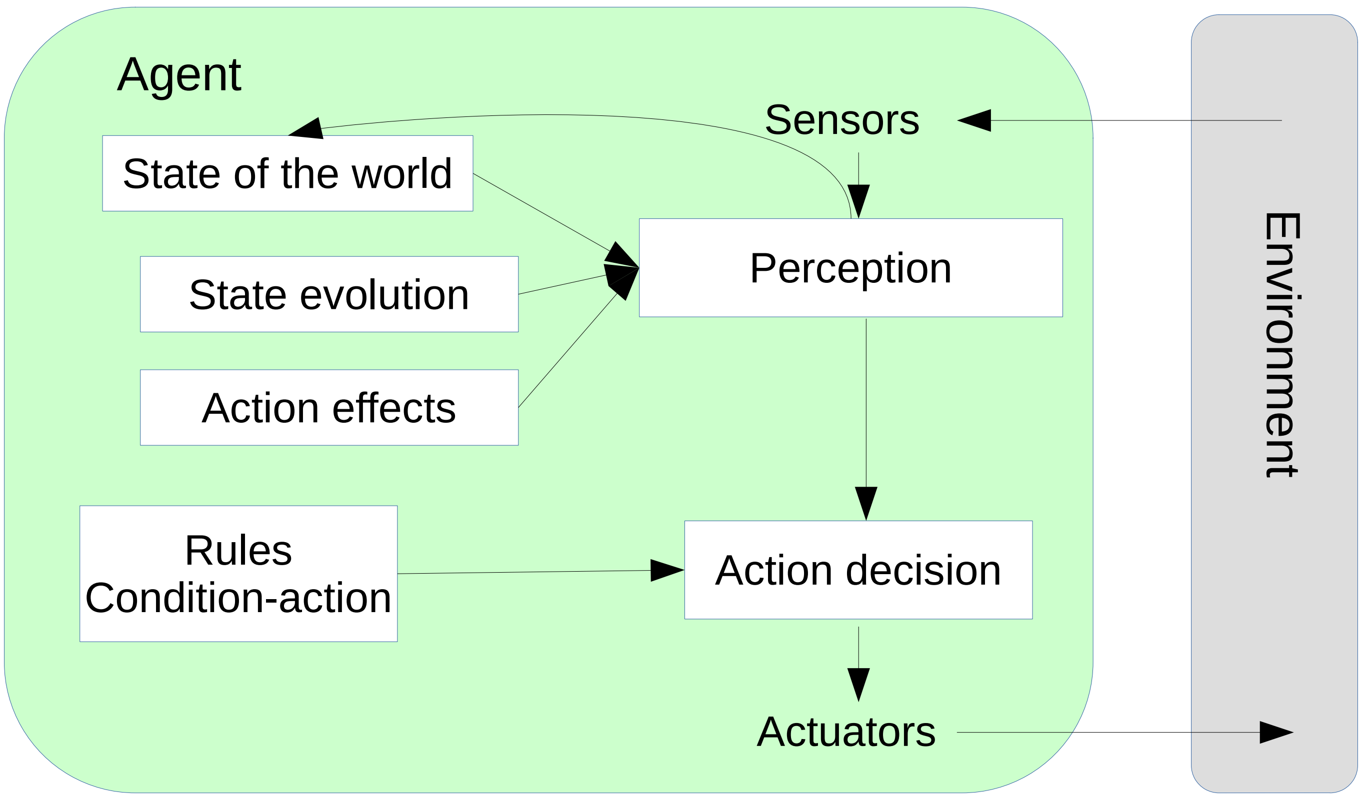
Example (written in Jason)

+*dirty* <- *suck* if fact *dirty* is true, action *suck* is rational

Agenda

- Introduction to Multiagent systems
- Decision making
- **Reactive agents**
  - General architecture
  - Programming reactive agents in Jason
- Cognitive agents

Jason reactive architecture



- Closed control loop: stimulus → response
- The environnement directly leads agents' behaviors
- No explicit world representaion
- No history

Reactive control function

```
do {
    percepts:= see() ;
    state:= interpret_inputs(percepts) ;
    rule:= match(state,rules) ;
    execute(rule[action]) ;
} while (1) ;
```

Jason : Beliefs

Beliefs are represented by first-order logic literals

- *literal(term$_1$, …, term$_n$) [annot$_1$, …, annot$_m$]*

Examples

- *red(box1)[source(percept)].*
- *friend(bob,alice)[source(bob)].*
- *~dirty[source(self)]*

Jason : Beliefs dynamics

The source annotation is created automatically according to the origin of the belief

- Intention to add a belief : the + and – operators add/remove a belief coming from the agent itself
  - +dirty ; // adds dirty[source(self)]
  - -dirty ; // removes dirty[source(self)]
- By communication : effect of the send action
  - .send(tom,tell,dirty) ; // sent by bob
  - // adds dirty[source(bob)] to tom's beliefs
  - .send(tom,untell,dirty) ; // sent by bob
  - // removes dirty[source(bob)] from tom's beliefs

Jason : Description of a plan

An AgentSpeak plan has the following structure:

triggering_event : context $\leftarrow$ body.

- triggering_event refers to the event that the plan has to consider
- context defines activation conditions
- body is the sequence of actions to perform

Jason : Triggering events

Triggering event can be from the following cases

- +b (belief addition)
- -b (belief removal)
- +!g (goal addition)
- -!g (goal removal)
- +?g (goal addition)
- -?g (goal removal)

Jason : Operators for context

Boolean operators

- & (and)
- | (or)
- not (not)
- = (unification)
- >, >= (relational)
- <, <= (relational)
- == (equals)
- \ == (different)

Arithmetic operators

- + (sum)
- - (subtraction)
- * (multiply)
- / (divide)
- div (divide – integer)
- mod (remainder)
- ** (power)

Jason : Negations

- Weak negation (not) = lack of belief
- Strong negation (~) = belief that a fact is false

```
+pos(X) : ~dirty
    <- move.

+pos(X) : not dirty & not ~dirty
    <- see(X).
```

Exercice

Write Jason plans to control a vacuum cleaner robot in a simulated environment with 4 squares.

Beliefs are automatically created by percepts

- *dirty* the robot square is dirty
- *clean* the robot square is clean
- *pos(X)* with X = 1, 2, 3 or 4 the robot is on the square X

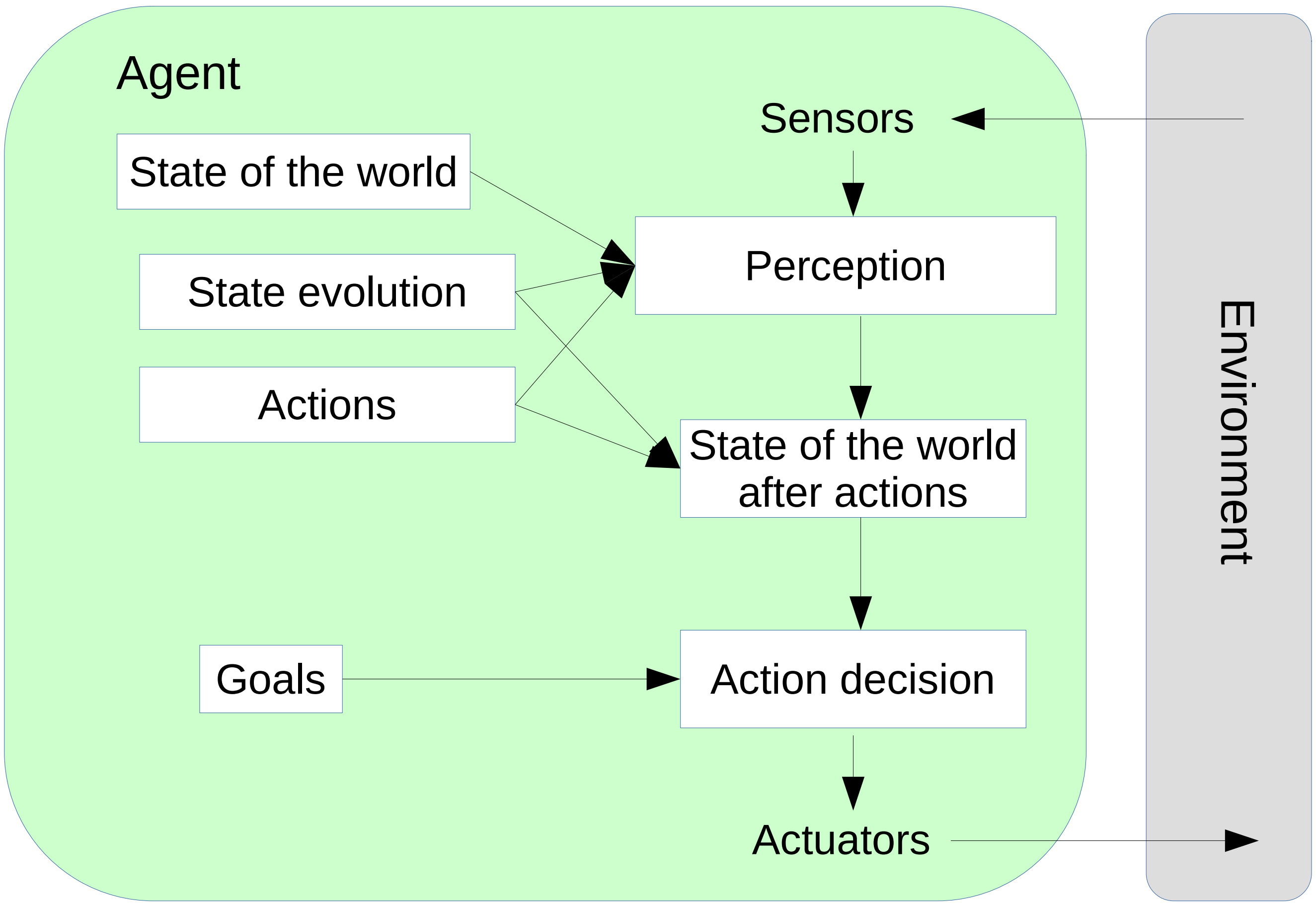
Possible actions

- *suck* the robot cleans the square
- *left* the robot moves left
- *right* the robot moves right
- *up* the robot moves up
- *down* the robot moves down

Agenda

- Introduction to Multiagent systems
- Decision making
- Reactive agents
- **Cognitive agents**
  - General architecture
  - Programming cognitive agents in Jason

Cognitive architecture



- Deliberative process between perception and action stages
- Explicit representation of the world
- Representation of past beliefs (history)

Cognitive control function

```
s : state
eq : event queue

s := initialize() ;
do {
    options:= option_generator(eq,s);
    selected:= deliberate(options, s) ;
    s := update_state(selected, s) ;
    execute(s) ;
    eq:= see() ;
} while (1) ;
```

BDI model for cognitive agents

- BDI agent [Rao, Georgeff 91], according to this model an agent has :
  - **Beliefs** about itself and about the world,
  - **Desires** that may be contradictory,
  - A set of **Intentions** that are not in conflict,
  - A reasoning mechanism to update its beliefs, to choose a desire and to generate new intentions.

BDI langages and architectures

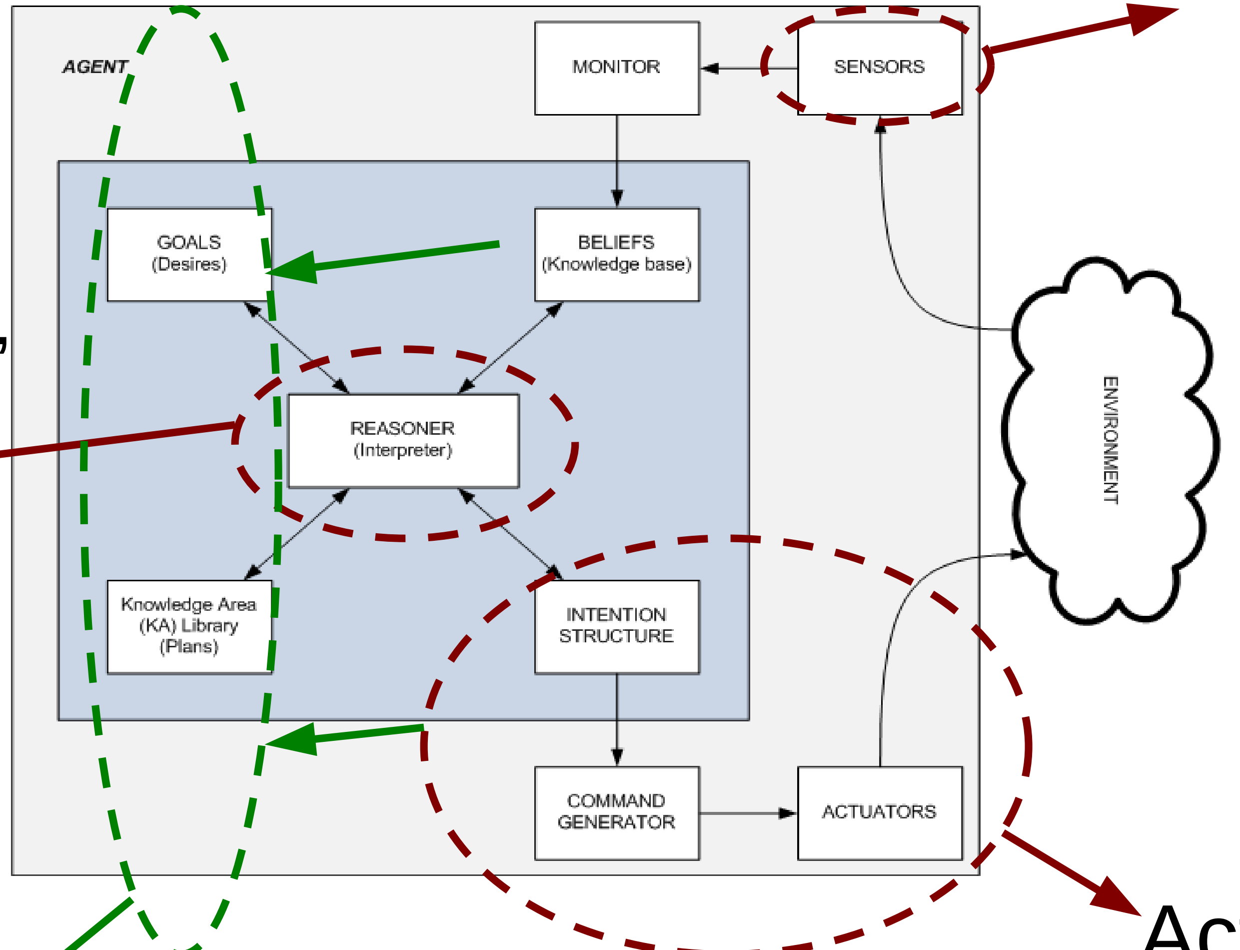
- BDI formal model is a reference for the development of several associated agents programming languages
  - AgentSpeak, dMars, Jack, Jadex, Jason, …
  - Each of them complies (sometimes partially) to the BDI logics

    an interpreter executes agents reasoning according to the BDI approach
- The architecture of a BDI agent defines
  - Its software components,
  - Their control processes,
  - Their data exchange processes,
  - The exchanges with entities external to the agent.

Example : Procedural Reasoning System (PRS)

Perceptions

An ”intelligent” program

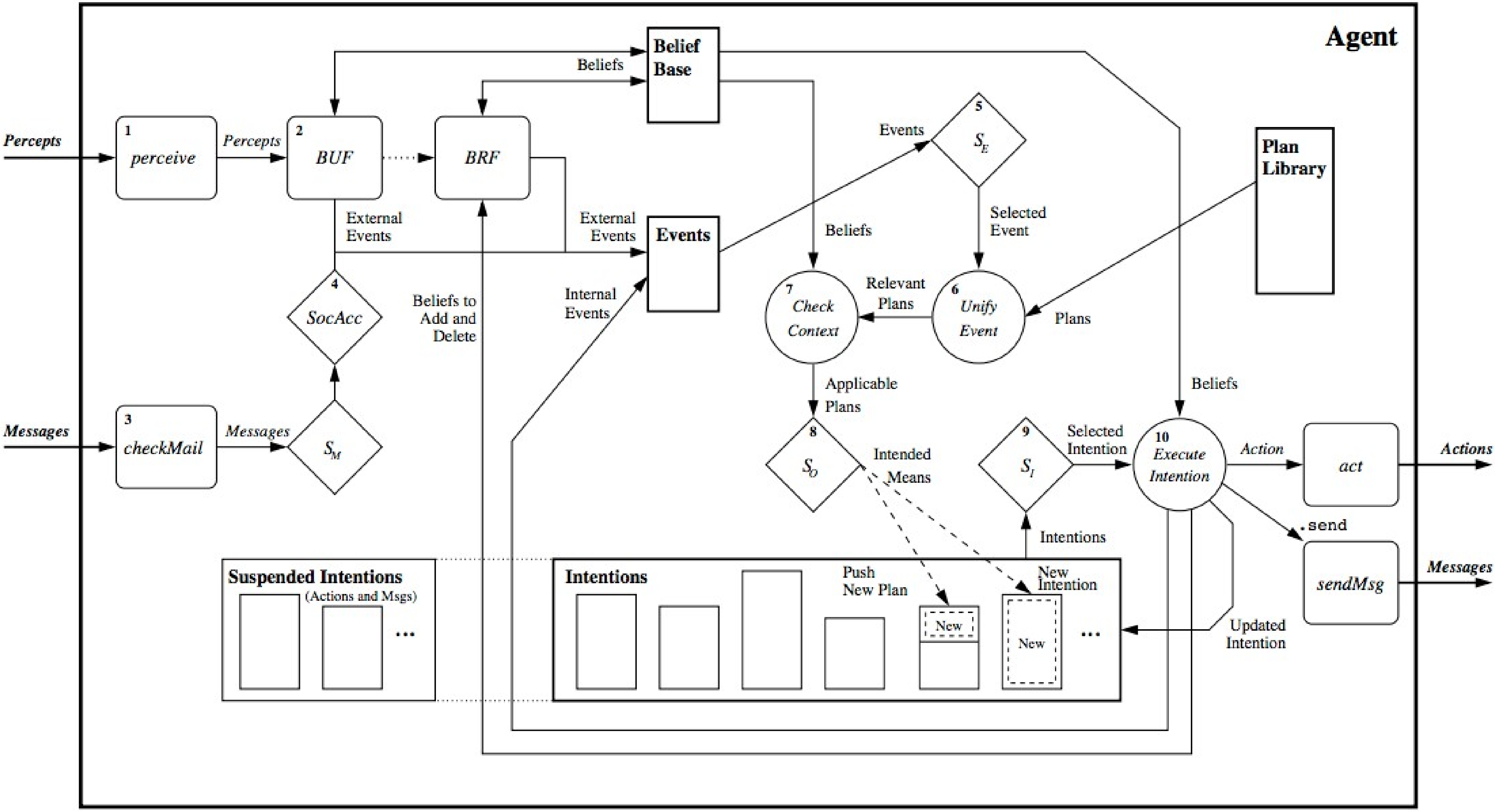


Actions

goals and plans

(inherited from STRIPS)

The Jason reasoning cycle



Main elements of the language

Beliefs: represent the information available to an agent

Goals: represent the states the agent wants to achieve

Plans: represent the agent's "know-how", i.e. the means to achieve goals.

Events: represent changes in the agent's beliefs or goals

Intentions: instantiated plans for achieving goals

Main components of the architecture

Belief base: where beliefs are stored

Event set: where events that the agent must handle are stored

Plan library: where plans known to the agent are stored

Intention set: where the actions the agent must perform as part of a plan to achieve a goal are stored

Jason interpreter

- Perceives the environment and updates the belief base
- Processes new messages
- Selects events
- Selects relevant plans
- Selects applicable plans
- Creates/updates intentions
- Select intention to execute

Goals

- Achievement goal : a task that has to be accomplished
  - !goal
- Test goal : a goal to know an information
  - ?goal
- Examples
  - !write(book) ; // adds the goal !write(book)[source(self)]
  - ?publisher(P); // adds the goal ?publisher(P)[source(self)]

Goals dynamics (1)

The source annotation is automatically created according t othe origin of the goal

- Intention to add a goal : the plan operators ! and ? adds/removes a goal coming from the agent itself

- By communication : effect of the send action for achievement goals
  - .send(tom,achieve,write(book)) ; // sent by bob
  - // adds write(book)[source(bob)] to tom’s goals
  - .send(tom,unachieve,write(book)) ; // sent by bob
  - // removes write(book)[source(bob)] from tom’s goals

Goals dynamics (2)

- By communication : effect of the send action for test goals
  - .send(tom,askOne,published(P),answer) ; // sent by bob
  - // adds ?published(P)[source(bob)] to tom's goals
  - // the response will use answer
  - .send(tom,askAll,published(P),answer) ; // sent by bob

Body of a plan

The body of a plan can contain

- Operators on beliefs + - -+
- Operators on goals ! ? !!
- Actions

```
+rain : time to leave(T) & clock.now(H) & H >= T
   <- !g1; // new sub-goal
      !!g2; // new goal
      ?b(X); // new test goal
      +b1(T-H); // add belief
      -b2(T-H);// remove belief
      -+b3(T*H);// update belief
      close(door).// external action
```

Plans examples

```
+green_patch(Rock)[source(percept)]
: not battery charge(low)
<- ?location(Rock,Coordinates);
     !at(Coordinates);
     !examine(Rock).

+!at(Coords)
: not at(Coords) & safe path(Coords)
<- move towards(Coords);
     !at(Coords).

+!at(Coords)
: not at(Coords) & not safe path(Coords)
<- …

+!at(Coords) :  at(Coords).
```

Plans dynamics

The plan library is composed by

- plans initialy defined by the developer
- plans dynamically modified by the actions
  - .add_plan
  - .remove_plan
- plans communicated by messages
  - tellHow
  - untellHow

Jason library

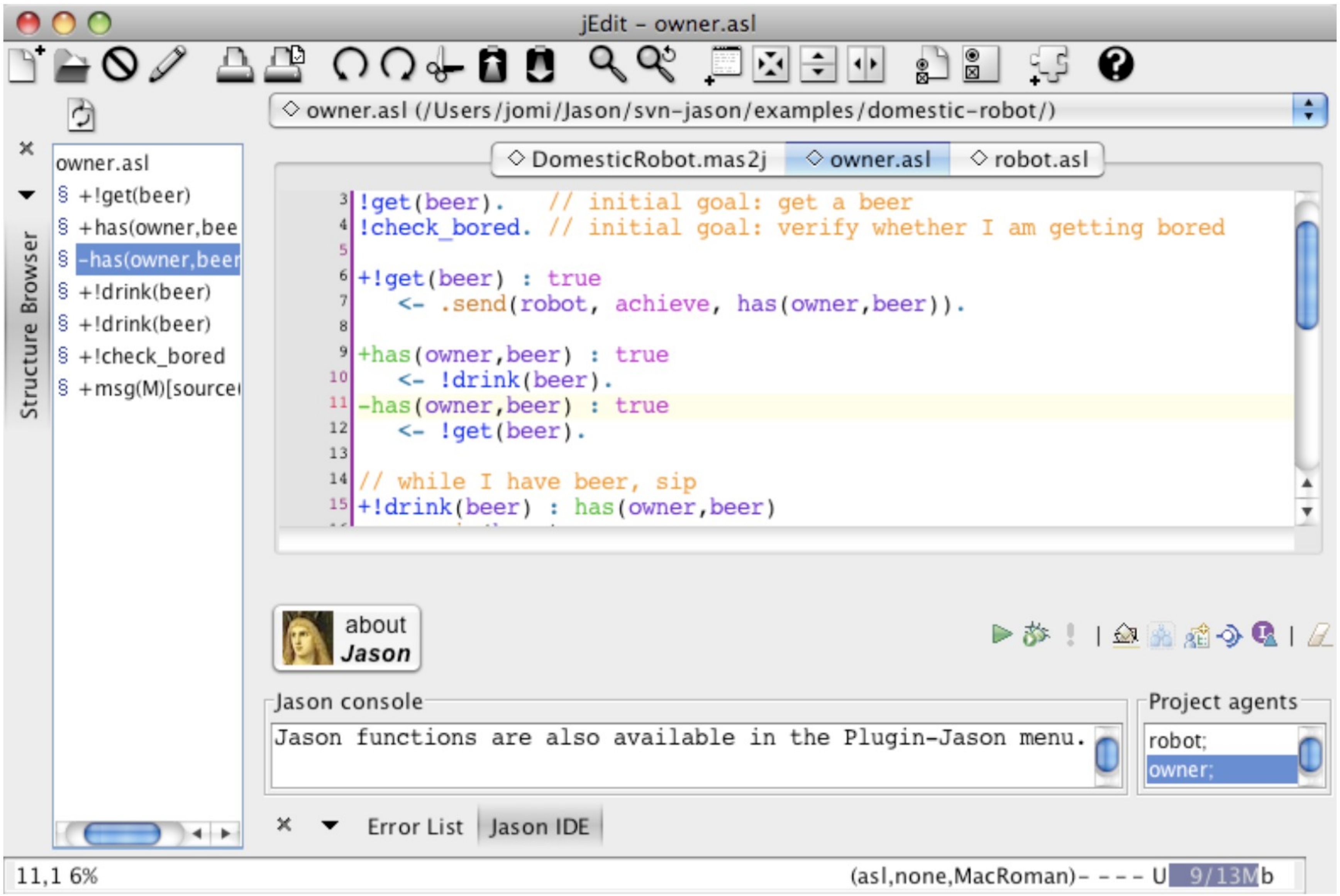
- Java library
  - Agent class extendable and able to interpret the Jason AgentSpeak language
  - Environment class to extend its own simulated environment
  - Multiagent communication nfrastructure
- Definition of the MAS to execute in a configuration file

Jason configuration file

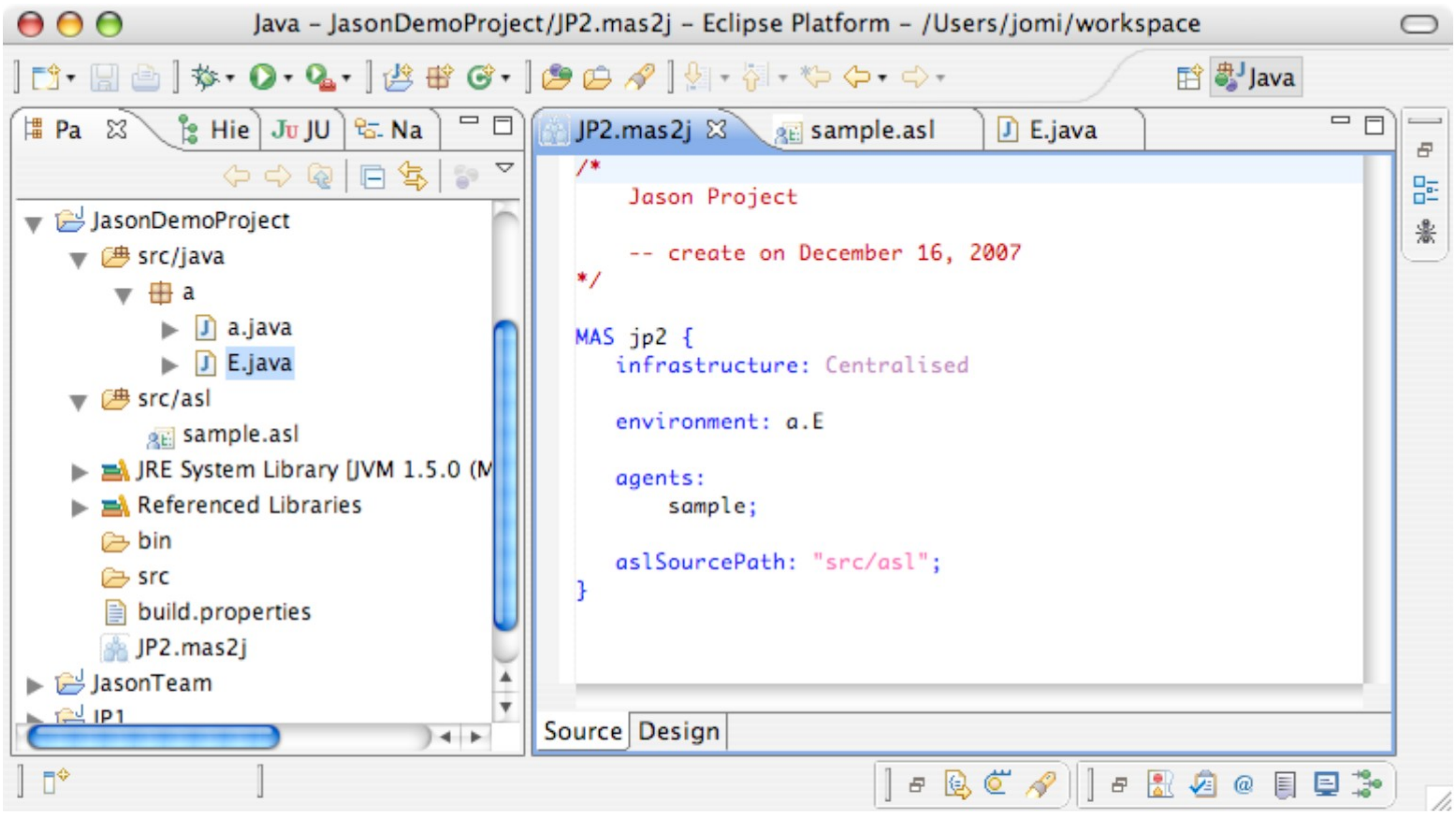
Example of configuration :

```
MAS my_system {
   infrastructure: Jade
   environment: robotEnv
   agents:
        c3po;
        r2d2 at jason.sourceforge.net;
        bob #10; // 10 instances of bob
   classpath: ”../lib/graph.jar”;
}
```

Plugin JEdit



Plugin Eclipse



Mind inspector

Sources et liens

Livres de références

- Multiagent systems, G. Weiss, MIT Press, 2nd edition
- Multiagent Systems, M. Wooldrdige, 2nd edition
- Les Systèmes Multi-Agents, J. Ferber
- Intelligence Artificielle, Russel & Norvig, 3ème édition

Cours accessibles en ligne

- O. Boissier (Ecole des Mines de Saint-Etienne) http://www.emse.fr/~boissier/enseignement/maop14/index.html
- R. Courdier (université Réunion) http://lim.univ-reunion.fr/staff/courdier/cours/index.html
- J.-P. Sansonnet (université Orsay) http://perso.limsi.fr/jps/
- ...