# Multi-Agent Systems : some definitions

**Wassila Ouerdane & Nicolas Sabouret**

CentraleSupélec

28 décembre 2018

# Contents

# Contents

# MAS in Software Engineering

## From Functions to Agents

- In the 50's, computing relied on function calls
- In the 70's, object-oriented programming
  - Variables and methods are associated with data structures
- In the 90's, service-oriented programming
  - Objects's interface made accessible (*e.g.* WSDL)

# MAS in Software Engineering

## From Functions to Agents

- In the 50's, computing relied on function calls
- In the 70's, object-oriented programming
  - Variables and methods are associated with data structures
- In the 90's, service-oriented programming
  - Objects's interface made accessible (*e.g.* WSDL)

## Agents

- Agents encapsulate data
- Agents interact using structured messages
- Interaction follow well-designed protocols
- Agents are situated in an environment
- Actions are structured with preconditions and effects

# MAS in Distributed Computing

## From Processes to Agents

- In the 60's, multiple processes on a single system
- In the 80's, limited distribution with client-server architectures
- In the 90's, distributed computing : any host can act as a server
    - System's reliability depend on service availability

# MAS in Distributed Computing

## From Processes to Agents

- In the 60's, multiple processes on a single system
- In the 80's, limited distribution with client-server architectures
- In the 90's, distributed computing : any host can act as a server
  - System's reliability depend on service availability

## Multi-Agent Systems

- Agents as asynchronous entities
- Mechanisms to handle :
  - Asynchronous and distributed communication ;
  - Concurrency (shared ressources with no deadlock nor starving) ;
- Replication : different agents can offer the same service
- Service section and transparent load balancing

# MAS in Artificial Ingelligence

## From Problem Solver to Agents

- In the 60's, heuristic search, gradient descent
- In the 90's, distributed search
  (ant algorithm for TSP, monte-carlo tree search, …)
  - Solution emerges from multiple simple explorations
- In the 80's, knowledge representation and reasoning
  - Logics for actions and changes
  - Automated planning

# MAS in Artificial Ingelligence

## From Problem Solver to Agents

- In the 60's, heuristic search, gradient descent
- In the 90's, distributed search
  (ant algorithm for TSP, monte-carlo tree search, ...)
  - Solution emerges from multiple simple explorations
- In the 80's, knowledge representation and reasoning
  - Logics for actions and changes
  - Automated planning

## Multi-Agent Systems

- Agents reason about information from others
- Agents plan and act accordingly
- Multi-Agent simulation for complex system study

# A definition ?

## AgentLink's roadmap (2004)

Not one unique definition → `http://www.agentlink.org/roadmap/`

# A definition ?

## AgentLink's roadmap (2004)

Not one unique definition → `http://www.agentlink.org/roadmap/`

## Agents

- Process (or any sort of scheduling)
- Encapsulation of data and well-structured actions
- "Reasoning" about actions (or any action selection mechanism based on observations)
- Asynchronous interaction mechanism

## MultiAgent Systems

- Open systems within a physical and logical environement
- Logical or physical distribution of computing
- Coordination mechanism between agents (including protocols)
- User in the loop !

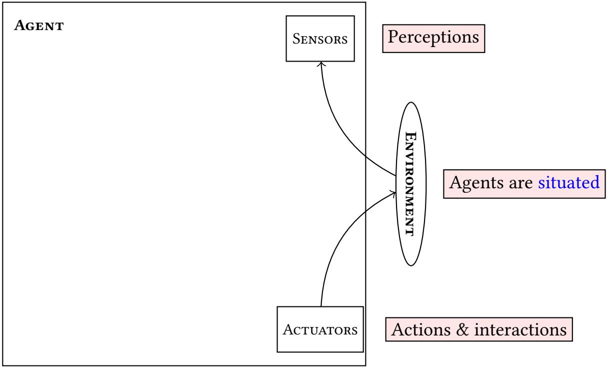
# Contents

# Procedural Reasoning System (PRS)

*(Georgeff, 1993)*

# Procedural Reasoning System (PRS)

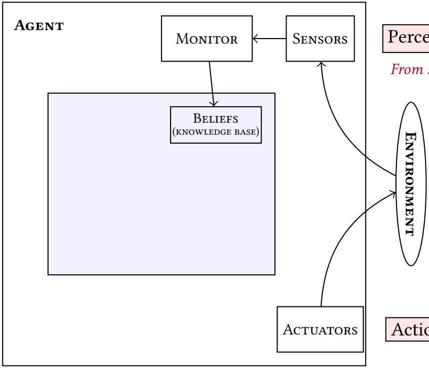
*(Georgeff, 1993)*



# Procedural Reasoning System (PRS)

*(Georgeff, 1993)*

Beliefs




Perceptions

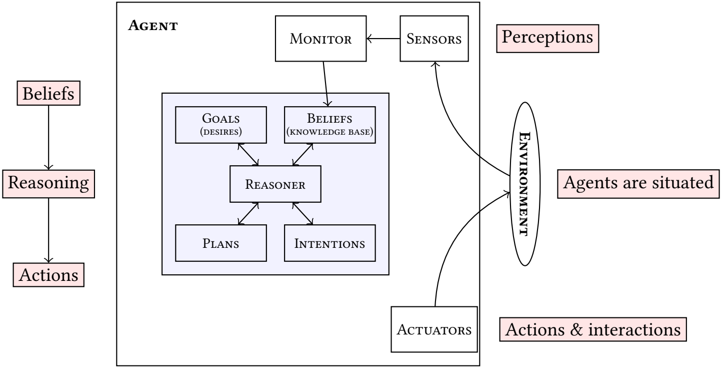
*From signal to knowledge…*

Agents are situated

Actions & interactions

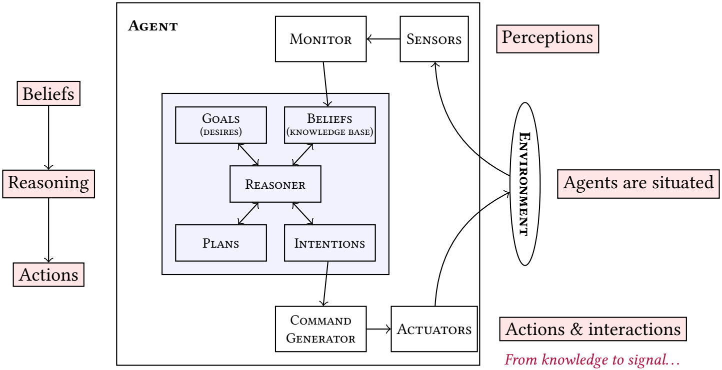
# Procedural Reasoning System (PRS)

*(Georgeff, 1993)*



# Procedural Reasoning System (PRS)

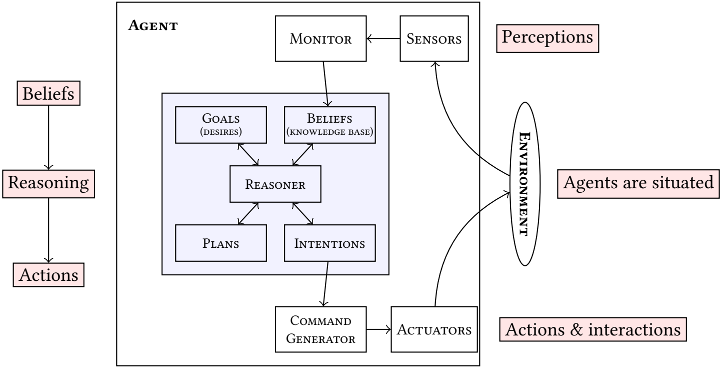
*(Georgeff, 1993)*




*From knowledge to signal…*

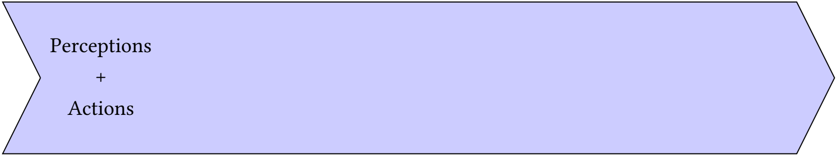
# Procedural Reasoning System (PRS)

*(Georgeff, 1993)*



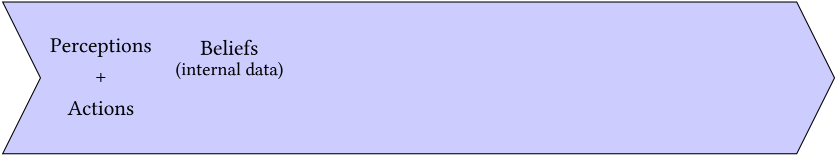

→ Procedural Loop

# A Continuum of Architectures



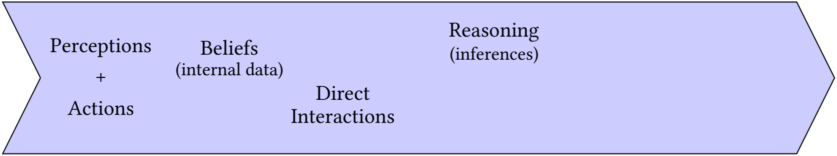

Reactive Agent

# A Continuum of Architectures




Reactive Agent Cognitive Agent

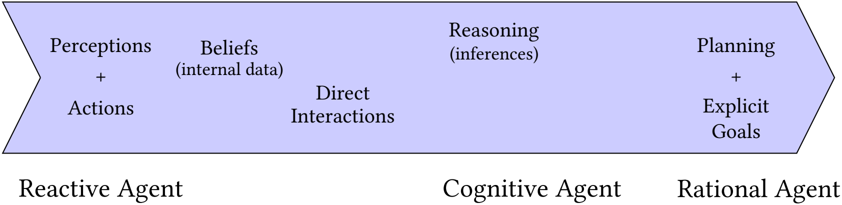
# A Continuum of Architectures



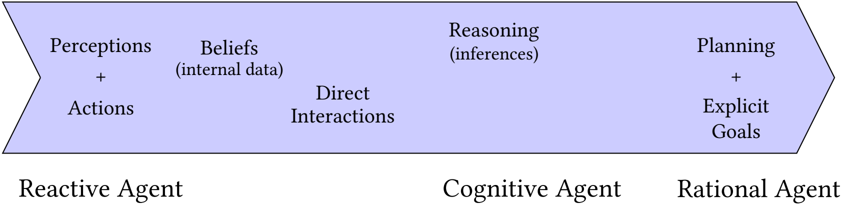

Reactive Agent

Cognitive Agent

# A Continuum of Architectures



# A Continuum of Architectures




## Agents = actions

- Reactive agent = rules of the form “ perception → action ”
- Cognitive and rational agent = reasoning

## Rational agents = goals

- Planning (or plan selection) + adaptation

# Definitions

## Agent

- Encapsulation $\rightarrow$ no direct access from other agents (use methods)
- Action = preconditions (when to activate) + effects (what to do)
  - Internal actions (modifications of beliefs)
  - Exogeneous actions = actions on the environment (+ interactions)
- Perception can be passive or active (*i.e.* as part of actions)
  - *Interaction is generally active*

# Definitions

## Agent

- Encapsulation → no direct access from other agents (use methods)
- Action = preconditions (when to activate) + effects (what to do)
  - Internal actions (modifications of beliefs)
  - Exogeneous actions = actions on the environment (+ interactions)
- Perception can be passive or active (*i.e.* as part of actions)
  - *Interaction is generally active*

## Environment

Runtime environment + can also encapsulate data

- Runtime : synchronous or asynchronous agents
- Situatedness : methods for perception and actions

# Definitions

## Agent

- Encapsulation → no direct access from other agents (use methods)
- Action = preconditions (when to activate) + effects (what to do)
  - Internal actions (modifications of beliefs)
  - Exogeneous actions = actions on the environment (+ interactions)
- Perception can be passive or active (*i.e.* as part of actions)
  - *Interaction is generally active*

## Environment

Runtime environment + can also encapsulate data

- Runtime : synchronous or asynchronous agents
- Situatedness : (explicit) methods for perception and actions

# Definitions

## Agent

- Encapsulation → no direct access from other agents (use methods)
- Action = preconditions (when to activate) + effects (what to do)
  - Internal actions (modifications of beliefs)
  - Exogeneous actions = actions on the environment (+ interactions)
- Perception can be passive or active (*i.e.* as part of actions)
  - *Interaction is generally active*

## Environment

Runtime environment + can also encapsulate data

- Runtime : synchronous or asynchronous agents
- Situatedness : (explicit) methods for perception and actions
- Communication : synchronous method calls to support asynchronous interaction

  (*next lecture…*)

# Two agents within an environnement I

```
public class Agent {
  private int id;
  public Agent(int n) { id = n; }

  public void procedural_loop() {
    while (true) {
      // one single action, exogenous, no precond
      System.out.println(id + ": hello!");
    }
  }

  public static void main(String [] args) {
    (new Agent(1)).procedural_loop(); // agent #1
    (new Agent(2)).procedural_loop(); // agent #2
  }
}
```

**MultiAgent System**

A MAS starts at 3 ! $\geq 2$ agents + 1 environment

# Two agents within an environnement I

```
public class Agent {
  private int id;
  public Agent(int n) { id = n; }

  public void procedural_loop() {
    while (true) {
      // one single action, exogenous, no precond
      System.out.println(id + ": hello!");
    }
  }

  public static void main(String [] args) {
    (new Agent(1)).procedural_loop(); // agent #1
    (new Agent(2)).procedural_loop(); // agent #2
  }
}
```

**MultiAgent System**

A MAS starts at 3 ! $\geq 2$ agents + 1 environment

**Runtime**

1 : hello !

1 : hello !

...

# Two agents within an environnement II

```
public class Agent {
  int id;
  public Agent(int n) { id = n; }

  public void procedural_loop() {
    System.out.println(id + ": hello!");
  }

  public static void main(String [] args) {
    Agent[] t = { new Agent(1), new Agent(2) };
    while (true) {
      for(Agent a : t) { a.procedural_loop(); }
    }
  }
}
```

**Synchronous solution**

The runtime environment simulates processes scheduling.

- Do not assume that the agents are called in a specific order!

# Two agents within an environnement III

```
public class Agent extends Thread {
  int id;
  public Agent(int n) { id = n; }

  @Override
  public void run() {
    while (true) {
      procedural_loop();
      Thread.sleep(10);
    }
  }
  public void procedural_loop() { ... }

  public static void main(String [] args) {
    (new Agent(1)).start();
    (new Agent(2)).start();
  }
}
```

**Asynchronous solution**

Runtime is handled by the OS via threads.

# Two agents within an environnement III

```
public class Agent extends Thread {
  int id;
  public Agent(int n) { id = n; }

  @Override
  public void run() {
    while (true) {
      procedural_loop();
      Thread.sleep(10);
    }
  }
  public void procedural_loop() { ... }

  public static void main(String [] args) {
    (new Agent(1)).start();
    (new Agent(2)).start();
  }
}
```

## Note

Agents in these examples do not communicate.
They do not even perceive their environment : they act blindly (*print* is an exogeneous action).

- This is not a MAS.

## Asynchronous solution

Runtime is handled by the OS via threads.

# Two agents within an environnement IV

## Perception…

…can be a passive process !

```
public class Agent extends Thread {
  static int env_variable = 0;
  int id;
  public Agent(int n) { id = n; }
  ...
  public void procedural_loop() {
    if (env_variable != id) {    // precondition
      env_variable = id;         // effect
    }
  }
  public static void main(String [] args) { ... }
}
```

# Two agents within an environnement IV

## Perception…

…can be a passive process !

- This is a reactive agent : perception → actions

```
public class Agent extends Thread {
  static int env_variable = 0;
  int id;
  public Agent(int n) { id = n; }
  ...
  public void procedural_loop() {
    if (env_variable != id) {    // precondition
      env_variable = id;         // effect
    }
  }
  public static void main(String [] args) { ... }
}
```

# Two agents within an environnement IV

## Perception…

…can be a passive process !

- Cognitive agents will separate perception from action selection

```
public class Agent extends Thread {
  static int env_variable = 0;
  int id, my_variable = 0;       // belief
  ...
  public void procedural_loop() {
    my_variable = env_variable;  // perception
    if (my_variable != id) {     // precondition
      env_variable = id;         // effect
      my_variable = id;
    }
  }
  public static void main(String [] args) { ... }
}
```

# Two agents within an environnement IV

## Perception...

...can be a passive process !

- Cognitive agents will separate perception from action selection

```
public class Agent extends Thread {
  static int env_variable = 0;
  int id, my_variable = 0;       // belief
  ...
  public void procedural_loop() {
    my_variable = env_variable;  // perception
    if (my_variable != id) {     // precondition
      env_variable = id;         // effect
      my_variable = id;
    }
  }
  public static void main(String [] args) { ... }
}
```

## Difference ?

# Two agents within an environnement IV

## Perception…

…can be a passive process !

- Cognitive agents will separate perception from action selection

```
public class Agent extends Thread {
  static int env_variable = 0;
  int id, my_variable = 0;       // belief
  ...
  public void procedural_loop() {
    my_variable = env_variable;  // perception
    if (my_variable != id) {     // precondition
      env_variable = id;         // effect
      my_variable = id;
    }
  }
  public static void main(String [] args) { ... }
}
```

## Difference ?

- Beliefs = internal variable
- May differ from environment's actual state

# Two agents within an environnement V

## Actions

- Agents consider a set of possible actions at each turn
- The procedural loop consists in
    1. Perception
    2. Action(s) selection based on preconditions
    3. Effects application

```
public interface Action {
  public boolean preconditions(Agent a);
  public void effect(Agent a, Environment e);
}
public class Agent {
  ...
  public void procedural_loop() {
    perceive(getEnvironment());
    List<Action> possibleActions = ...;
    for(Action a : getActions())
      if (a.preconditions())
        possibleActions.add(a);
   selectAndPerform(possibleActions,getEnvironment());
  }
}
```

- One possibile implementation among plenty of others…

*We will see how this is implemented in Jade*

# Concurrency

Multiagent systems consists of several processes executing in parallel

## Concurrency

- At the implementation level : different agents access a shared ressource in the system

  E.g. Access to data in the environment

- At the logical level : agents have to share ressources with incompatible goals

  E.g. A wants i>10 while B wants i<5

# Concurrency

Multiagent systems consists of several processes executing in parallel

## Concurrency

- At the implementation level : different agents access a shared ressource in the system

  E.g. Access to data in the environment

- At the logical level : agents have to share ressources with incompatible goals

  E.g. A wants i>10 while B wants i<5

## Difficulties

- The MAS platform prevents concurrent access to ressources

  In general, ressource encapsulated within an agent that manages concurrency

- MAS interaction protocols, such as negotiation and argumentation protocols, are used to deal with concurrency issues

# Synchronicity

Multiagent systems consists of several processes executing in parallel

## Synchronous agents

Agents always interact in a asynchronous manner

- Interaction is done throught action and perception → asynchronous

see course 2

# Synchronicity

Multiagent systems consists of several processes executing in parallel

## Synchronous agents

Agents always interact in a asynchronous manner

- Interaction is done throught action and perception → asynchronous

see course 2

## Synchronous systems

A MAS is synchronous if all agents perform one perception-decision-action cycle one after the other, in turn.

- No assumption should be made on the order of execution of the agents !

# Synchronicity

Multiagent systems consists of several processes executing in parallel

## Synchronous agents

Agents always interact in a asynchronous manner

- Interaction is done throught action and perception → asynchronous

see course 2

## Synchronous systems

A MAS is synchronous if all agents perform one perception-decision-action cycle one after the other, in turn.

- No assumption should be made on the order of execution of the agents !

## Classical problem : Two Generals Problem

Asynchronous systems + message loss → non-determinism !

# Openness, Coupling, Homogeneity

## Open MAS

Agents can enter and leave the system randomly

- New features/services can appear, others can disappear
- Possible messages loss $\rightarrow$ using timeouts (see course 2)

# Openness, Coupling, Homogeneity

## Open MAS

Agents can enter and leave the system randomly

- New features/services can appear, others can disappear
- Possible messages loss → using timeouts (see course 2)

## Loosely coupled MAS

A MAS is tightly coupled if the agents' behaviours encode assumptions about the other agents's services and data.

E.g. when you develop your own MAS knowing what each agent can do

- Loosely coupled MAS raises similar questions to openness

# Openness, Coupling, Homogeneity

## Open MAS

Agents can enter and leave the system randomly

- New features/services can appear, others can disappear
- Possible messages loss $\rightarrow$ using timeouts (see course 2)

## Loosely coupled MAS

A MAS is tightly coupled if the agents' behaviours encode assumptions about the other agents's services and data.

E.g. when you develop your own MAS knowing what each agent can do

- Loosely coupled MAS raises similar questions to openness

## Homogenous MAS

All agents have offer the same features and knowledge representation

More to come during course 2

# Distribution

## Level of distribution

- Conceptual level

  E.g. Netlogo or Mason → central scheduler, one process, agent-based concepts implemented in the execution loop

- Software level

  E.g. Jade or Repast on a single machine→ Agents are Threads

- Hardware level

  E.g. Jade or Repast on multiple hosts → decentralized execution with brokers

# Autonomy

## What is autonomy ?

- Agents do not behave randomly !

They execute precise code, follow well-designed interaction protocols

# Autonomy

## What is autonomy ?

- Agents do not behave randomly !

They execute precise code, follow well-designed interaction protocols

An agent can be autonomous :

- w.r.t. its environment
  E.g. do not obey a stop sign
- w.r.t. other agents
  E.g. do not obey whistle, do not answer a message, etc.

# Autonomy

## What is autonomy ?

- Agents do not behave randomly !

They execute precise code, follow well-designed interaction protocols

An agent can be autonomous :

- w.r.t. its environment
  E.g. do not obey a stop sign
- w.r.t. other agents
  E.g. do not obey whistle, do not answer a message, etc.

…but this has to be programmed !

# Autonomy

## What is autonomy ?

- Agents do not behave randomly !

They execute precise code, follow well-designed interaction protocols

An agent can be autonomous :

- w.r.t. its environment
  E.g. do not obey a stop sign
- w.r.t. other agents
  E.g. do not obey whistle, do not answer a message, etc.

…but this has to be programmed !

## Agent's conception

Take into account the autonomy of its peer (especially in open, loosely coupled, heterogenous MAS) !

# How to design MAS

## Environment

- What is to be computed by the MAS ?
- What are the perception and action operations ?

## Agent

- What are the internal data (beliefs)
- What are the actions :
  - Preconditions (cognitive agents = internal data ; reactive = perceptions)
  - Effects $\rightarrow$ internal data or exogeneous operations

## General mechanisms

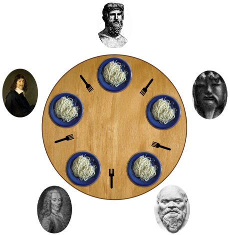
- Synchronous or asynchronous runtime for agents ?
- One or several actions per cycle ?
- Interactions (*next lecture...*)

# Practical exercice : homework

## Dining Philosophers

*Proposed by Dijkstra in 1971 to study concurrency*

- A set of *n* agents ;
- Environment = circular table with a fork between each pair of agents ;
- Agents need its two nearby forks to eat ;
- Agents must have eaten to think and they need to eat after thinking ;
- To avoid deadlocks, agents only take the forks if both are available.



- Design this problem as a MAS.