Agent and Multi-Agent Systems: architectures and reasoning

Concepts and Definitions

27.09.2021

CentraleSupélec- SAFRAN AI Training

Organization of the course

Organization

Dates	
16/09/2020	Course 1: Introduction to Agents and MAS
17/09/2020	Course 2 : Multi-Agent based Simulation
18/09/2020	Course 3 : Interaction mechanisms: models and Implementation

Some References

- Ferber, J. (1995), Les Systèmes Multi-Agents, Inter Editions. (French version)
- Ferber, J. (1999): Multi-agent systems: An introduction to distributed artificial intelligence (english version)
- Michael Wooldridge (2002), An Introduction to MultiAgent Systems, John Wiley & Sons Ltd
- Robert E. Shannon (1977), Simulation modeling and methodology, SIGSIM Simulation Digital.
- Robert E. Shannon (1998), Introduction to the art and science of simulation, IEEE Computer Society Press.
- Bernard P. Zeigler (2000), Theory of Modeling and Simulation, Academic Press, Inc.

Artificial Intelligence (AI)

"Designing and building machines that do things that would require intelligence if performed by humans" [Marvin Minsky].

Source: https://humanoides.fr/robot-appris-jouer-echecs-tout-seul/

Artificial Intelligence (AI)

"Designing and building machines that do things that would require intelligence if performed by humans" [Marvin Minsky].

Source: https://humanoides.fr/robot-appris-jouer-echecs-tout-seul/

➙ the concept of "Intelligent agent" at the base of AI

Table of contents

1. A brief history of Multiagent Systems

2. What is an agent?

3. Properties and Defintions

4. The mesa platform for Python

A brief history of Multiagent Systems

A brief history of Multiagent Systems

- Multiagent Systems are a part of Computer Science, at the frontier between: *software engineering*, *distributed computing* and *artificial intelligence*.
- They appeared in the 1990s when the research and engineering questions of these three subfields came to similar solutions to deal with different problems

MAS as seen by a software engineer

- Software engineering: how to maximise the speed, the reusability, the readability, the security of code, ...
- *From Object to Service*: a piece of code, possibly a thread, should be provided with some machine-understandable interface that could allow other code to use it without a priori knowledge of its operational content.
- Software engineers rely on **interaction protocols** to define how the different services should be assembled, how the information has to be exchanged between them, so as to produce some expected result.

MAS as seen by a software engineer

- Multiagent systems: a series of platforms to support the development of such service-oriented software, with coordination entities capable to interpreting protocols.
- for a software engineer's point of view, what matters in multiagent systems is the possibility to **assemble separate components or threads by designing well-formed interaction protocols**.

MAS as seen by a distributed systems engineer

- Idea: different processes are located on different physical systems (all connected by some network infrastructure) and interact with each other using some message passing mechanism.
- the agents (or processes) can run on one single computer. What matters is that they have *independent memory spaces* and *runtimes*. $\rightsquigarrow$ cannot rely on synchronous interactions

```
1 def compute(x,y,z):
2     ... computation here ...
3     return result
4
5 def calling_function():
6     ... beginning of code ...
7     res = compute(1,3,'hello')
8     ... end of code ...
```

MAS as seen by a distributed systems engineer

DS consider situation in which several problems can occur:

- The called process can not be running, or even not be accessible in the system;
- The called process can not answer, whatever the reason;
- If an answer arrives, there is no guarantee on the delay before it arrives;

All these problems assume a perfectly functioning infrastructure

$\rightsquigarrow$ programming a piece of software by taking into account the fact that the other half of the code is run in a completely asynchronous manner.

MAS as seen by a distributed systems engineer

Consequences !

- Agents (processes) send message but don't wait for the answer: they must carry on their activity while a possible answer is computed by another agent;
- Agents must use timeouts to deal with the absence of answers.

MAS as seen by a distributed systems engineer

Consequences !

- Agents (processes) send message but don't wait for the answer: they must carry on their activity while a possible answer is computed by another agent;
- Agents must use timeouts to deal with the absence of answers.

$\rightsquigarrow$ For a distributed system engineer, what matters in multiagent systems is **the capacity of agent to deal with asynchronous interactions and possible errors**.

MAS as seen by an Artificial Intelligence engineer

- Artificial Intelligence is about having machine solve problems (through computation) that human beings solve with their intelligence.
- A subfiled of AI: Planning
  - design of actions and changes
  - solve problems that consist in finding a sequence of actions to go from a given state to another.
- The STanford Research Institute Problem Solver (STRIPS) is one of the first (and most famous) planner.

MAS as seen by an Artificial Intelligence engineer

```
1
2  Initial state: At(A), Level(low), BoxAt(C), BananasAt(B)
3      Goal state:    Have(bananas)
4
5      Actions:
6                  // move from X to Y
7                  _Move(X, Y)_
8                  Preconditions:  At(X), Level(low)
9                  Postconditions: not At(X), At(Y)
10
11                 // climb up on the box
12                 _ClimbUp(Location)_
13                 Preconditions:  At(Location), BoxAt(Location), Level(low)
14                 Postconditions: Level(high), not Level(low)
15
16                 // climb down from the box
17                 _ClimbDown(Location)_
18                 Preconditions:  At(Location), BoxAt(Location), Level(high)
19                 Postconditions: Level(low), not Level(high)
20
21                 // move monkey and box from X to Y
22                 _MoveBox(X, Y)_
23                 Preconditions:  At(X), BoxAt(X), Level(low)
24                 Postconditions: BoxAt(Y), not BoxAt(X), At(Y), not At(X)
25
26                 // take the bananas
27                 _TakeBananas(Location)_
28                 Preconditions:  At(Location), BananasAt(Location), Level(high)
29                 Postconditions: Have(bananas)
```

A solution: Move(A,C), MoveBox(C,B), ClimbUp(B), TakeBananas(B) (automatically is an NP-hard problem).

MAS as seen by an Artificial Intelligence engineer

Collective Intelligence: two approaches

- BDI (Belief, Desire, Intention): a new logic for modelling actions and changes and simulating human-like behaviour

  $\rightsquigarrow$ programming systems that *simulate* human decision making for action selection based on its perception of the environment

- Social Animals Behaviour. The solution does not arise from the behaviour of one single individual, but it *emerges* from the interactions between the individuals.

  $\rightsquigarrow$ programming systems that interact with each other to compute a solution.

$\rightsquigarrow$ For an AI engineer, what matters in multiagent systems is **the capacity of agents to solve problems in a distributed manner, using interactions and automated reasoning**.

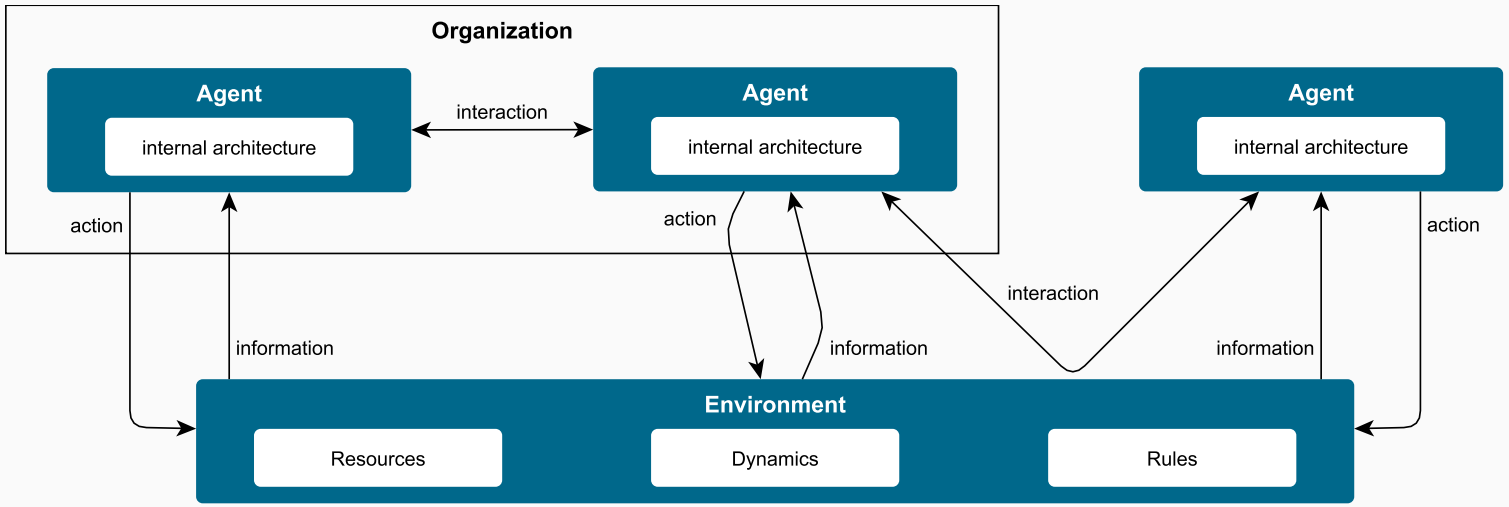
To be remembered

An agent is :

- a software process (or part of a process, or possibly attached to some physical components e.g. in robotics. . .)
- with some encapsulated data (i.e. other agents/processes can't modify directly this data),
- capable of providing a predefined set of services (its actions)
- and capable of reasoning about its actions and the other agents' actions and to coordinate with the other agents so as to participate in a **collective problem solving**.

To be remembered

- A MAS is a group of agents that share a common environment and that act in a distributed manner (the agents runtime are supposed to be asynchronous) to solve a problem for a user.
- A MAS always has a predefined purpose, and it is important to define the expected outcomes, should it be a solution to a concrete problem (e.g. a path in a TSP problem) or a set of observable properties in a multiagent based simulation.



Some Practical Application of MAS

MAS Applied to Fault Detection

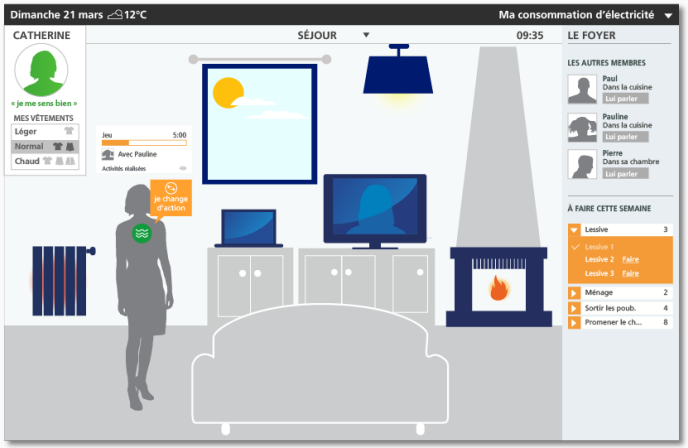
- SurferLab is a joint research lab formed by Bombardier Transport and the university of Valencienne, France.
- MAS is used for diagnosis of trains with the Jade platform.
- The multiagent systems is capable to detect faulty cars so as to correct them faster[1].

[1] Antoine Le Mortellec, Joffrey Clarhaut, Yves Sallez, Thierry Berger, Damien Trentesaux, Embedded holonic fault diagnosis of complex transportation systems, Engineering Applications of Artificial Intelligence, Volume 26, Issue 1, 2013

Some Practical Application of MAS

MAS Applied to the Study of Electrical Consumption

- The R&D branch of EDF has been working on a MAS for the simulation of electrical consumption in households. This applied research project is called SMACH[2].
- The MAS platform is used for the generation of realistic load curves in answer to prospective studies: market research & pricing, new production means, new consumption habits,etc.



[2] https://www.youtube.com/watch?v=Hyc7XX4vEjw&t=17s

Some Practical Application of MAS

MAS Applied to Taxi Fleet Management

- The company Magenta Technology has developed in 2008 a MAS-based technology for the management of taxi fleets.
- This software solution is now being used by many taxi companies, such as Green Tomato Cars or Blackberry Cars, two major UK companies in London, UK.

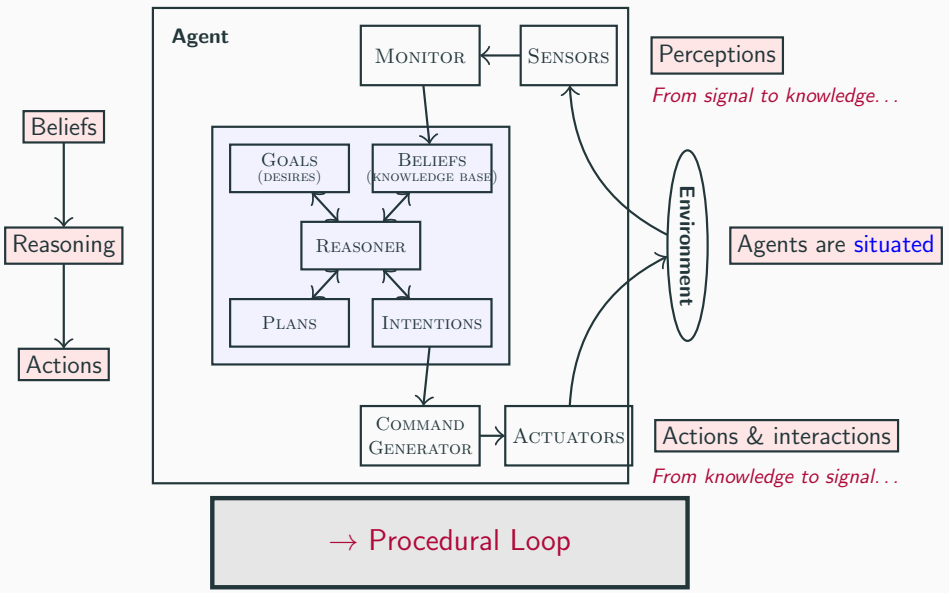
Some Practical Application of MAS

MAS for Crowd Simulation in Movies

- **Massive Software** was originally developed for use in Peter Jackson's The Lord Of The Rings film trilogy.
- Massive has become the leading software for crowd related visual effects and autonomous character animation.
- It combines MultiAgent Systems with Graphical Design.

What is an agent?

PRS architecture [Georgeff and Lansky 87]



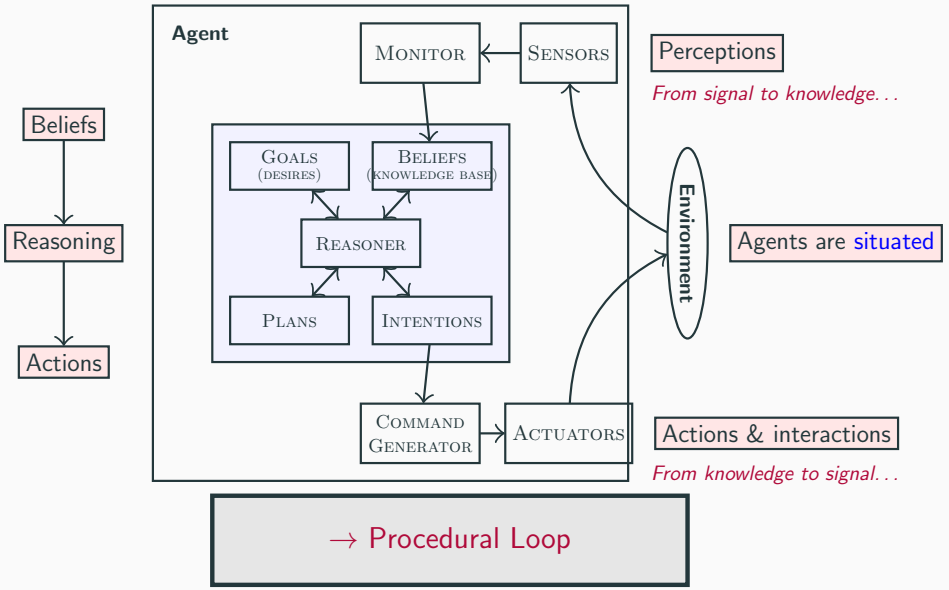
- PRS: Procedural Resoning System
- The environment: characterized by a (partially) observable state:
  - it can be modified,
  - some of the variable values can be observed by the agents.

  $\rightsquigarrow$ a set of observation functions and a set of modification functions.
- Agent: sensors (to access the observation function) + actuators (to access the modification functions).

PRS architecture [Georgeff and Lansky 87]

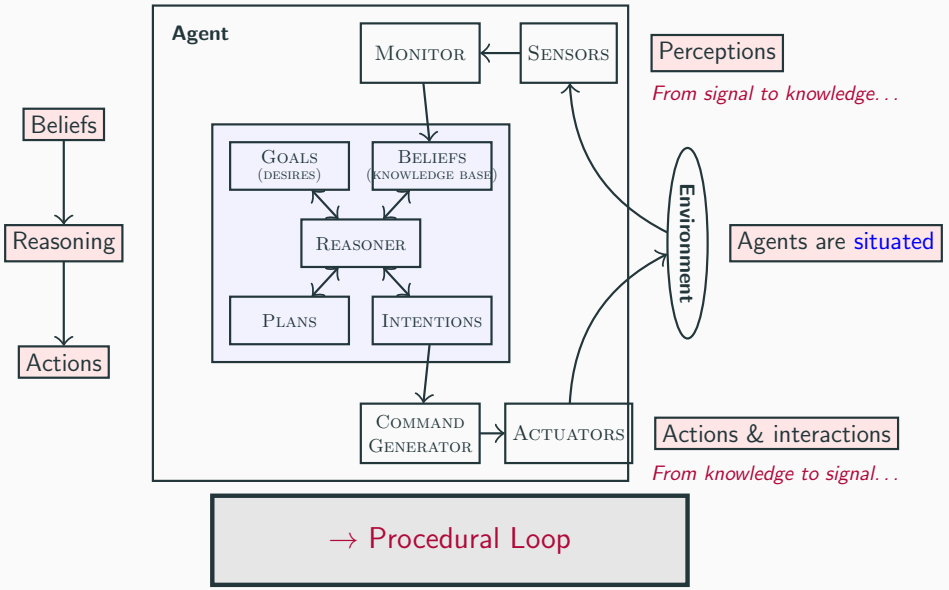
```
environment_variable = 0

def operation_increase_variable():
    global environment_variable
    environment_variable += 1

def perception_get_variable():
    global environment_variable
    return environment_variable

def agent(preferred_value):
    v = perception_get_variable() # this is a sensor
    while (v<preferred_value):
        operation_increase_variable() # this is an actuator
        v = perception_get_variable()

```

PRS architecture [Georgeff and Lansky 87]



- Procedural Loop: an agent runs continuously in three steps:
  1. Perception of the environment
  2. Deliberation = selection of the action (i.e. transforms the result of the perception into internal variables, called *beliefs*. It then combines these with a *set of goals* and a *set of plans* so as to decide for some *intention* that will be turn into a *cconcrete action* in the environment.)
  3. Action on the environment

PRS architecture [Georgeff and Lansky 87]



- Procedural Loop: an agent runs continuously in three steps:
  1. Perception of the environment
  2. Deliberation = selection of the action (i.e. transforms the result of the perception into internal variables, called *beliefs*. It then combines these with a *set of goals* and a *set of plans* so as to decide for some *intention* that will be turn into a *cconcrete action* in the environment.)
  3. Action on the environment

To keep in mind: an agent

- Is situated in an environment: it can perceive and act;
- Continuously performs a procedural loop: perception, deliberation, action.

An agent: remark

There is not "the" definition of an agent

[Russel and Norvig,1995]

"An agent is anything that can be viewed as perceiving its environment through *sensors* and acting upon that environment through *effectors*."

An agent: remark

There is not "the" definition of an agent

[Russel and Norvig,1995]

"An agent is anything that can be viewed as perceiving its environment through *sensors* and acting upon that environment through *effectors*."

[Wooldridge & Jennings, 1995]

"An agent is a computer system that is *situated* in some environment, and that is capable of *autonomous* action in this environment in order to meet its design objectives"

...

An agent

Questions?

Agent Type	Environment	Actuators	Sensors
Medical Diagnosis System			
Satellite image analysis system			

Procedural Reasoning System

Examples of agent types

Agent Type	Environment	Actuators	Sensors
Medical Diagnosis System	Patients, hospital, staff	Display questions, tests, diagnosis, treatments	Entry of symptoms, findings, patient's answer
Satellite image analysis system	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays

Agent Models

key concepts

- Reactive agents: do not make use of any internal variable: they directly wire all perception to a specific action (or set of actions), hence their name: they perceive and react.

Agent Models

key concepts

- Reactive agents: do not make use of any internal variable: they directly wire all perception to a specific action (or set of actions), hence their name: they perceive and react.
- Cognitive agents:
  - add internal variables and data
  - a first type: internal variables (beliefs) to store its perception $\rightsquigarrow$ it has a memory of what it perceived, and it can make decisions based on this memory.

STRIPS example

```
Move(A,C), MoveBox(C,B), ClimbUp(B), TakeBananas(B)
```

Agent Models

key concepts

- Reactive agents: do not make use of any internal variable: they directly wire all perception to a specific action (or set of actions), hence their name: they perceive and react.
- Cognitive agents:
  - add internal variables and data
  - a first type: internal variables (beliefs) to store its perception $\rightsquigarrow$ it has a memory of what it perceived, and it can make decisions based on this memory.
  - must be capable of handling with action failures.

STRIPS example

```
Move(A,C), MoveBox(C,B), ClimbUp(B), TakeBananas(B)
```

Agent Models

key concepts

- Reactive agents: do not make use of any internal variable: they directly wire all perception to a specific action (or set of actions), hence their name: they perceive and react.
- Cognitive agents:
  - add internal variables and data
  - a first type: internal variables (beliefs) to store its perception $\rightsquigarrow$ it has a memory of what it perceived, and it can make decisions based on this memory.
  - must be capable of handling with action failures.
  - "explicit planning" is not mandatory. Agent simply checks its current state of beliefs and search for a possible action. The "plan" is then hard-wired in the code. Different levels of "reasoning" about the beliefs: simple deduction, belief revision, diagnosis, ...

STRIPS example

```
Move(A,C), MoveBox(C,B), ClimbUp(B), TakeBananas(B)
```

Agent Models

key concepts

- Reactive agents: do not make use of any internal variable: they directly wire all perceptions to a specific action (or set of actions), hence their name: they perceive and react.
- Cognitive agents:
  - add internal variables and data
  - a first type: internal variables (beliefs) to store its perception $\rightsquigarrow$ it has a memory of what it perceived, and it can make decisions based on this memory.
  - must be capable of handling with action failures.
  - "explicit planning" is not mandatory. Agent simply checks its current state of beliefs and search for a possible action. The "plan" is then hard-wired in the code. Different levels of "reasoning" about the beliefs: simple deduction, belief revision, diagnosis, ...
- "explicit planning": agents that have explicit goals and that perform any sort of computation (from simple plan selection in a predefined list to complex planing algorithms such as SATPLAN) $\rightsquigarrow$ rational agents.

To be remembered

According to the PRS model that is at the core of agent modeling, an *agent*:

- is situated in the environment (it can perceive and act);
- might have some internal data called beliefs and these are encapsulated within the agent (they are not accessible to other agents or to the environment);
- runs with a procedural loop (perceive, decide for an action, act) in an asynchronous w.r.t other agents;
- in the decision phase, it can do any sort of reasoning, from simple reaction (reactive agents) to more complex reasoning on the beliefs (cognitive agents);
- cognitive agents consider actions with preconditions and effects (note: this can be way more complex but we shall skip the details for this MAS introductory course);
- agents that have *explicit goals* and *planning procedures* in the deliberation phase are called rational agents.

Agent and MAS: Exercices

Open the notebook: IntroductionMAS_2021_Student.ipynb

```
from time import sleep

class Environment:
    def act(self,message):
        print(message)
    def perceive(self):
        pass

class Agent:
    def __init__(self,name,env):
        self.name = name
        self.env = env
    def procedural_loop(self):
        while True:
            self.env.act("Agent "+self.name+" says hello!")
            sleep(0.1)

class Runtime:
    def __init__(self):
        e = Environment()
        (Agent("Alice",e)).procedural_loop()
        (Agent("Bob",e)).procedural_loop()

Runtime()

```

Agent and MAS: Exercices

Questions

- Why does this not behave as a multiagent system?
- What is the problem?

Agent and MAS: Exercices

Questions

- Why does this not behave as a multiagent system?
- What is the problem?

Answers

Running this code produces a continuous output of “Agent Alice says hello!”.
It does not behave as a MAS because the agents do not run asynchronously.
The reason is that the runtime never leaves the procedural loop of agent Alice.
Run this code in debug mode, with a break point in the first line of the procedural_loop method if you have any doubt.

Agent and MAS: Exercices

Questions

To overcome the above limitation, two solutions can be considered.

The first one is to write a **scheduler**, i.e. a piece of code that calls the procedural loops of all agents, one after the other.

- Modify the previous code so that Runtime creates two agents and calls a single-step procedural loop for all agents.

Answers. Version1: with home made scheduler

```
1
2  from time import sleep
3
4  class Environment:
5      def act(self,message):
6          print(message)
7      def perceive(self):
8          pass
9
10 class Agent:
11     def __init__(self,name,env):
12         self.name = name
13         self.env = env
14     def procedural_loop(self):
15         self.env.act("Agent "+self.name+" says hello!")
16         sleep(0.1)
17
18 class Runtime:
19     def __init__(self):
20         e = Environment()
21         a = Agent("Alice",e)
22         b = Agent("Bob",e)
23         while True:
24             a.procedural_loop()
25             b.procedural_loop()
26
27 Runtime()
28
```

Answers. Version1: with home made scheduler

1. This MAS verifies a specific property, which is that the procedural loop of all agents is performed at each time step: all agents run at the same "speed".
   This is called a **synchronous MAS**. While the agent has its own runtime, interleaved with the one of the other agents (which corresponds to the specification of a MAS) these runtimes are synchronised.
2. The agents procedural loops are always invoked in the same order, which is not a valid hypothesis in a MAS. Agents should **never** use that property.

If you want to avoid this, you can simply modify the Runtime class as follows:

Answers. Version1: with home made scheduler – Avoiding the same order

```
1
2  from time import sleep
3
4  class Environment:
5      def act(self,message):
6          print(message)
7      def perceive(self):
8          pass
9
10 class Agent:
11     def __init__(self,name,env):
12         self.name = name
13         self.env = env
14     def procedural_loop(self):
15         self.env.act("Agent "+self.name+" says hello!")
16         sleep(0.1)
17
18 class Runtime:
19     def __init__(self):
20         e = Environment()
21         agents = [Agent("Alice", e), Agent("Bob", e)]
22         while True:
23             shuffle(agents)
24             for a in agents:
25                 a.procedural_loop()
26 Runtime()
27
```

Note that this new version still is a synchronous MAS.

Answers. Version2: with Threads

```
1  from time import sleep
2  from threading import Thread
3
4  class Environment:
5      def act(self, message):
6          print(message)
7
8      def perceive(self):
9          pass
10
11 class Agent(Thread):
12     def __init__(self, name, env):
13         Thread.__init__(self)
14         self.name = name
15         self.env = env
16
17     def run(self):
18         while True:
19             self.procedural_loop()
20
21     def procedural_loop(self):
22         self.env.act("Agent " + self.name + " says hello!")
23         sleep(0.1)
24
25 class Runtime:
26     def __init__(self):
27         e = Environment()
28         a = Agent("Alice", e)
29         b = Agent("Bob", e)
30         a.start()
31         b.start()
32
33 Runtime()
```

Agent and MAS: Exercices

Questions

Do we have a multiagent system yet?

Agent and MAS: Exercices

Questions

Do we have a multiagent system yet?

Answers

The agents in the preceding example are not really situated since the perception phase does nothing. They use encapsulated data (the agent's name), but we can't really call these a set of beliefs since they are not connected to the perception. Although they have asynchronous runtimes, with a procedural loop, we cannot really call them agents, even not reactive agents. In order to write reactive agents, we want the deliberation phase, in the procedural loop, to depend on the perceptions.

Agent and MAS: Exercices

Questions

Write a new multiagent system, either in the synchronous or in the asynchronous version, in which the environment has some variable that the agents can perceive. Write two reactive agents: the first one increases the variable by a random value when it is even, the other one when it odd.

Answers. Two agents and an environment's variable

```
1
2  from time import sleep
3  from random import *
4  from threading import Thread
5
6  class Environment:
7      v = 0
8
9      def increase(self, name):
10         x = randint(1,4)
11         print("Agent " + name + " increase the value by "+ str(x))
12         self.v = self.v+x
13         print("  --> " + str(self.v))
14
15     def perceive(self):
16         return self.v
17
18 class Agent(Thread):
19     def __init__(self, name, env):
20         Thread.__init__(self)
21         self.name = name
22         self.env = env
23
24     def run(self):
25         while True:
26             self.procedural_loop()
27
28     def procedural_loop(self):
29         self.b = self.env.perceive()
30         self.act()
31         sleep(uniform(0.1,0.5))
32
```

Answers. Two agents and an environment's variable (next)

```
1
2
3  class AgentOdd(Agent):
4      def act(self):
5          if self.b%2!=0:
6              self.env.increase(self.name)
7
8  class AgentEven(Agent):
9      def act(self):
10         if self.b % 2 == 0:
11             self.env.increase(self.name)
12
13 class Runtime:
14     def __init__(self):
15         e = Environment()
16         a = AgentOdd("Alice", e)
17         b = AgentEven("Bob", e)
18         a.start()
19         b.start()
20
21 Runtime()
22
```

Concurrent Modifications

Writing asynchronous MAS with threads requires to consider possible concurrent modifications.

```
self.b = self.env.perceive()     # in Agent.procedural_loop()
if condition_on_b:                # in AgentX.act()
    x = randint(1,4)              # in Environment.increase() -- 4
     lines
    print("Agent " + name + " increase the value by "+ str(x))
    self.v = self.v+
    print("  --> " + str(self.v))

```

Concurrent Modifications

Writing asynchronous MAS with threads requires to consider possible concurrent modifications.

```
1 self.b = self.env.perceive()     # in Agent.procedural_loop()
2 if condition_on_b:                # in AgentX.act()
3     x = randint(1,4)              # in Environment.increase() -- 4
       lines
4     print("Agent " + name + " increase the value by "+ str(x))
5     self.v = self.v+
6     print("  --> " + str(self.v))
7
```

Questions

- What do you expect to be a problem? Explain why.
- What is a possible solution?

Concurrent Modifications: Answers

- the scheduler interrupts the code between line 1 and 2, or even between line 2 and 3, and passes the "handle" to the other agent:
  $\rightsquigarrow$ the first agent will perform an increase operation on a variable that might not be of the right parity. This is perfectly correct according to a MAS specification!

Concurrent Modifications: Answers

- the scheduler interrupts the code between line 1 and 2, or even between line 2 and 3, and passes the "handle" to the other agent:
  $\rightsquigarrow$ the first agent will perform an increase operation on a variable that might not be of the right parity. This is perfectly correct according to a MAS specification!
- the scheduler stops in the middle of the Environment.increase() method.
  $\rightsquigarrow$ This is a problem because the calling agent could perfectly assume this method to be atomic, uninterruptible. This is what you expect when you perform operations in an environment.

  Property of a MAS: actions in the environment must be atomic

Concurrent Modifications: Answers

- the scheduler interrupts the code between line 1 and 2, or even between line 2 and 3, and passes the "handle" to the other agent:
  $\rightsquigarrow$ the first agent will perform an increase operation on a variable that might not be of the right parity. This is perfectly correct according to a MAS specification!
- the scheduler stops in the middle of the Environment.increase() method.
  $\rightsquigarrow$ This is a problem because the calling agent could perfectly assume this method to be atomic, uninterruptible. This is what you expect when you perform operations in an environment.

  Property of a MAS: actions in the environment must be atomic

```
1 from threading import Lock
2 l = Lock()
3 with l:
4     ... concurrent code ...
5
```

Properties and Defintions

Properties of an environment [Russel &Norvig 2003]

- Fully observable vs. partially observable – can agents obtain complete and correct information about the state of the world?
  $\rightsquigarrow$ Fully observable environments mean that all variables in the environment can be accessed by any agent.

```
class Environment:
    variables = { 'name1':'value1',
                  'name2':'value2',
                  ... }

    def perceive(self, name):
        return variables[name]

    def act(self, name, value):
        variables[name] = value


```

Properties of an environment [Russel &Norvig 2003]

- Deterministic vs. stochastic –Do actions have guaranteed and uniquely defined effects?

```
1  from threading import Thread
2  class Environment:
3      v = 0
4      def act(self):
5          self.v = 1 - self.v
6          print(self.v)
7      def perceive(self):
8          return self.v
9  class Agent(Thread):
10     def __init__(self, name, preferred_value, env):
11         Thread.__init__(self)
12         self.name = name
13         self.env = env
14         self.pv = preferred_value
15     def run(self):
16         while True:
17             self.procedural_loop()
18     def procedural_loop(self):
19         if self.env.perceive() != self.pv:
20             self.env.act()
21 class Runtime:
22     def __init__(self):
23         e = Environment()
24         Agent("Alice", 0, e).start()
25         Agent("Bob", 1, e).start()
26
27 Runtime()
28
```

Properties of an environment [Russel &Norvig 2003]

- Static vs. dynamic – Does the environment change by processes beyond agent control?

Properties of an environment [Russel &Norvig 2003]

- Discrete vs. continuous – Is the number of actions and percepts fixed and finite?

Resume

Properties of an environment [Russel &Norvig 2003]

- Fully observable vs. partially observable – can agents obtain complete and correct information about the state of the world?
- Deterministic vs. stochastic –Do actions have guaranteed and uniquely defined effects?
  independent episodes?
- Static vs. dynamic – Does the environment change by processes beyond agent control?
- Discrete vs. continuous – Is the number of actions and percepts fixed and finite?
  depends on the behavior of other agents?

Properties of an environment

Questions ?

Environment	Accessible	Deterministic	Static	Discrete
Medical Diagnosis System				

Properties of an environment

Examples of environments

Environment	Accessible	Deterministic	Static	Discrete
Medical Diagnosis System	✗	✗	✗	✗

Properties of the Agents in the MAS

Autonomous agents

The independence of one agent's behaviour (i.e. action selection) with respect to some part of the system.

1. **At the agent level**, pro-action is the first degree of autonomy (i.e. independently from any specific signal from the environment).
2. **At the interaction level**, autonomy consists in having agents that do not necessarily accept other agent's requests (ignoring messages because they are considered irrelevant; not answering immediately while it could have; agent answers but refuses the request; etc.).
3. **At the MAS level**, there is no autonomy: agents are not autonomous w.r.t. the system. On the contrary, they perform what they are designed for. They follow well-defined protocols so that the global task is achieved. Otherwise, there is some misconception in the system.

Properties of the Agents in the MAS

Loosely vs tightly coupled?

1. **In tightly coupled MAS**, the developer has a complete view of the MAS, of possible actions performed by other agents. In other words, it can use some of this information to design action mechanism in a more efficient manner.
2. **In a loosely coupled MAS**, you must make no assumption on the design of other agents in the MAS. This means that you must define very robust behaviours to avoid deadlocks, infinite loops or unnecessary waiting times in communication.

Properties of the Agents in the MAS

Open MAS?

- a specific kind of system in which you assume that agents can enter and leave the system at any time during the execution.
- requires programmers to design specific mechanism to ensure that messages are not left unanswered, that some feature in the system will not disappear when not expected, etc.
- Concrete examples of open MAS are ant colonies or prey-predator simulations

Properties of the Agents in the MAS

Distributed systems?

1. Conceptual level: it implements distribution principles: encapsulated data, accessed via perception and action methods only, using a *procedural loop* that interleaves the perceptions and actions of agents.
   $\rightsquigarrow$ Netlogo or Gamma
2. Software level: agents get separated in different threads or processes (depending on the language and implementation). These threads generally run on a single computer.
   $\rightsquigarrow$ Repast Symphony
3. Physical level: specific code to connect the abstract actions and perceptions to physical operations, message passing, etc.
   $\rightsquigarrow$ The Java Agent Development Environment (Jade) , The Robot OS (ROS)

The mesa platform for Python

The mesa platform for Python

Open Notebook: money_model_Students.ipynb

- Mesa is a Python framework for agent-based modeling.
- The running example will be the implementation of a simple Agent-based Computational Economics (ACE) model that simulates distributions of money, income, and wealth in society using statistical physics[3].

[3] https://arxiv.org/abs/cond-mat/0211175