



# Agent-based Modeling and Agent-Oriented Programming with the SARL Language

SARL Lectures - UHasselt - 2016

**prof.dr.habil. Stéphane GALLAND**

- A Introduction to Agent-Based Systems and SARL Language**
- B Agent Environment: Definition and Examples**
- C Organizational and Holonic Modeling**
- D Design and Implementation of an Agent Platform**



# Introduction to Agent-Based Systems and SARL Language

prof.dr.habil. Stéphane GALLAND

During this lecture, I will present:

- 1 the basics of the SARL programming language;
- 2 the basics of the multiagent simulation;
  
- 3 models of physic environment,
  - Examples: highway and crowd simulations;
- 4 a cyber-physic model,
  - Examples: intelligent autonomous vehicle.

- 1** Introduction to Multiagent Systems
- 2** Multiagent Simulation
- 3** Simulation with a Physic Environment
- 4** Cyber-physical System

## 1 Introduction to Multiagent Systems

- Agents
- Multiagent Systems
- Programming Multiagent Systems with SARL
- Development Environment
- Execution Environment

## 2 Multiagent Simulation

## 3 Simulation with a Physic Environment

## 4 Cyber-physical System

Five ongoing trends have marked the history of computing

- Ubiquity;
- Interconnection;
- Intelligence;
- Delegation;
- Human-orientation: easy/natural to design/implement/use.

Other Trends in Computer Science

- Grid Computing;
- Ubiquitous Computing;
- Semantic Web.

### Agent [Wooldridge, 2001]

An agent is an entity with (at least) the following attributes / characteristics:

- Autonomy
- Reactivity
- Pro-activity
- Social Skills - Sociability

No commonly/universally accepted definition.

## Autonomy

Agents encapsulate their internal state (that is not accessible to other agents), and make decisions about what to do based on this state, without the direct intervention of humans or others;

- Able to **act without any direct intervention** of human users or other agents.
- Has **control** over his own **internal state**.
- Has **control** over his own **actions** (no master/slave relationship)
- Can, if necessary/required, modify his behavior according to his personal or social experience (**adaptation-learning**).

## Reactivity

Agents are **situated in an environment**, (physical world, a user via a GUI, a collection of other agents, Internet, or perhaps many of these combined), are able to **perceive** this environment (through the use of potentially imperfect sensors), and are able to **respond in a timely fashion** to changes that occur in it;

- Environment static ⇒ the program can execute itself blindly.
- Real world as a lot of systems are highly **dynamic**: constantly changing, partial/incomplete information
- Design software in dynamic environment is difficult: failures, changes, etc.
- A reactive system perceives its environment and **responds in a timely appropriate fashion to the changes** that occur in this environment (Event-directed).

## Pro-activity

Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by **taking the initiative**; They pursue their own personal or collective goals.

- Reactivity is limited (e.g. Stimulus  $\Rightarrow$  Response).
- A proactive system generates and attempts to capture objectives, it is **not directed only by events, take the initiative**.
- Recognize/Identify opportunities to act/trigger something.

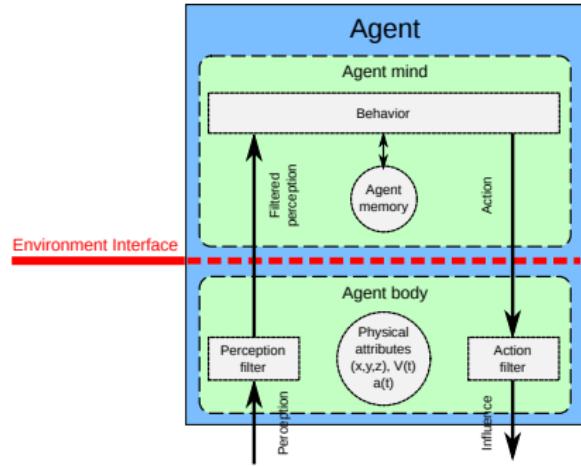
## Sociability - Social Ability

Agents interact with other agents (and possibly humans), and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals. Unity is strength.

- Many tasks can only be done by cooperating with others
- An agent must be able to interact with virtual or/and real entities
- Require a mechanism to exchange information either directly (Agent-to-Agent) or indirectly (through the environment).
- May require a specific (agent-communication) language.

An agent:

- is located in an environment (situatedness)
- **perceives** the environment through its **sensors**.
- **acts** upon that environment through its **effectors**.
- tends to maximize progress towards its goals by acting in the environment.



More details will be given during Seminar #2

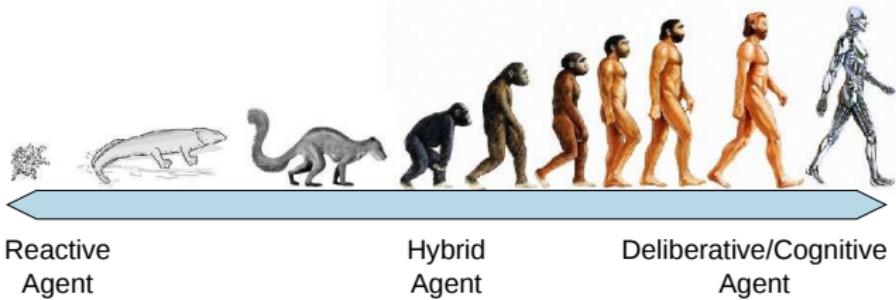
- **Mobility:** agent's ability to move through different nodes of a network/grid.
- **Adaptability:** ability to modify his actions/behavior according to external conditions and perceptions.
- **Versatility:** ability to perform different tasks or to meet different objectives.
- **Trustiness:** level of confidence that inspires the agent to delegate tasks, perform action, collaborate with other agents.
- **Robustness:** ability to continue to operate in fault situations, even with lower performances
- **Persistence:** Ability to keep continuously running by retrieving or saving their internal state even after a crash or unexpected situations.
- **Altruism:** disposition of an agent to assist other agents in their tasks.

### Agent [Ferber, 1999]

Agent is a virtual (software) or physical entity which:

- 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 or of a satisfaction/survival function which it tries to optimize);
- possesses resources of its own;
- is capable to perceive its environment (but up to a limited extent);
- has only a partial representation of this environment (and perhaps none at all);
- possesses skills and can offer services; Add a comment to this line
- may be able to reproduce itself;
- whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation and the communication it receives.

- **Reactive:** Each agent has a mechanism of **reaction to events**, without having an explanation/understanding of the objectives nor planning mechanisms. **Typical Example:** The ant colony.
- **Cognitive/Deliberative:** Each agent has a **knowledge** base that contains all information and skills necessary for the accomplishment of their **tasks/goals** and the management of his interactions with other agents and his environment: reasoning, planning, normative. **Typical Example:** Multi-Expert Systems.



- Agents are autonomous, they decide on the execution of services: “*Objects do it for free; agents do it because they want to*”.
- The agents allow flexible (reactive, pro-active, social) and autonomous behavior.
- Agent-based systems are inherently distributed and/or parallel (“multi-thread”). Each agent has its own resource of execution.

## Expert System Principle

An Expert System (ES) maintains various facts about the world that are used to draw **conclusions**.

## Key Differences

- Traditional ES are not situated in an environment. There is no direct coupling with the environment and requires a user that acts as an intermediary ue.
- ES are usually not capable of exhibiting flexible behavior (reactive and proactive).
- ES are usually not provided with social skills (communication and interaction with other agents).

### *Mono-agent approach*

- The system is composed of a single agent.
- Example: Personal Assistant

### *Multi-agent approach*

- The system is composed of multiple agents.
- The realization of global/collective task relies on a set of agents, on the composition of their actions.
- The solution emerges from the interactions of agents in an environment.

## Multiagent systems

An MultiAgent Systems (MAS) is a system composed of agents that interact together and through their environment.

### Interactions:

- Direct, agent to agent
- Indirect, Stigmergy, through the Environment

## Micro perspective (local): Agent

### Individual level

- Reactivity - Pro-activity
- Autonomy
- Delegation

## Macro perspective (global): Multiagent systems

### Society/Community level

- Distribution
- Decentralization (control and/or authority)
- Hierarchy
- Agreement technologies (coordination)
- Emergence, social order/pattern, norms

## Multiagent Systems [Ferber, 1999]

System comprising the following elements:

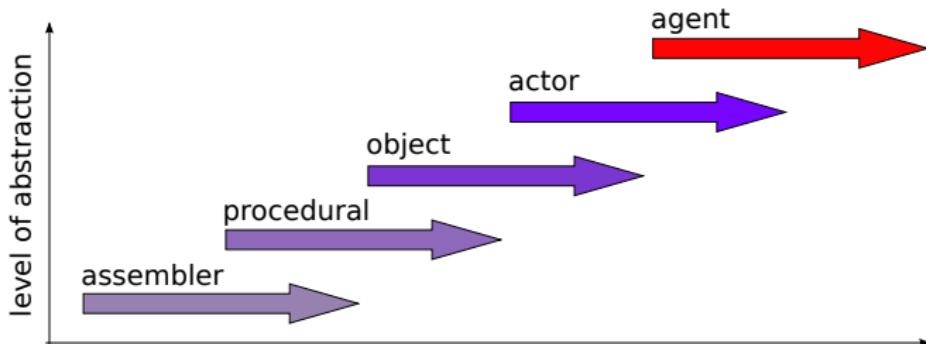
- An environment  $E$ , usually a space (with volume, 3D).
- An array of objects,  $O$ . These objects are situated.
- A set of agents,  $A$ , which are specific objects.
- A set of relations,  $R$ , which links the objects (and thus agents).
- A set of operations,  $Op$ , making it possible for agents to receive, produce, process and manipulate the objects in  $O$ .
- Operators with the task of representing the application of these operations and the reaction of the world to this attempt of modification.

## MAS' interests [Wooldridge, 2001]

- Natural metaphors
- Distribution of Data or Control
- Legacy Systems: Wrap/Encapsulate these systems into an agent for enabling interactions
- Open Systems requires flexible autonomous decision making
- Complex Systems

## Agent: a new paradigm ?

- Agent-Oriented Programming (AOP) reuses concepts and language artifacts from Actors and OOP.
- It also provides an higher-level abstraction than the other paradigms.



## Language

- All agents are holonic (recursive agents).
- There is not only one way of interacting but infinite.
- Event-driven interactions as the default interaction mode.
- Agent/environment architecture-independent.
- Massively parallel.
- Coding should be simple and fun.

## Execution Platform

- Clear separation between Language and Platform related aspects.
- Everything is distributed, and it should be transparent.
- Platform-independent.

Name	Domain	Hierar. <sup>a</sup>	Simu. <sup>b</sup>	C.Phys. <sup>c</sup>	Lang.	Beginners <sup>d</sup>	Free
GAMA	Spatial simulations		✓		GAML, Java	**[*]	✓
Jade	General		✓	✓	Java	*	✓
Jason	General		✓	✓	Agent-Speaks	*	✓
Madkit	General		✓		Java	**	✓
NetLogo	Social/natural sciences		✓		Logo	***	✓
Repast	Social/natural sciences		✓		Java, Python, .Net	**	
SARL	General	✓	✓ <sup>e</sup>	✓	SARL, Java, Xtend, Python	**[*]	✓

a Native support of hierarchies of agents.

b Could be used for agent-based simulation.

c Could be used for cyber-physical systems, or ambient systems.

d \*: experienced developers; \*\*: for Computer Science Students; \*\*\*: for others beginners.

e Ready-to-use Library:  Jaak Simulation Library

## Multiagent System in SARL

A collection of agents interacting together in a collection of shared distributed spaces.

### 4 main concepts

- Agent
- Capacity
- Skill
- Space

### 3 main dimensions

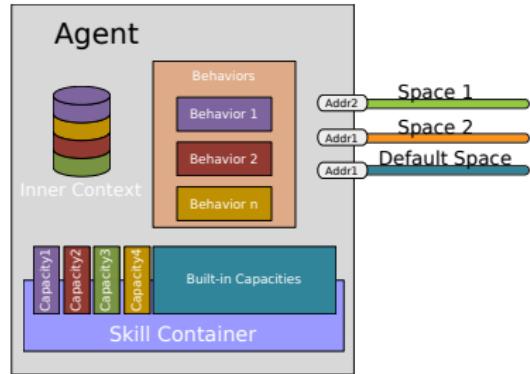
- **Individual:** the Agent abstraction (Agent, Capacity, Skill)
- **Collective:** the Interaction abstraction (Space, Event, etc.)
- **Hierarchical:** the Holon abstraction (Context)

**SARL: a general-purpose agent-oriented programming language.** Rodriguez, S., Gaud, N., Galland, S. (2014) Presented at the The 2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE Computer Society Press, Warsaw, Poland. [Rodriguez, 2014]

<http://www.sarl.io>

## Agent

- An agent is an autonomous entity having some intrinsic skills to implement the capacities it exhibits.
- An agent initially owns native capacities called **Built-in Capacities**.
- An agent defines a **Context**.



```
agent HelloAgent {  
    on Initialize {  
        println("Hello World!")  
    }  
    on Destroy {  
        println("Goodbye World!")  
    }  
}
```

```
package org.multiagent.example
```

```
agent HelloAgent {  
    var myvariable : int  
    val myconstant = "abc"  
  
    on Initialize {  
        println("Hello World!")  
    }  
  
    on Destroy {  
        println("Goodbye World!")  
    }  
}
```

The content of the file will be assumed to be in the given package.

```
package org.multiagent.example
```

```
agent HelloAgent {
```

```
    var myvariable : int  
    val myconstant = "abc"
```

Define the code of all  
the agents of type  
HelloAgent

```
    on Initialize {  
        println("Hello World!")  
    }
```

```
    on Destroy {  
        println("Goodbye World!")  
    }
```

```
}
```

```
package org.multiagent.example
```

```
agent HelloAgent {  
    var myvariable : int  
    val myconstant = "abc"
```

This block of code contains all the elements related to the agent.

```
    on Initialize {  
        println("Hello World!")  
    }
```

```
    on Destroy {  
        println("Goodbye World!")  
    }
```



```
package org.multiagent.example

agent HelloAgent {
    var myvariable : int
    val myconstant = "abc"

    on Initialize {
        println("Hello World!")
    }

    on Destroy {
        println("Goodbye World!")
    }
}
```

var myvariable : int  
val myconstant = "abc"

Define a variable with name "myvariable" and of type integer

```
package org.multiagent.example

agent HelloAgent {

    var myvariable : int
    val myconstant = "abc"

    on Initialize {
        println("Hello World!")
    }

    on Destroy {
        println("Goodbye World!")
    }
}
```

Define a constant with name "myconstant" and the given value.

```
package org.multiagent.example

agent HelloAgent {

    var myvariable : int
    val myconstant = "abc"

    on Initialize {
        println("Hello World!")
    }

    on Destroy {
        println("Goodbye World!")
    }
}
```

Execute the block of code when an event of type "Initialize" is received by the agent.

```
package org.multiagent.example

agent HelloAgent {

    var myvariable : int
    val myconstant = "abc"

    on Initialize {
        println("Hello World!")
    }

    on Destroy {
        println("Goodbye World!")
    }
}
```

Events predefined in the SARN language:  
- When initializing the agent  
- When destroying the agent

### Action

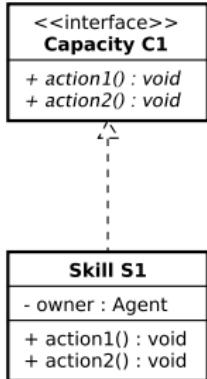
- A specification of a transformation of a part of the designed system or its environment.
- Guarantees resulting properties if the system before the transformation satisfies a set of constraints.
- Defined in terms of pre- and post-conditions.

### Capacity

Specification of a collection of actions.

### Skill

A possible implementation of a capacity fulfilling all the constraints of its specification, the capacity.



Enable the separation between a generic behavior and agent-specific capabilities.

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skill BasicConsoleLogging  
implements Logging {  
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }  
  
    def info(s : String) {  
        println("INFO:" + s)  
    }  
}
```

```
agent HelloAgent {  
    uses Logging  
  
    on Initialize {  
        setSkill( Logging,  
                  new BasicConsoleLogging)  
        info("Hello World!")  
    }  
}
```

Definition of a capacity that permits to an agent to print messages into the log system.

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skill BasicConsoleLogging  
implements Logging {  
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }  
  
    def info(s : String) {  
        println("INFO:" + s)  
    }  
}
```

```
agent HelloAgent {  
    uses Logging  
  
    on Initialize {  
        setSkill( Logging,  
                  new BasicConsoleLogging)  
        info("Hello World!")  
    }  
    info("Hello World!")  
}
```

Define a function that could be invoked by the agent.

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skill BasicConsoleLogging  
implements Logging {  
  
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }  
  
    def info(s : String) {  
        println("INFO:" + s)  
    }  
}
```

Define the skill that implements the Logging capacity.

```
uses Logging  
  
on Initialize {  
    setSkill( Logging,  
              new BasicConsoleLogging)  
    info("Hello World!")  
}  
  
on Destroy {  
    info("Goodbye World!")  
}
```

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}
```

```
skill BasicConsoleLogging  
implements Logging {  
  
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }  
  
    def info(s : String) {  
        println("INFO:" + s)  
    }  
}
```

Every function declared into the implemented capacity must be implemented in the skill. The current implementations output the message onto the standard output stream.

```
    info("Hello World!")  
}  
  
on Destroy {  
    info("Goodbye World!")  
}  
}
```

```
capacity  
def info(s : String)
```

```
}
```

```
skill BasicConsoleLogging  
implements Logging {
```

```
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }
```

```
    def info(s : String) {  
        println("INFO:" + s)  
    }
```

```
}
```

The use of a capacity into the agent code is enabled by the "uses" keyword.

```
agent HelloAgent {  
    uses Logging  
  
    on Initialize {  
        setSkill( Logging,  
                  new BasicConsoleLogging )  
        info("Hello World!")  
    }  
  
    on Destroy {  
        info("Goodbye World!")  
    }  
}
```

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skip  
import java.util.logging.Logger  
import java.util.logging.Level  
import java.util.logging.ConsoleHandler  
import java.util.logging.SimpleFormatter  
import org.jtcl.core.JTCL  
  
d  
    println("DEBUG:" + s)  
}  
  
def info(s : String) {  
    println("INFO:" + s)  
}  
  
}
```

All functions defined into the used capacities are directly callable from the source code.

```
agent HelloAgent {  
    uses Logging  
  
    on Initialize {  
        setSkill( Logging,  
                  new BasicConsoleLogging()  
        info("Hello World!")  
    }  
  
    on Destroy {  
        info("Goodbye World!")  
    }  
}
```

```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skill Logging {  
    implements Logging {  
        def info(s : String) {  
            println("INFO:" + s)  
        }  
    }  
}
```

An agent MUST specify the skill to use for a capacity (except for the buildin skills that are provided by the execution framework)

```
agent HelloAgent {  
    uses Logging  
  
    on Initialize {  
        setSkill( Logging,  
                  new BasicConsoleLogging)  
        info("Hello World!")  
    }  
  
    on Destroy {  
        info("Goodbye World!")  
    }  
}
```

## Space

Support of interaction between agents respecting the rules defined in various Space Specifications.

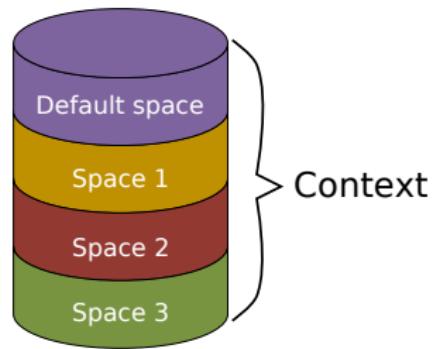
## Space Specification

- Defines the rules (including action and perception) for interacting within a given set of Spaces respecting this specification.
- Defines the way agents are addressed and perceived by other agents in the same space.
- A way for implementing new interaction means.

The spaces and space specifications must be written with the Java programming language

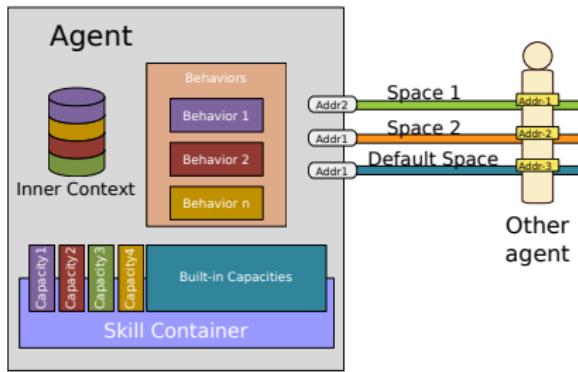
## Context

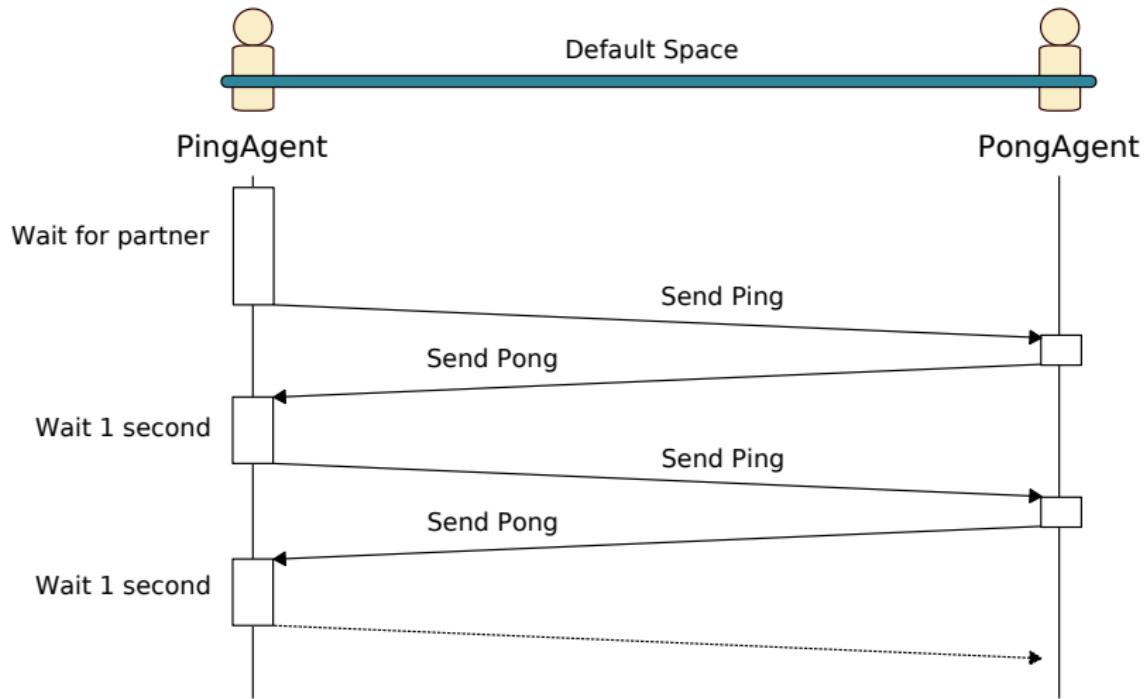
- Defines the boundary of a sub-system.
- Collection of Spaces.
- Every Context has a **Default Space**.
- Every Agent has a **Default Context**, the context where it was spawned.



### Default Space: an Event Space

- Event-driven interaction space.
- Default Space of a context, contains all agents of the considered context.
- Event: the specification of some **occurrence** in a Space that may potentially trigger effects by a participant.





```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for Ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

Definition of an event for the PING.  
It contains the ID of the Ping message  
in its "value" field.  
The field is initialized in the constructor  
of the event.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.participants.size > 1) {
            (new Ping(count))
            count = count + 1
        }
        in(2000) [ sendPing ]
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {  
    var value : Integer  
    new (v : Integer) {  
        value = v  
    }  
}  
  
event Pong {  
    var value : Integer  
    new (v : Integer) {  
        value = v  
    }  
}  
  
agent PongAgent {  
    uses DefaultContextInteractions  
    on Initialize {  
        println("Waiting for ping")  
    }  
    on Ping {  
        println("Recv Ping: "+occurrence.  
            value)  
        println("Send Pong: "+occurrence.  
            value)  
        emit(new Pong(occurrence.value))  
    }  
}
```

Definition of an event for the PONG.  
It contains the ID of the Ping message  
in its "value" field.  
The field is initialized in the constructor  
of the event.

Interactions

PingAgent...")

```
    count = 0  
    in(2000) [ sendPing ]  
}  
def sendPing {  
    if (defaultSpace.  
        participants.size > 1) {  
        emit(new Ping(count))  
        count = count + 1  
    } else {  
        in(2000) [ sendPing ]  
    }  
}  
on Pong {  
    in(1000) [  
        println("Send Ping: "+count)  
        emit(new Ping(count))  
        count = count + 1  
    ]  
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PingAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

The PingAgent is waiting for a partner.  
Check every 2 seconds if a partner  
is available.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {  
    var value : Integer  
    new (v : Integer) {  
        value = v  
    }  
}  
  
event Pong {  
    var value : Integer  
    new (v : Integer) {  
        value = v  
    }  
}  
  
agen  
us  
on  
}  
on  
}  
  
Special syntax:  
fct(param1, param2, ...) [ code ]  
  
Call the function "fct" with the given  
parameters AND  
a piece of code as the last parameter.  
  
The function is supposed to execute the  
piece of code according to its semantic.
```

```
agent PingAgent {  
    uses Schedules  
    uses DefaultContextInteractions  
    var count : Integer  
    on Initialize {  
        println("Starting PingAgent...")  
        count = 0  
        in(2000) [ sendPing ]  
    }  
    def sendPing {  
        if (defaultSpace.  
            participants.size > 1) {  
            emit(new Ping(count))  
            count = count + 1  
        } else {  
            in(2000) [ sendPing ]  
        }  
    }  
    on Pong {  
        in(1000) [  
            println("Send Ping: "+count)  
            emit(new Ping(count))  
            count = count + 1  
        ]  
    }  
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
} sendPing is the function that tries to
contact the partner for the first time.
Run this function in 2 seconds.
The in() function is provided by the
"Schedules" built-in capacity.
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.
            value)
        println("Send Pong: "+occurrence.
            value)
        emit(new Pong(occurrence.value))
    }
}
```

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}
event v
n
}
Test if the number of participants
in the default space is greater than 1
(1 when PingAgent, 2 when PingAgent
and PongAgent are inside).
}
The defaultSpace is accessible via the
built-in capacity
"DefaultContextInteraction".
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.
            value)
        println("Send Pong: "+occurrence.
            value)
        emit(new Pong(occurrence.value))
    }
}
```

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}
event PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
                emit(new Ping(count))
                count = count + 1
            } else {
                in(2000) [ sendPing ]
            }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

When the PongAgent is available, the PingAgent sends the first Ping message.

The count variable is defined for keeping track of the ID of the messages.

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Ping {
        println("Recv Ping: "+occurrence.
            value)
        println("Send Pong: "+occurrence.
            value)
        emit(new Pong(occurrence.value))
    }
}
```

When the PongAgent is NOT AVAILABLE  
the PingAgent waits 2 seconds before  
checking again if the PongAgent is  
alive.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.
            participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

Definition of the PongAgent with:  
- a constructor.  
- a event handler on the Ping messages.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        tln("Starting PingAgent...")
        count = 0
    }
    in(2000) [ sendPing ]
}
def sendPing {
    if (defaultSpace.participants.size > 1) {
        emit(new Ping(count))
        count = count + 1
    } else {
        in(2000) [ sendPing ]
    }
}
on Pong {
    in(1000) [
        println("Send Ping: "+count)
        emit(new Ping(count))
        count = count + 1
    ]
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

Declare an event handler:  
a piece of that is run when a message  
of type "Ping" is received.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        [ sendPing ]
    }
    on Ping {
        in(1000) [
            participants.size > 1) {
                emit(new Ping(count))
                count = count + 1
            } else {
                in(2000) [ sendPing ]
            }
        }
        on Pong {
            in(1000) [
                println("Send Ping: "+count)
                emit(new Ping(count))
                count = count + 1
            ]
        }
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

Instructions to execute when a message  
of type "Ping" is received.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    var count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
        in(2000) [ sendPing ]
    }
    def sendPing {
        if (defaultSpace.participants.size > 1) {
            Ping(count)
            count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent PongAgent {
    uses DefaultContextInteractions
    on Initialize {
        println("Waiting for ping")
    }
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

Emit a Pong in the default space of the default context.

The Pong message has the same value of the received Ping event.

"occurrence" is the variable that contains the received Ping event.

```
agent PingAgent {
    uses Schedules
    uses DefaultContextInteractions
    count : Integer
    on Initialize {
        println("Starting PingAgent...")
        count = 0
    }
    on sendPing {
        defaultSpace.
        participants.size > 1) {
            emit(new Ping(count))
            count = count + 1
        } else {
            in(2000) [ sendPing ]
        }
    }
    on Pong {
        in(1000) [
            println("Send Ping: "+count)
            emit(new Ping(count))
            count = count + 1
        ]
    }
}
```

```
event Ping {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

event Pong {
    var value : Integer
    new (v : Integer) {
        value = v
    }
}

agent us on occurrence {
    println("Waiting for ping")
    on Ping {
        println("Recv Ping: "+occurrence.value)
        println("Send Pong: "+occurrence.value)
        emit(new Pong(occurrence.value))
    }
}
```

When the PingAgent is receiving a Pong message, it waits 1 second and sends a Ping message with an incremented ID.

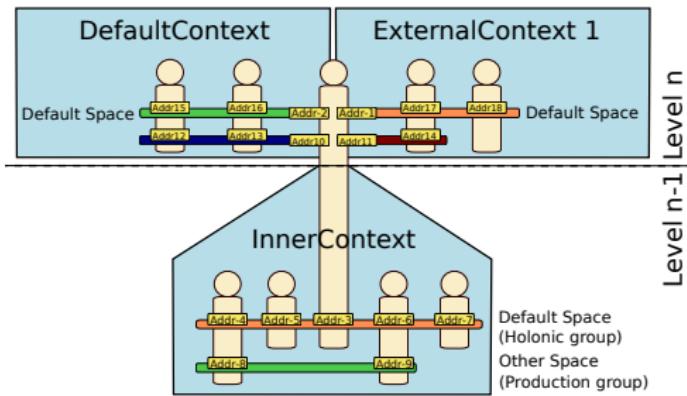
```
uses Schedules
uses DefaultContextInteractions
var count : Integer
on Initialize {
    println("Starting PingAgent...")
    count = 0
    in(2000) [ sendPing ]
}
def sendPing {
    if (defaultSpace.participants.size > 1) {
        emit(new Ping(count))
        count = count + 1
    } else {
        in(2000) [ sendPing ]
    }
}
on Pong {
    in(1000) [
        println("Send Ping: "+count)
        emit(new Ping(count))
        count = count + 1
    ]
}
```

## Contexts and Holonic properties

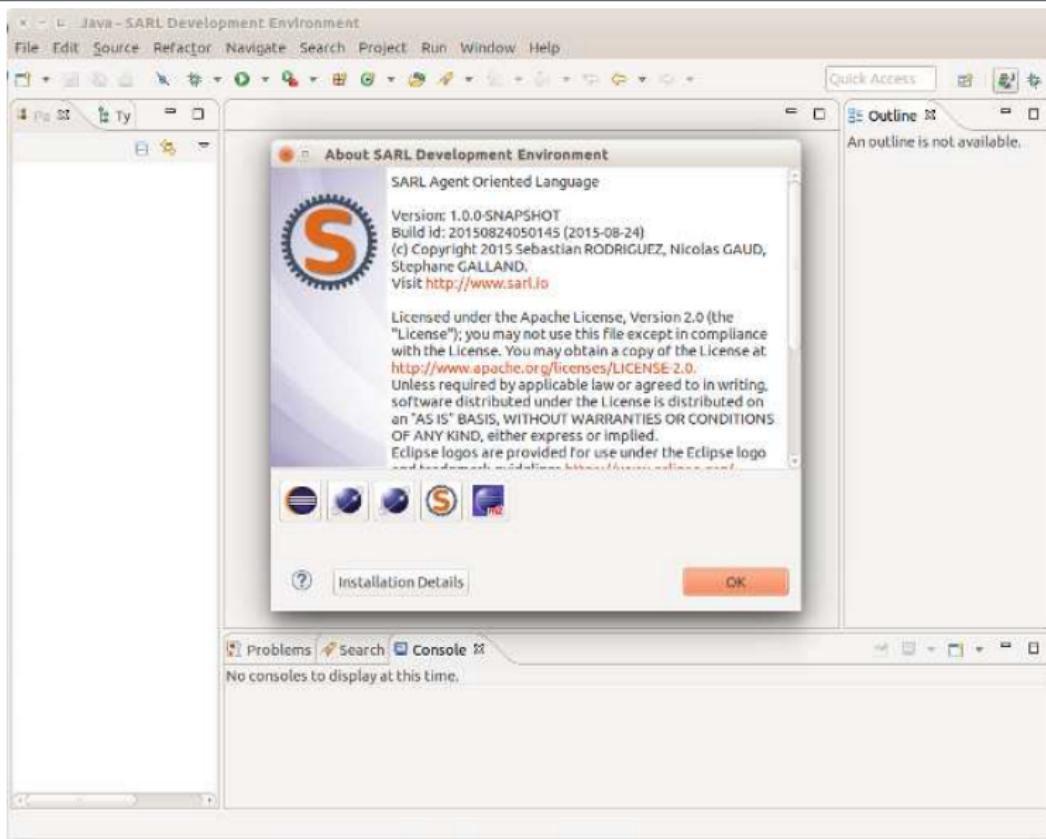
- All agents have at least one context: the default one.
- All agents participate in the default space of all contexts they belong to.

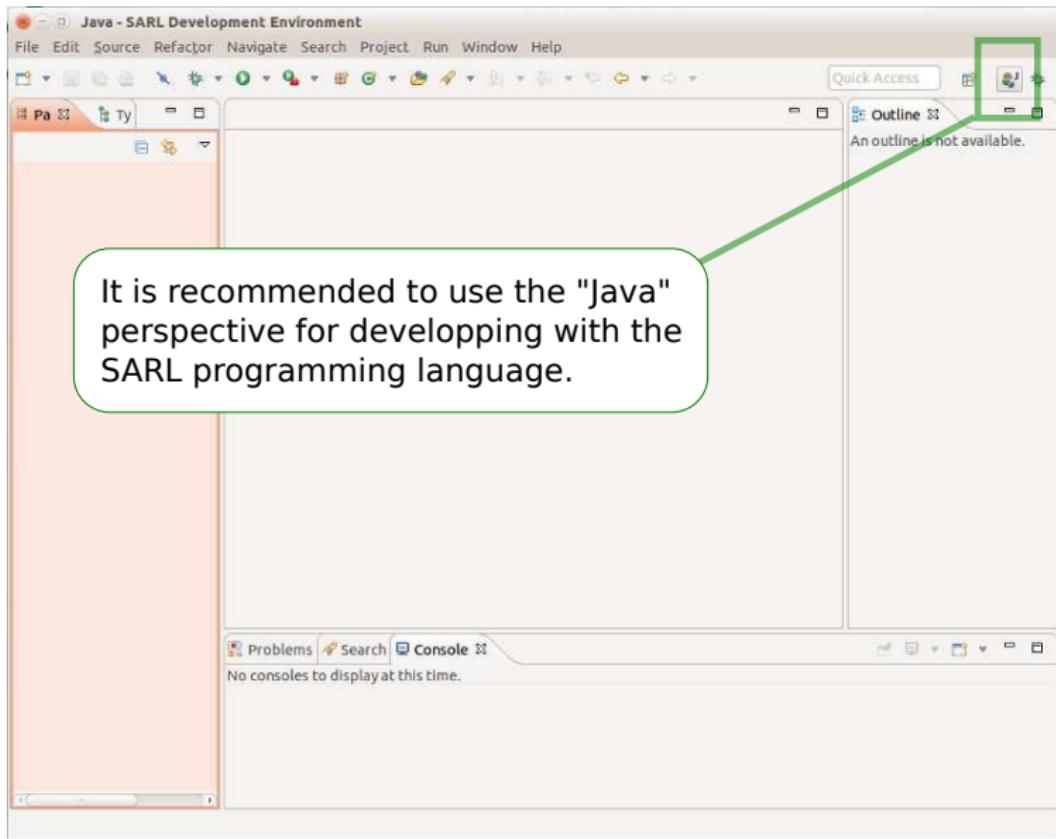
## Holonic Perspective

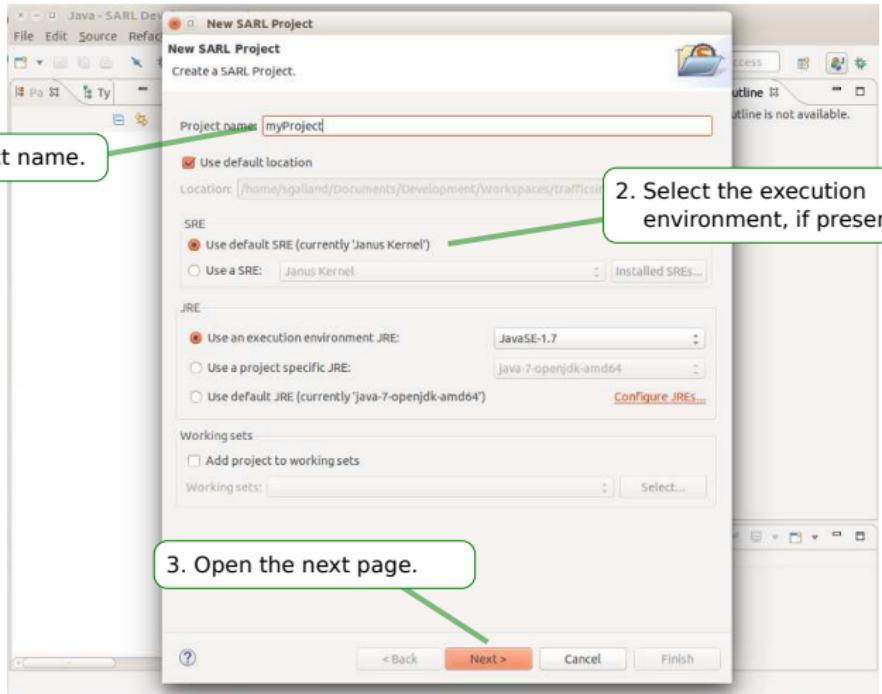
- All agents contains an internal context.
- Enable to build hierarchy of agents: holarchy.

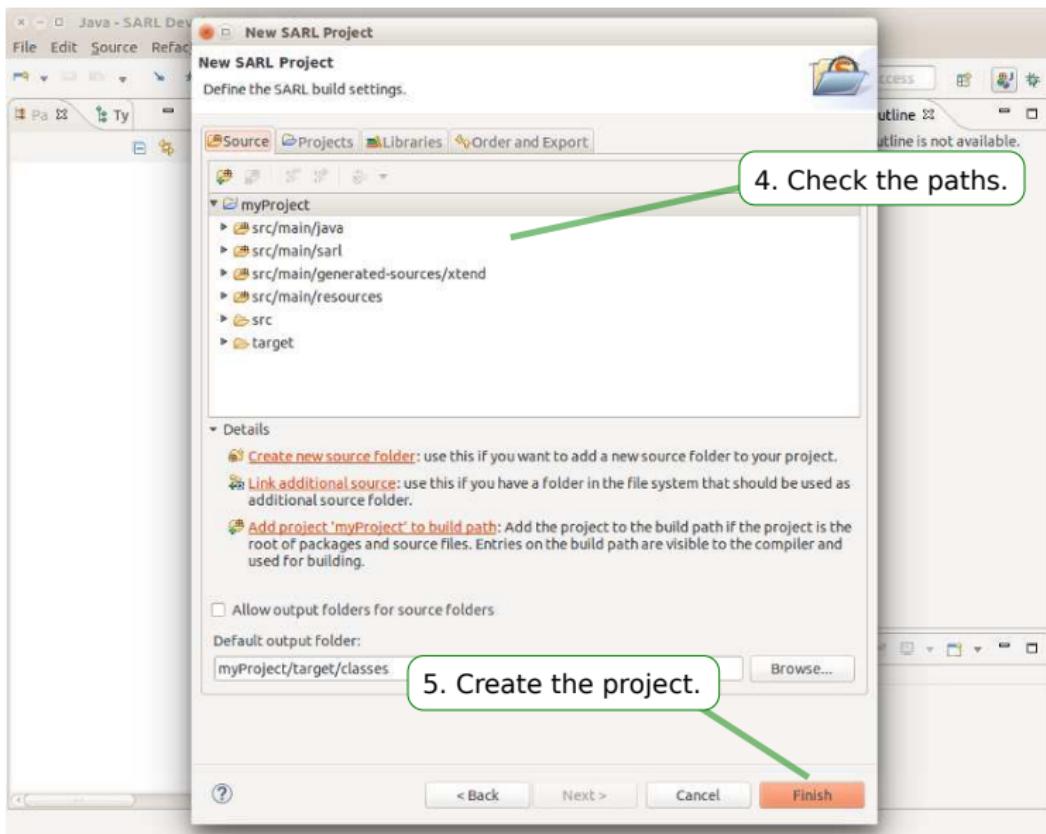


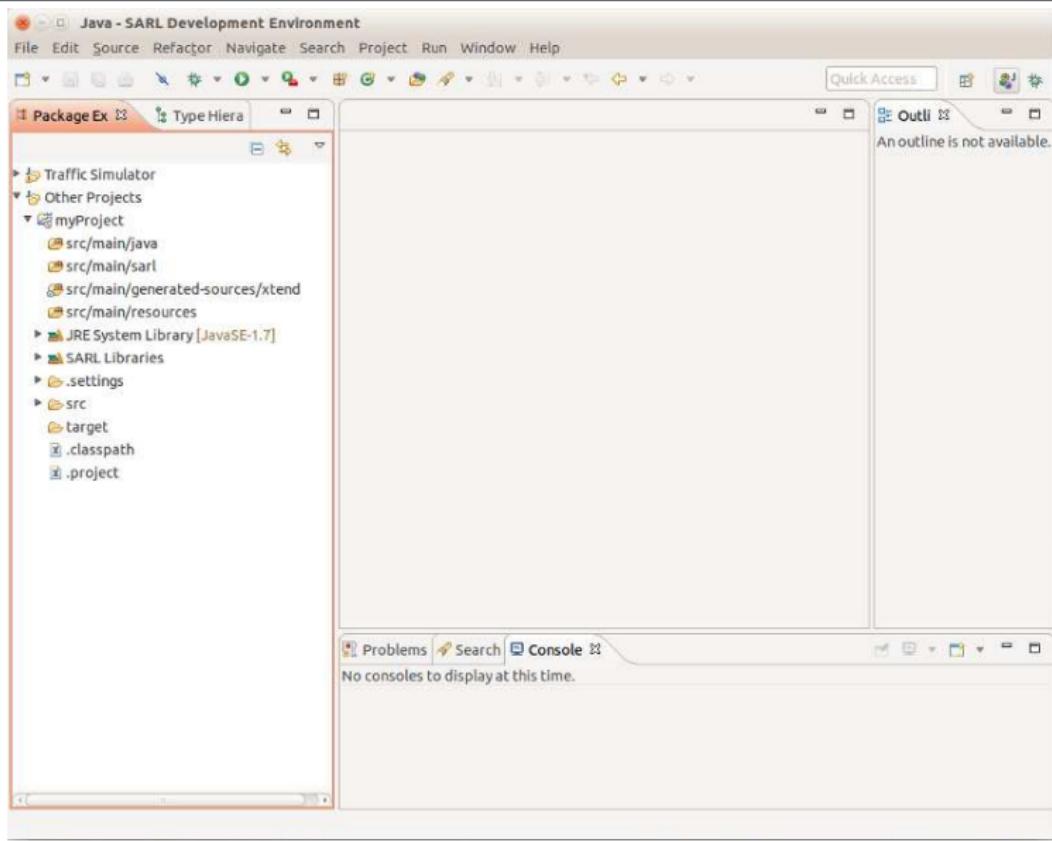
More details will be given during Seminar #3

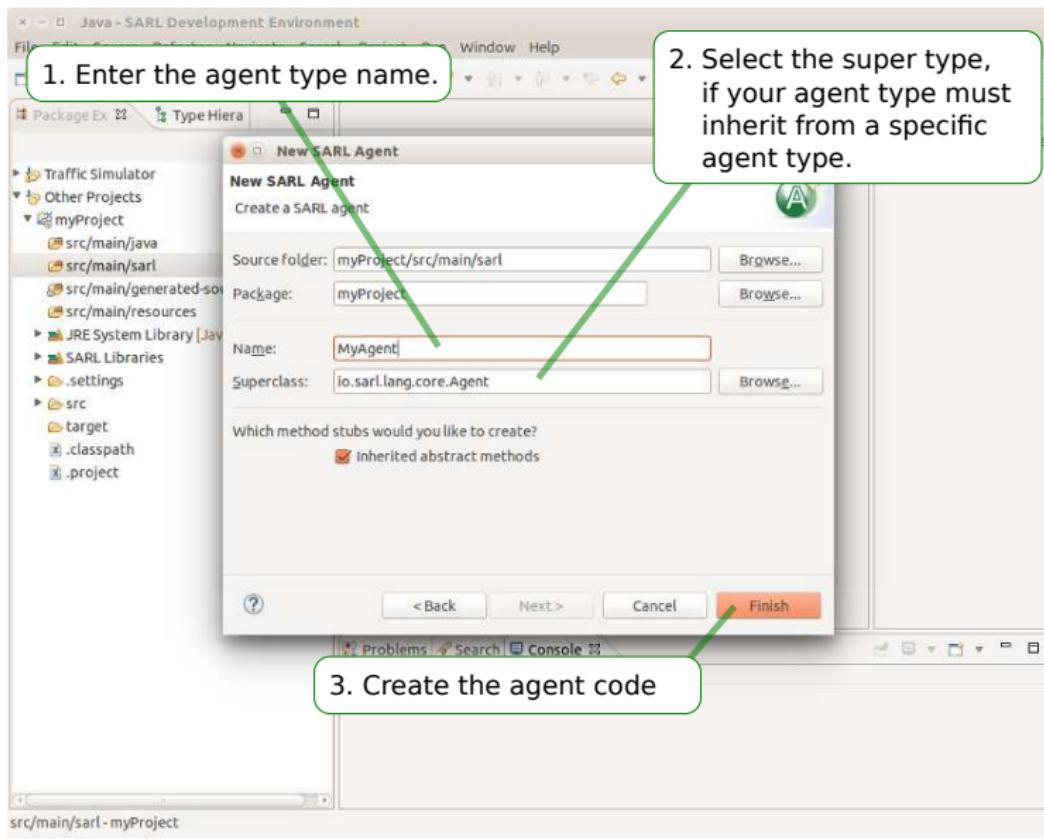


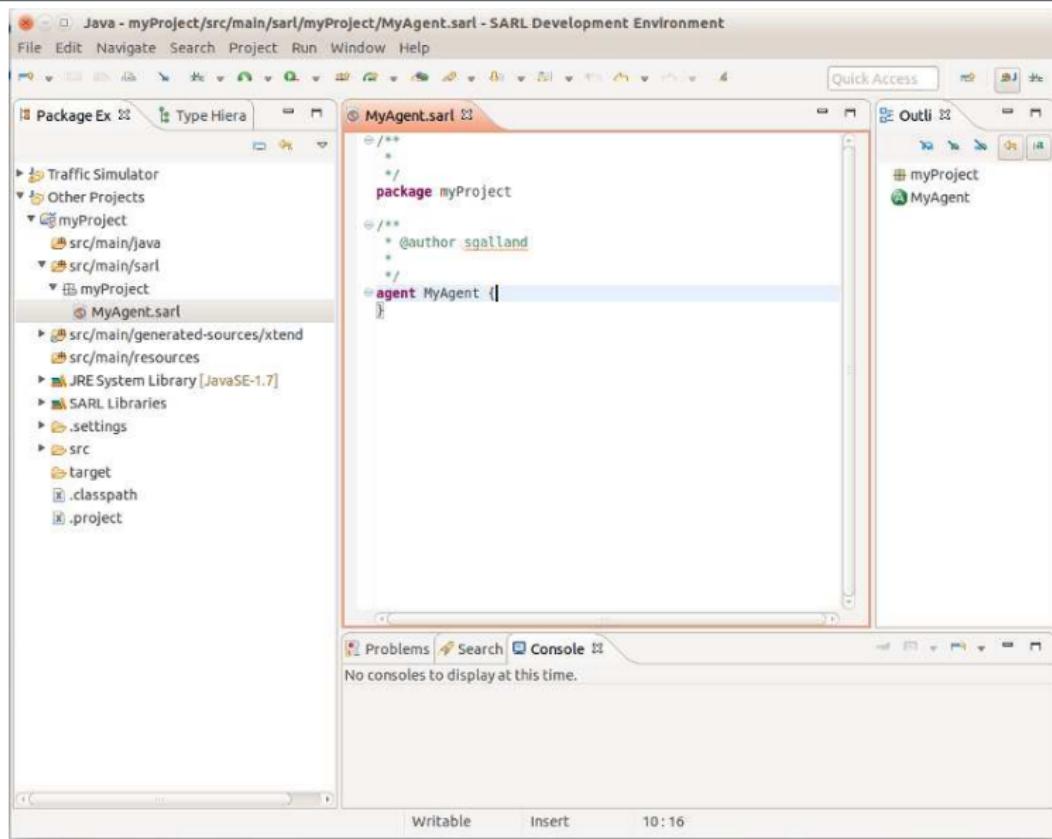


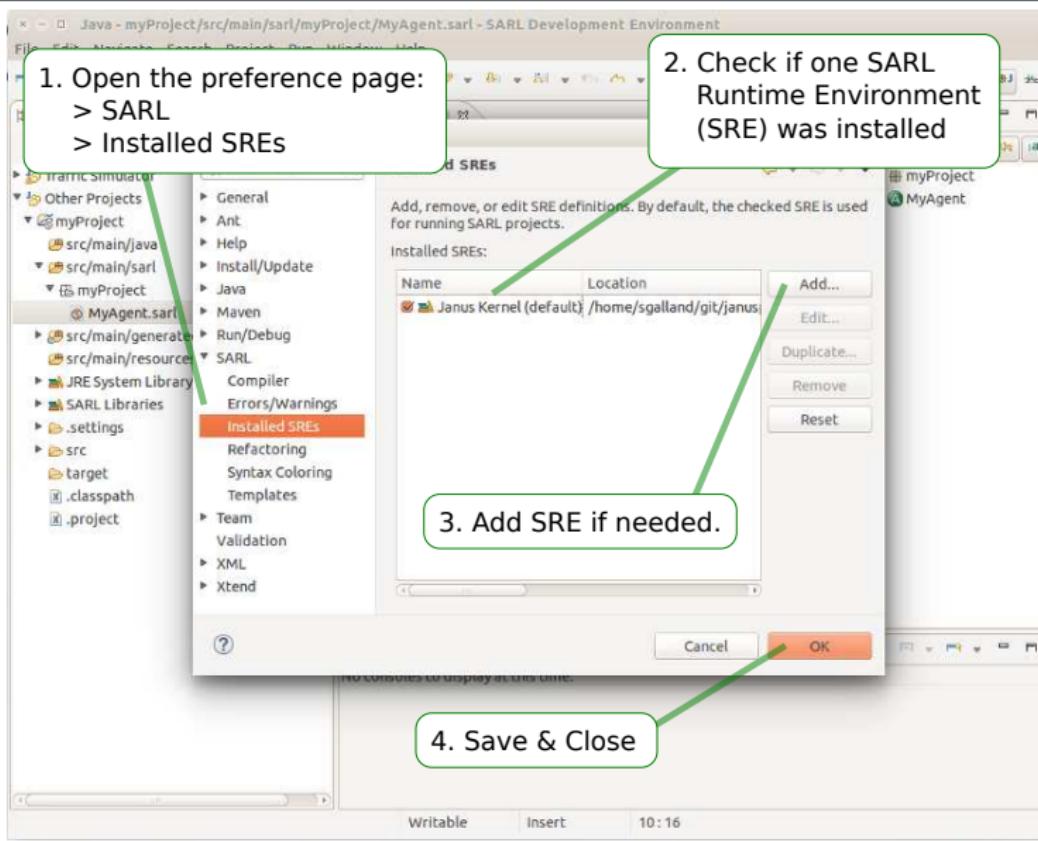


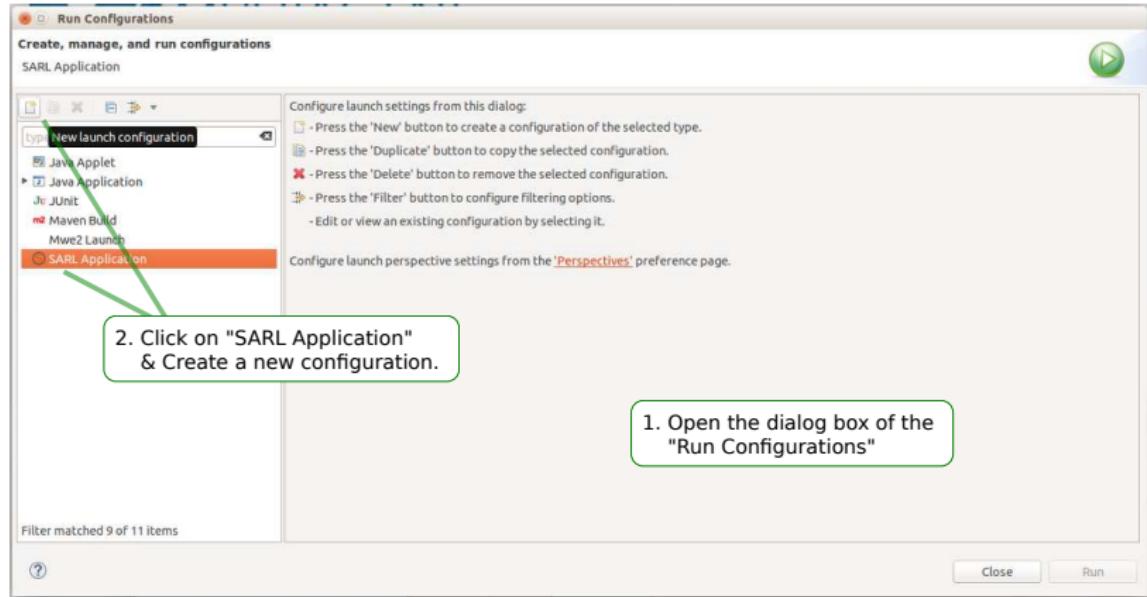


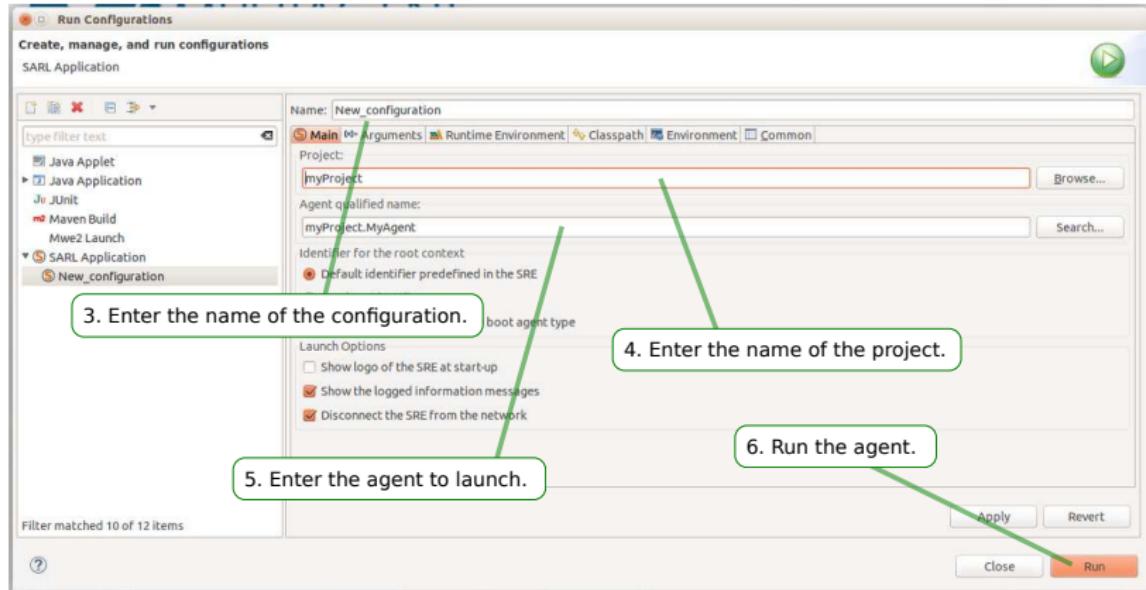












## Runtime Environment Requirements

- Implements SART concepts.
- Provides Built-in Capacities.
- Handles Agent's Lifecycle.
- Handles resources.

## Janus as a SART Runtime Environment

- Fully distributed.
- Dynamic discovery of Kernels.
- Automatic synchronization of kernels' data (easy recovery).
- Micro-Kernel implementation.
- Official website: <http://www.janusproject.io>



Other SREs may be created. See Seminar #4.

## 1 Introduction to Multiagent Systems

## 2 Multiagent Simulation

- Simulation Fundamentals
- Multiagent Based Simulation (MABS)
- Overview of a MABS Architecture
- Agent Architectures

## 3 Simulation with a Physic Environment

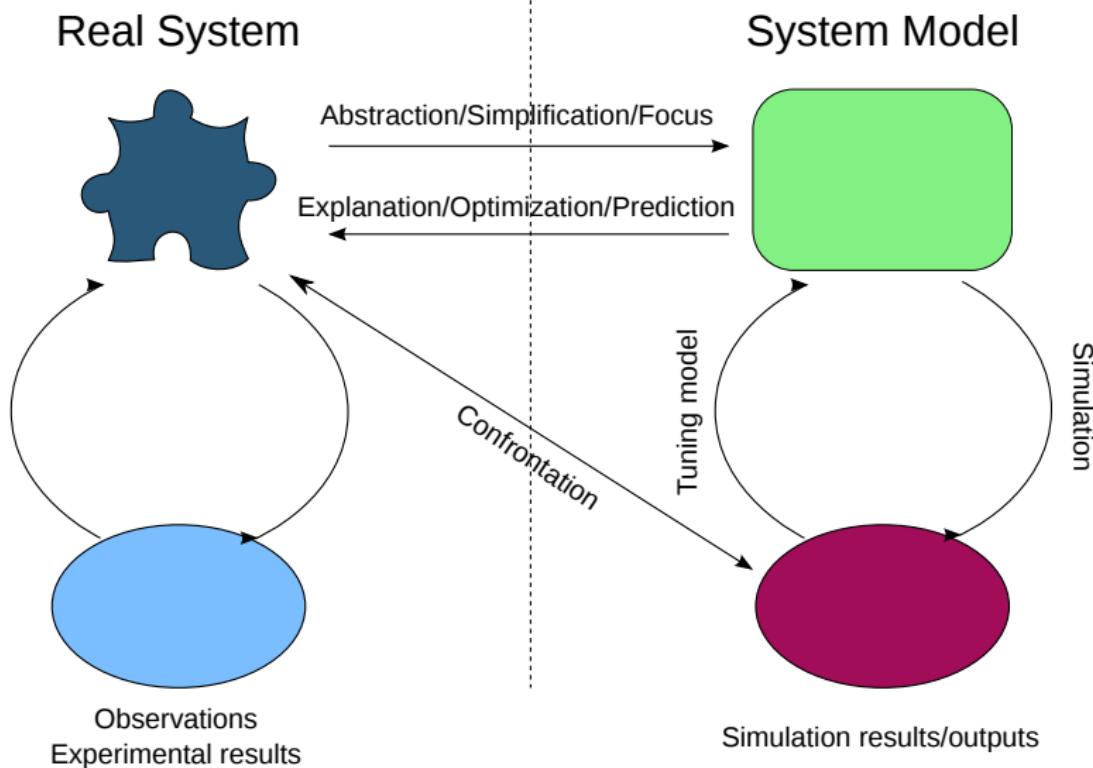
## 4 Cyber-physical System

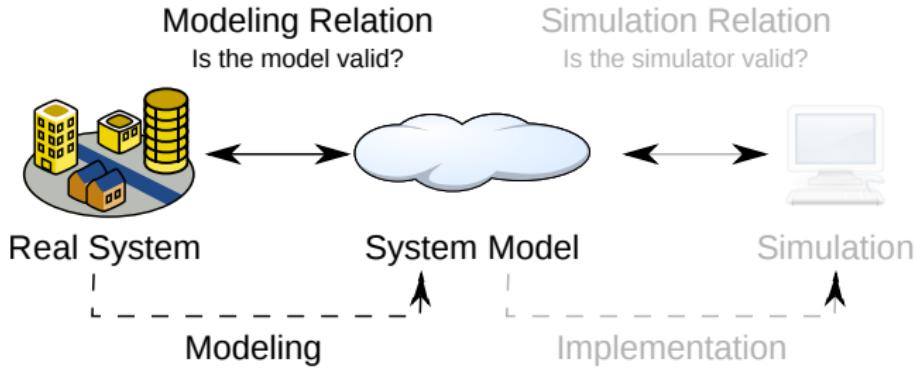
[Shannon, 1977]

The process of **designing a model** of a real system and **conducting experiments** with this model for the purpose either of **understanding** the behavior of the system or of **evaluating** various strategies (within the limits imposed by a criterion or a set of criteria) for the operation of the system.

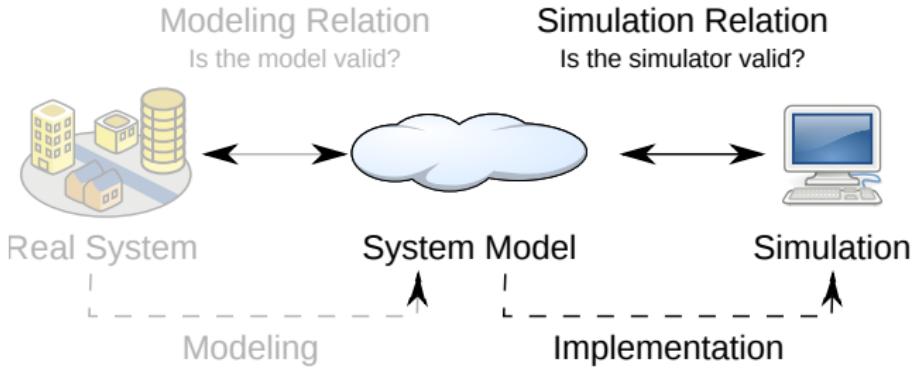
### Why simulate?

- Understand / optimize a system.
- Scenarii/strategies evaluation, testing hypotheses to explain a phenomenon (decision-helping tool).
- Predicting the evolution of a system, e.g. metrology.





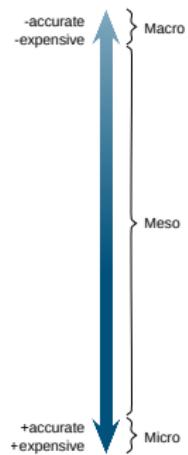
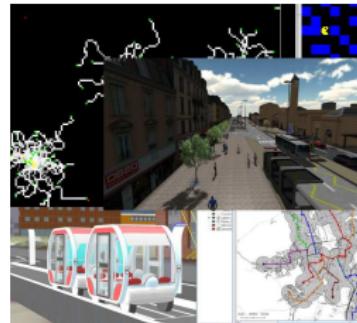
- To determine if the system model is an acceptable simplification in terms of quality criteria and experimentation objectives.
- Directly related to the consistency of the model simulation.



- To guarantee that the simulator, used to implement the model, correctly generates the behavior of the model.
- To be sure that the simulator reproduces clearly the mechanisms of change of state are formalized in the model.

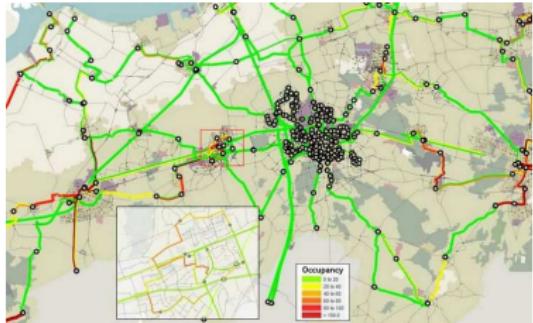
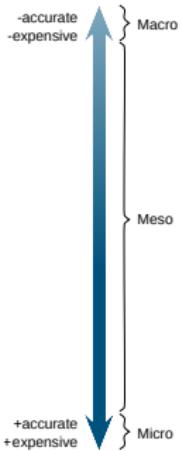
## Microscopic Simulation

- Explicitly attempts to model the behaviors of each individual.
- The system structure is viewed as emergent from the interactions between the individuals.



## Mesoscopic Simulation

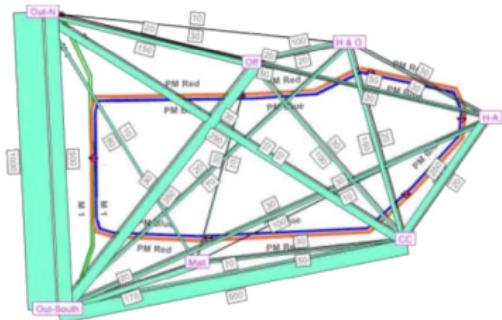
- Based on small groups, within which elements are considered homogeneous.
- Examples: vehicle platoon dynamics and household-level travel behavior.



[Hoogendoorn, 2001, Davidsson, 2000]

## Macroscopic Simulation

- Based on mathematical models, where the characteristics of a population are averaged together.
  - Simulate changes in these averaged characteristics for the whole population.
  - The set of individuals is viewed as a structure that can be characterized by a number of variables.

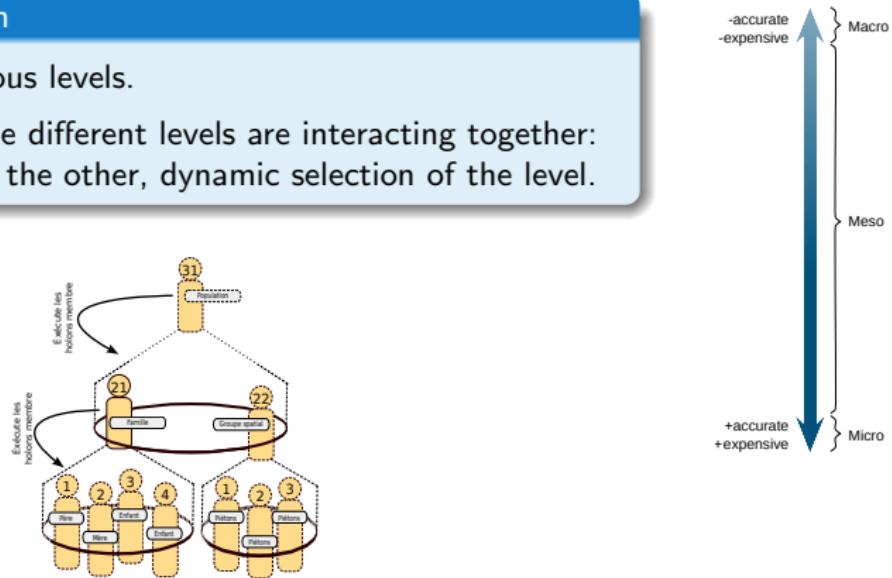


A vertical blue arrow pointing downwards, representing a hierarchy from Macro to Micro. The arrow is labeled with curly braces on the right side indicating levels: Macro at the top, Meso in the middle, and Micro at the bottom. At the top left, there is a label '-accurate -expensive'. At the bottom left, there is a label '+accurate +expensive'.

[Hoogendoorn, 2001, Davidsson, 2000]

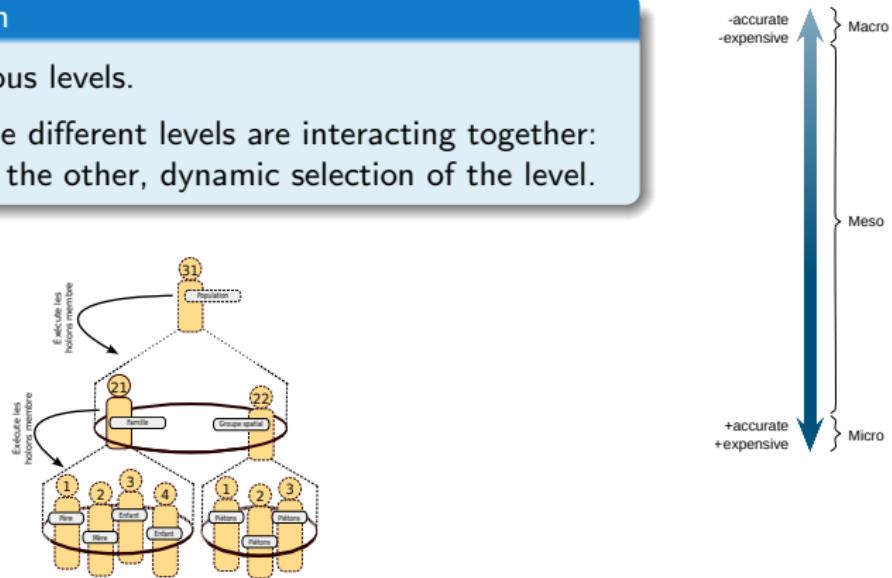
## Multilevel Simulation

- Combines various levels.
- Specify how the different levels are interacting together: one is input of the other, dynamic selection of the level.



## Multilevel Simulation

- Combines various levels.
- Specify how the different levels are interacting together: one is input of the other, dynamic selection of the level.



Multiagent-based Simulation (MABS), aka. ABS, is traditionnally considered as a special form of microscopic simulation, but not restricted to.

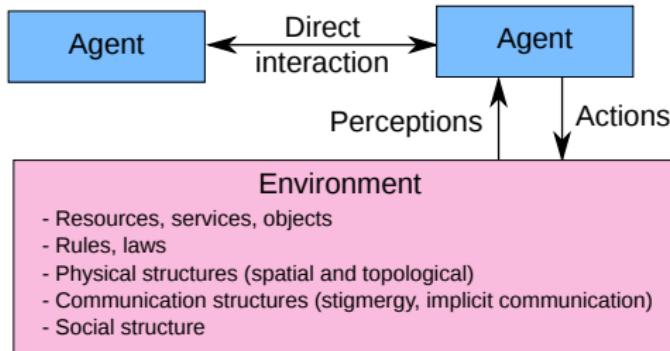
- Create an artificial world composed of interacting agents.
- The behavior of an agent results from:
  - its **perceptions/observations**;
  - its internal **motivations/goals/beliefs/desires**;
  - its eventual representations;
  - its **interaction** with the environment (indirect interactions, ressources) and the other agents (communications, direct interactions, stimuli).
- Agents act and modify the state of the environment through their actions.
- We observe the results of the interactions like in a Virtual Lab  
⇒ Emergence.

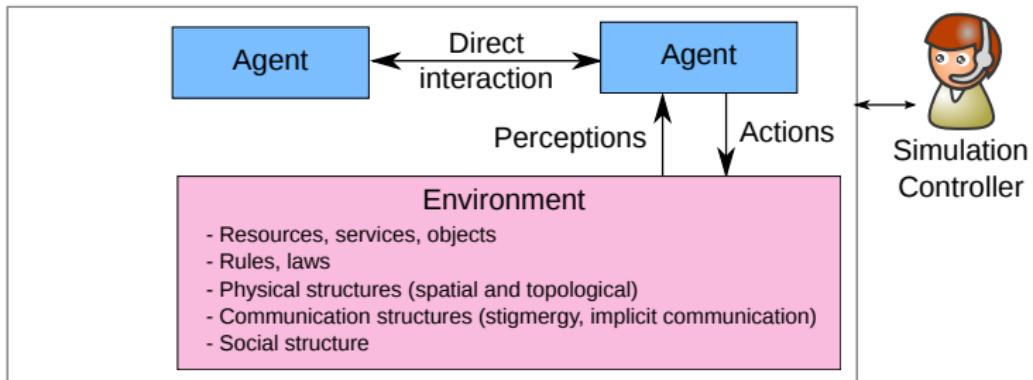
- More flexible than macroscopic models to simulate spatial and evolutionary phenomena.
- Dealing with real multiagent systems directly:  
real Agent = simulated Agent.
- Allows modelling of adaptation and evolution.
- Heterogeneous space and population.
- Multilevel modeling: integrate different levels of observation, and of agent's behaviors.

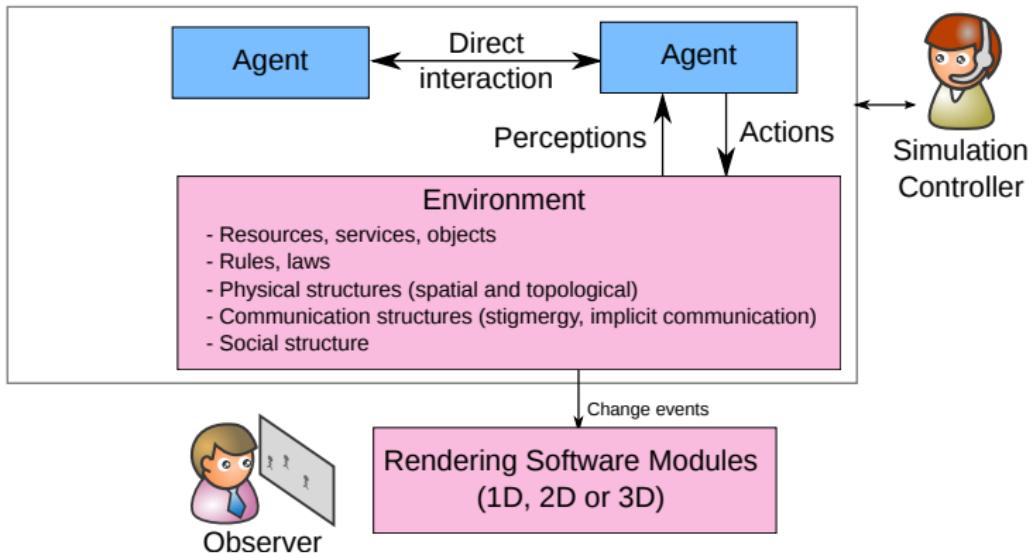


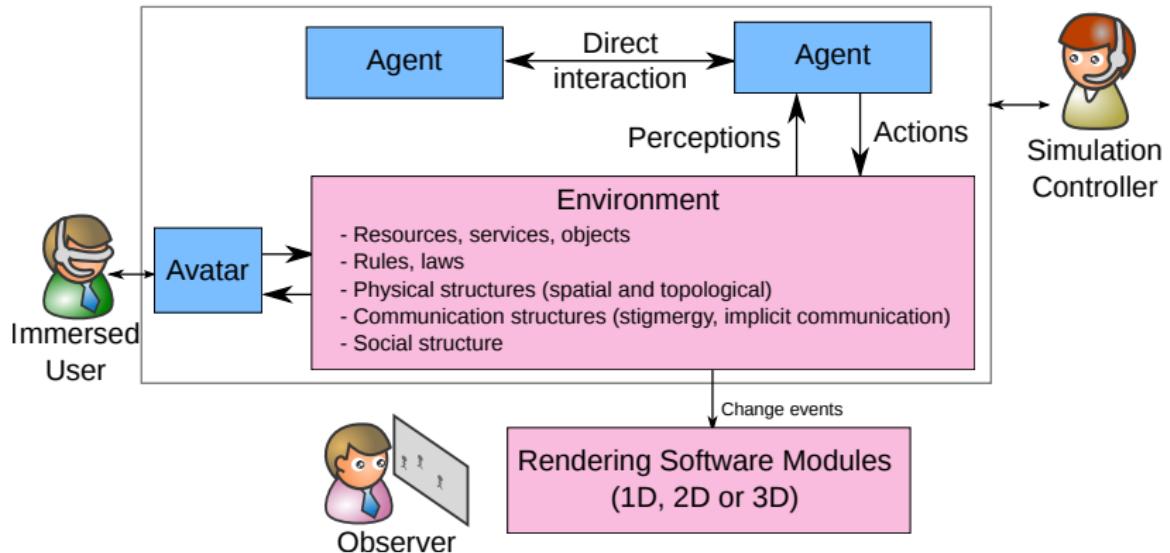
- Offer a significant level of accuracy at the expense of a larger computational cost.
- Require many and accurate data for their initialization.
- It is difficult to apply to large scale systems.
- Actual simulation models are costly in time and effort.

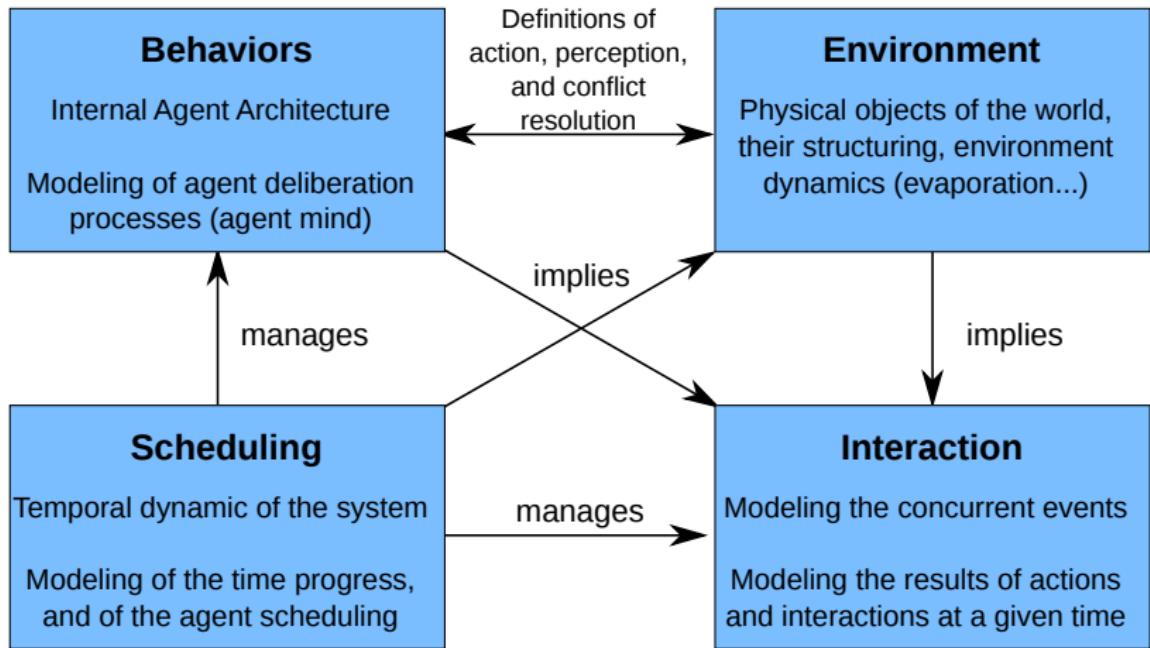




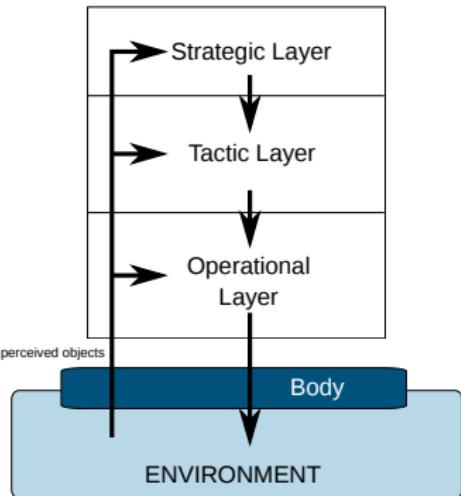






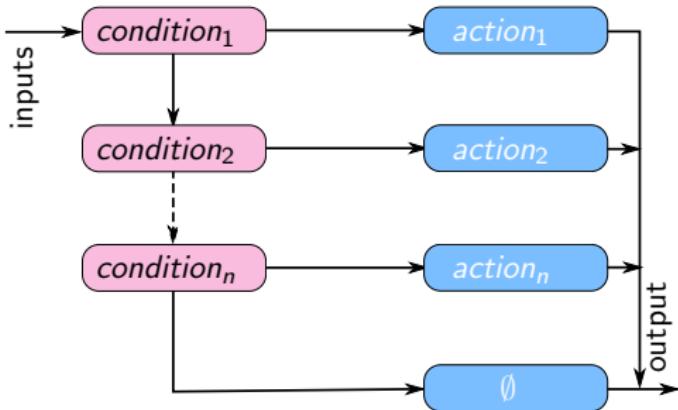


- **Strategic Layer:** general planning stage that includes the determination of goals, the route and the modal choice as well as a cost-risk evaluation.
- **Tactic Layer:** Maneuvering level behavior. Examples: obstacle avoidance, gap acceptance, turning, and overtaking.
- **Operational Layer:** Fundamental body controlling processes such as controlling speed, following the path, etc.



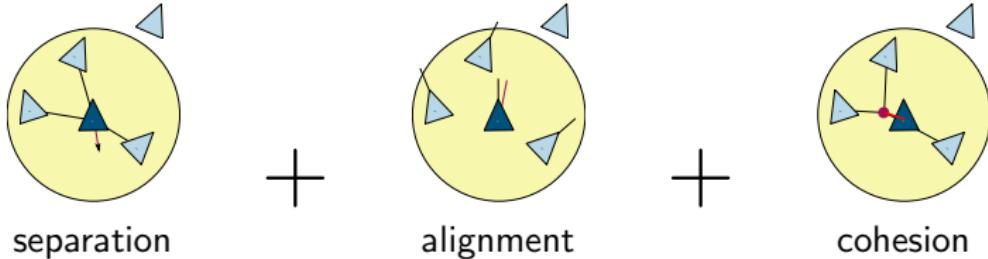
- Priority-ordering sequence of condition-action pairs.
- Generalization of the three-layer architecture: each pair is a layer.

```
if condition1 then  
    action1  
else  
    if condition2 then  
        action2  
    else  
        ...  
    end if  
end if
```

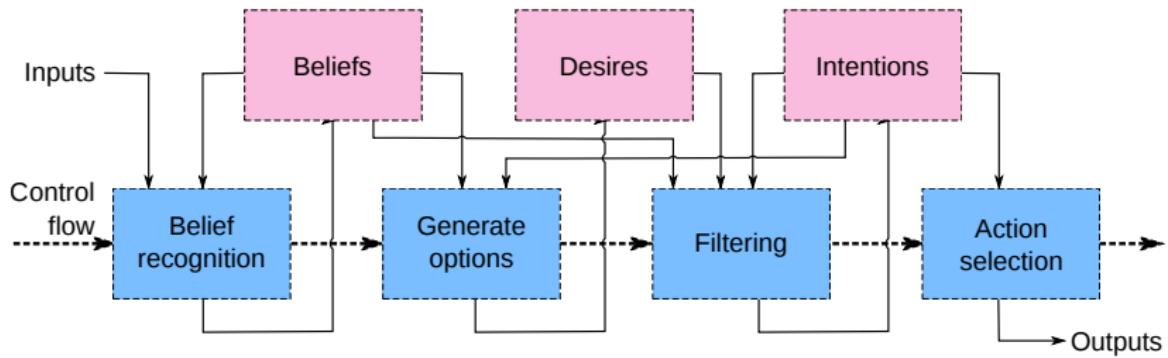


- Repulsive forces are computed and summed for obtaining the safer direction.
- May contribute to the two lower layers of the three-layer architecture.

$$\vec{f} = \left( \sum_{i=1}^n \frac{\alpha_i \cdot \widehat{\vec{p} - \vec{a}_i}}{|\vec{p} - \vec{a}_i|} \right) + \beta \cdot \left( \frac{\sum_{i=1}^n \vec{a}_i}{n} - \vec{p} \right)$$

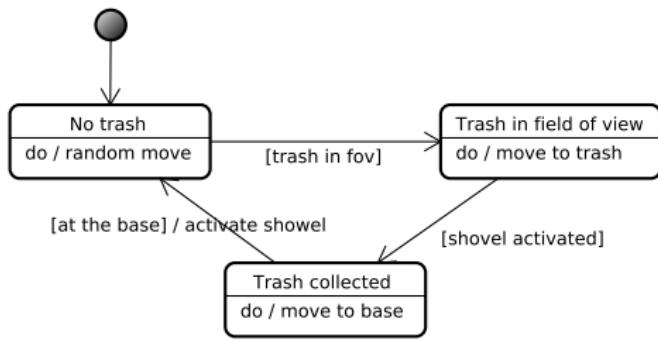


- **Beliefs:** the informational state of the agent, in other words its beliefs about the world (including itself and other agents).
- **Goals:** a desire that has been adopted for active pursuit by the agent.
- **Intentions:** the deliberative state of the agent – what the agent has chosen to do.
- **Plans:** sequences of actions for achieving one or more of its intentions.
- **Events:** triggers for reactive activity by the agent.
- BDI may contribute to the two upper layers of the three-layer architecture.



- Define the behavior in terms of states (of the agent knowledge) and transitions.
- Actions may be triggered on transitions or states.
- Markov models [Baum, 1966] or activity diagrams [Rumbaugh, 1999] may be used in place of state-transition diagrams.

```
agent A {  
    var state = State::NO_TRASH  
  
    on Perception [occurrence.  
        contains(Trash)] {  
        state = State::TRASH_IN_FOV  
    }  
  
    on Perception [|!occurrence.  
        contains(Trash)] {  
        randomMove()  
    }  
  
    ...  
}
```



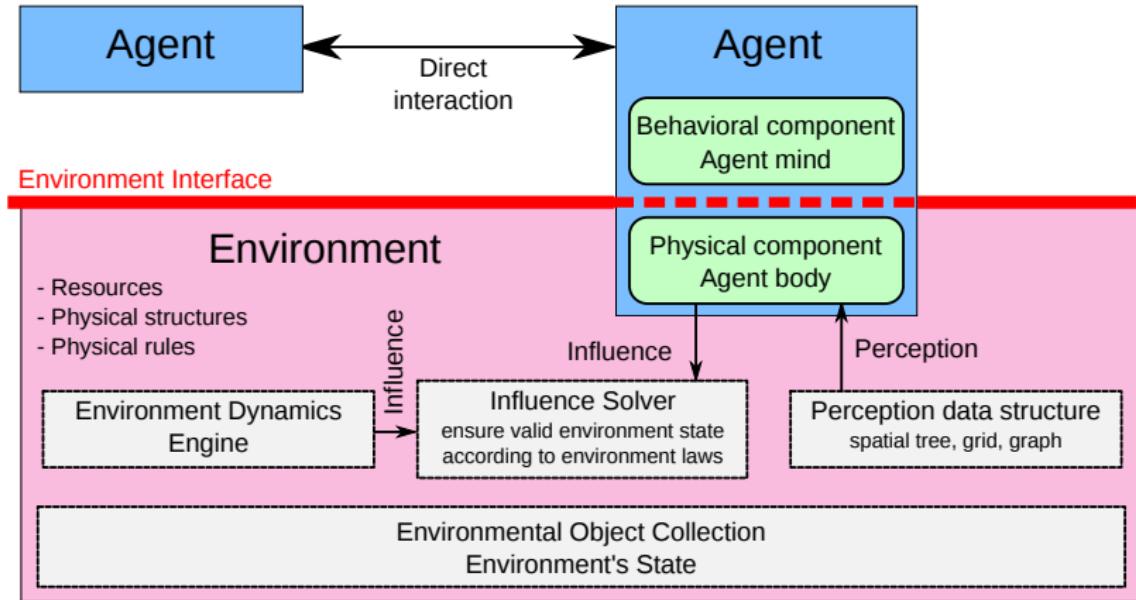
## 1 Introduction to Multiagent Systems

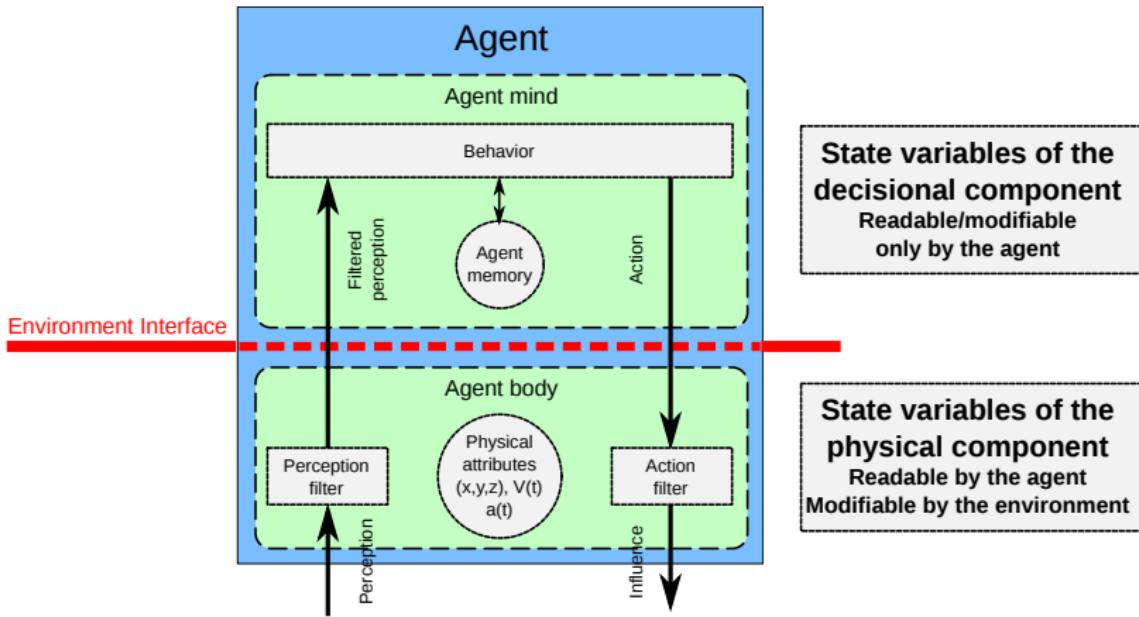
## 2 Multiagent Simulation

## 3 Simulation with a Physic Environment

- General Principles
- Example 1: Traffic Simulation
- Example 2: Pedestrian Simulation

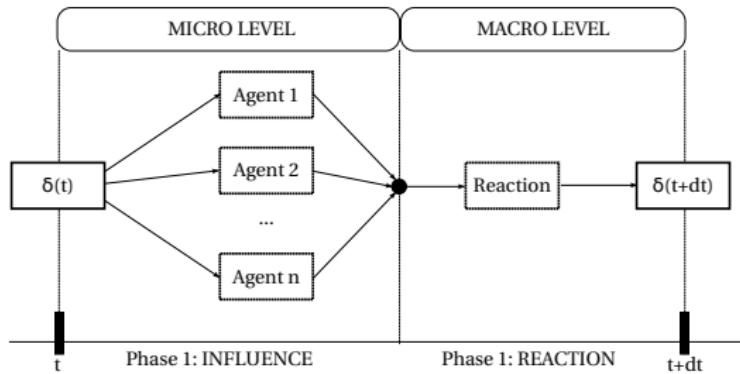
## 4 Cyber-physical System





How to support simultaneous actions from agents?

- 1 An agent does not change the state of the environment directly.
- 2 Agent gives a state-change expectation to the environment: the influence.
- 3 Environment gathers influences, and solves conflicts among them for obtaining its reaction.
- 4 Environment applies reaction for changing its state.



- The agent has the capacity to use its body.
- The body supports the interactions with the environment.

```
event Perception {  
    val object : Object  
    val relativePosition : Vector  
}  
  
capacity EnvironmentInteraction {  
    moveTheBody(motion : Vector)  
    move(object : Object, motion : Vector)  
    executeActionOn(object : Object,  
        actionName : String, parameters :  
        Object*)  
}  
  
space PhysicEnvironment {  
    def move(object : Object,  
        Vector) {  
        //...  
    }  
}
```

```
skill PhysicBody implements  
    EnvironmentInteraction {  
    val env : PhysicEnvironment  
    val body : Object  
    def moveTheBody(motion : Vector) {  
        move(this.body, motion)  
    }  
    def move(object : Object, motion :  
        Vector) {  
        env.move(object, motion)  
    }  
}
```

Event that contains the information extracted from the physic environment and perceived by the agent.

- The agent has the capacity to use its body.
- The body supports the interactions with the environment.

```
event Perception {  
    val object : Object  
    val relativePosition : Vector  
}
```

```
capacity EnvironmentInteraction {  
    moveTheBody(motion : Vector)  
    move(object : Object, motion : Vector)  
    executeActionOn(object : Object,  
        actionName : String, parameters :  
        Object*)  
}
```

```
space PhysicEnvironment {  
    def move(object : Object, motion :  
        Vector) {  
        //...  
    }  
}
```

```
skill PhysicBody implements  
    EnvironmentInteraction {  
    val env : PhysicEnvironment  
    val body : Object  
    def moveTheBody(motion : Vector) {  
        move(this.body, motion)  
    }  
    def move(object : Object, motion :  
        Vector) {  
        env.move(object, motion)  
    }  
}
```

Capacity that enables the agent to change the state of its body and to act into the physic environment.

- The agent has the capacity to use its body.
- The body supports the interactions with the environment.

```
event Perception {  
    val object : Object  
    val relativePosition : Vector  
}  
  
capacity EnvironmentInteraction {  
    moveTheBody(motion : Vector)  
    move(object : Object, motion : Vector)  
    executeActionOn(object : Object,  
        actionName : String, parameters :  
        Object*)  
}
```

```
space PhysicEnvironment {  
    def move(object : Object, motion :  
        Vector) {  
        //...  
    }  
}
```

```
skill PhysicBody implements  
    EnvironmentInteraction {  
    val env : PhysicEnvironment  
    val body : Object  
    def moveTheBody(motion : Vector) {  
        move(this.body, motion)  
    }  
    def move(object : Object, motion :  
        Vector) {  
        env.move(object, motion)  
    }  
}
```

Define the interaction space that is supporting the physic environment, with actions callable from skills.

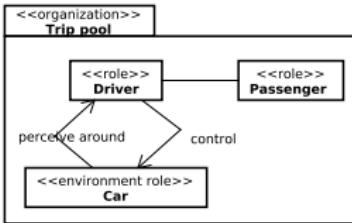
- The agent has the capacity to use its body.
- The body supports the interactions with the environment.

```
event Perception {  
    val object : Object  
    val relativePosition : Vector  
}  
  
capacity EnvironmentInteraction {  
    moveTheBody(motion : Vector)  
    move(object : Object, motion : Vector)  
    executeActionOn(object : Object,  
        actionName : String, parameters :  
        Object*)  
}  
  
space PhysicEnvironment {  
    def move(object : Object, motion :  
        Vector) {  
        //...  
    }  
}
```

Provide an implementation  
of the capacity that is binded  
to the physical environment.

```
skill PhysicBody implements  
    EnvironmentInteraction {  
    val env : PhysicEnvironment  
    val body : Object  
    def moveTheBody(motion : Vector) {  
        move(this.body, motion)  
    }  
    def move(object : Object, motion :  
        Vector) {  
        env.move(object, motion)  
    }  
}
```

- Each vehicle is simulated but road signs are skipped ⇒ mesoscopic simulation.
- The roads are extracted from a Geographical Information Database.
- The simulation model is composed of two parts [Galland, 2009]:
  - 1 the environment: the model of the road network, and the vehicles.
  - 2 the driver model: the behavior of the driver linked to a single vehicle.



## Road Network

- Road polylines:  $S = \{\langle path, objects \rangle \mid path = \langle (x_0, y_0) \dots \rangle\}$
- Graph:  $G = \{S, S \mapsto S, S \mapsto S\} = \{\text{segments}, \text{entering}, \text{exiting}\}$

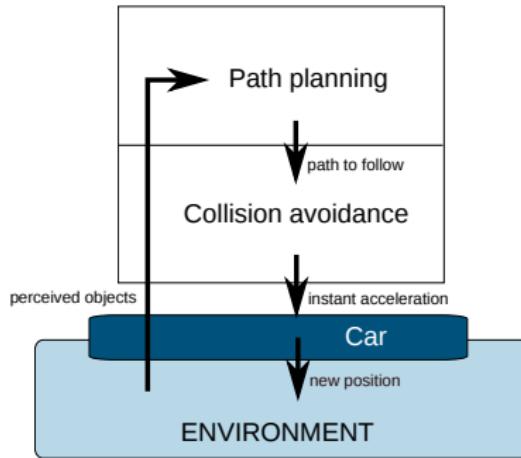
## Operations

- Compute the set of objects perceived by a driver (vehicles, roads...):

$$P = \left\{ o \middle| \begin{array}{l} distance(d, o) \leq \Delta \wedge \\ o \in O \wedge \\ \forall (s_1, s_2), path = s_1.\langle p, O \rangle.s_2 \end{array} \right\}$$

where *path* is the roads followed by a driver *d*.

- Move the vehicles, and avoid physical collisions.



Jasim model [Galland, 2009]

- Based on the A\* algorithm [Dechter, 1985, Delling, 2009]:
  - extension of the Dijkstra's algorithm: search shortest paths between the nodes of a graph.
  - introduce the heuristic function  $h$  to explore first the nodes that permits to converge to the target node.
- Inspired by the D\*-Lite algorithm [Koenig, 2005]:
  - A\* family.
  - supports dynamic changes in the graph topology and the values of the edges.

- **Principle:** compute the acceleration of the vehicle to avoid collisions with the other vehicles.
- Intelligent Driver Model [Treiber, 2000]

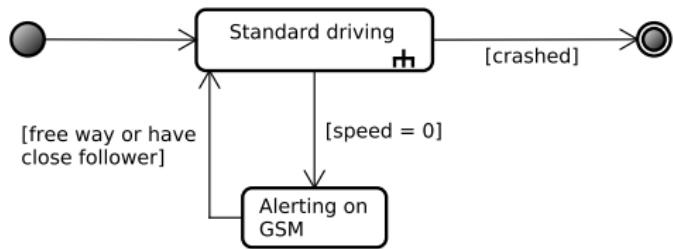
$$\text{followerDriving} = \begin{cases} -\frac{(v\Delta v)^2}{4b\Delta p^2} & \text{if the ahead object is far} \\ -a\frac{(s + vw)^2}{\Delta p^2} & \text{if the ahead object is near} \end{cases}$$

- Free driving:

$$\text{freeDriving} = a \left( 1 - \left( \frac{v}{v_c} \right)^4 \right)$$

## What is simulated?

- 1 Vehicles on a French highway.
- 2 Danger event → “an animal is crossing the highway and causes a crash” .
- 3 Alert events by GSM.
- 4 Arrival of the security and rescue services.





Video done with the SIMULATE® tool — 2012 © Voxelia S.A.S

## What is simulated?

- 1 Movements of pedestrians at a microscopic level.
- 2 Force-based model for avoiding collisions.

[Buisson, 2013]

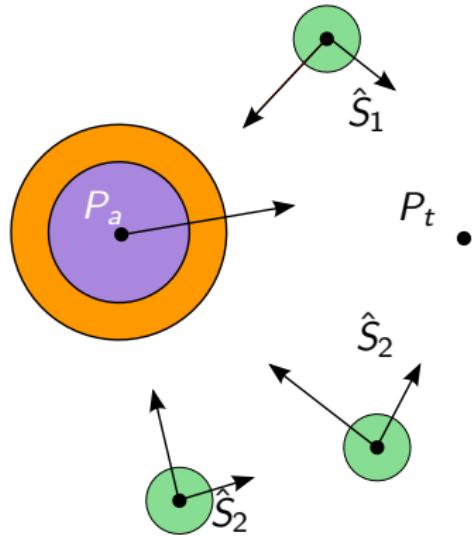


The force to apply to each agent is:

$$\vec{F}_a = \vec{F} + w_a \cdot \delta_{\|\vec{F}\|} \frac{\vec{p}_t - \vec{p}_a}{\|\vec{p}_t - \vec{p}_a\|}$$

$$\vec{F} = \sum_{i \in M} U(t_c^i) \cdot \hat{S}_i$$

- $\vec{F}$ : collision-avoidance force.
- $\hat{S}_i$ : a sliding force.
- $t_c^i$ : time to collision to object  $i$ .
- $U(t)$ : scaling function of the time to collision.
- $M$ : set objects around (including the other agents).
- $w_a$ : weight of the attractive force.
- $\delta_x g$ : is  $g$  if  $x \leq 0$ , 0 otherwise.

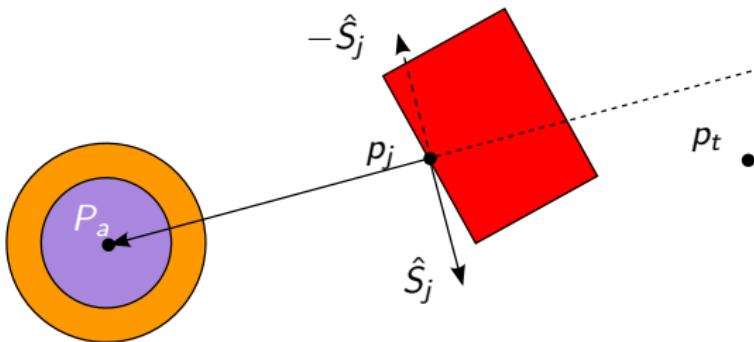


- The sliding force  $\vec{S}_j$  is:

$$\vec{s}_j = (p_j - p_a) \times \hat{y}$$

$$\hat{S}_j = \text{sign}(\vec{s}_j \cdot (\vec{p}_t - p_a)) \frac{\vec{s}_j}{\|\vec{s}_j\|}$$

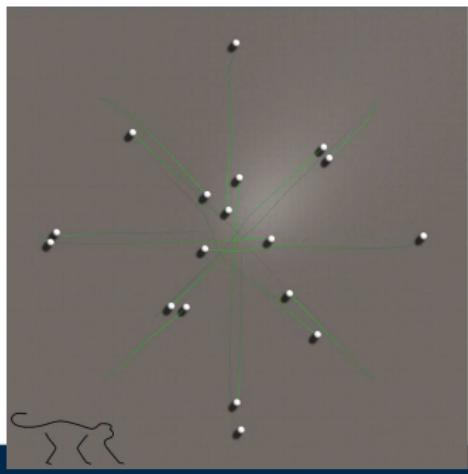
- $\hat{y}$ : vertical unit vector.



- How to scale  $\hat{S}_j$  to obtain the repulsive force?
- Many force-based models use a monotonic decreasing function of the distance to an obstacle.
- But it does not support the velocity of the agent.
- Solution: Use time-based force scaling function.

$$U(t) = \begin{cases} \frac{\sigma}{t^\phi} - \frac{\sigma}{t_{max}^\phi} & \text{if } 0 \leq t \leq t_{max} \\ 0 & \text{if } t > t_{max} \end{cases}$$

- $t$ : estimated time to collision.
- $t_{max}$ : the maximum anticipation time.
- $\sigma$  and  $\phi$  are constants, such that  $U(t_{max}) = 0$ .



Video done with the SIMULATE® tool — 2014 © Voxelia S.A.S

- 1 Introduction to Multiagent Systems
- 2 Multiagent Simulation
- 3 Simulation with a Physic Environment
- 4 Cyber-physical System
  - Definition
  - Intelligent Vehicle

### Definition [Cyb, 2008]

A cyber-physical system is a system where computer components work together to monitor and control physical entities.

- Each computer component is an agent, or a multiagent system.
- Perceptions from the physical sensors.
- Interactions for controlling the physical entity.
- Actions through the physical effectors.

## Intelligent Vehicle

- perceiving its environment: video, laser and GPS sensors,
- driving by itself, or taking control in urgency cases.



## Goals

- 1 Simulate the driver behavior.
- 2 Simulate the environment and the sensors in a virtual lab.
- 3 Deploy the driver software in the real vehicles without change.

## Intelligent Vehicle

- perceiving its environment: video, laser and GPS sensors,
- driving by itself, or taking control in urgency cases.



## Goals

- 1 Simulate the driver behavior.
- 2 Simulate the environment and the sensors in a virtual lab.
- 3 Deploy the driver software in the real vehicles without change.

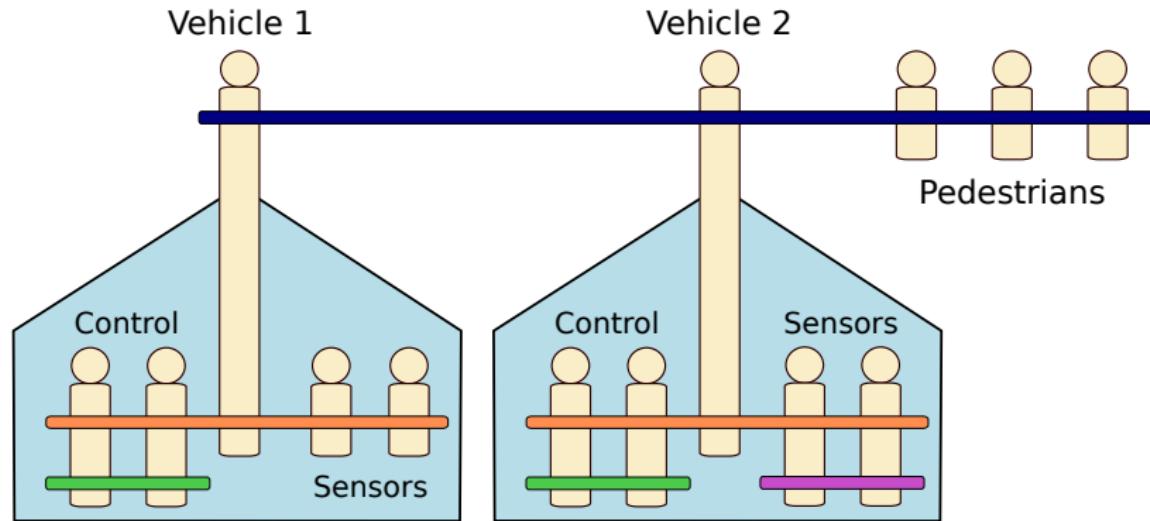
## Intelligent Vehicle

- perceiving its environment: video, laser and GPS sensors,
- driving by itself, or taking control in urgency cases.



## Goals

- 1 Simulate the driver behavior.
- 2 Simulate the environment and the sensors in a virtual lab.
- 3 Deploy the driver software in the real vehicles without change.





[Gechter, 2012]

### 1 Modeling of complex systems:

- ⇒ How to describe the interactions between agents (social, physic, etc.)?
- ⇒ How to model the environment?
- ⇒ How to manage the system  $\Leftrightarrow$  agent relationship?

### 2 Modeling of large-scale systems:

- ⇒ How to model the individuals of a large population at a microscopic level?
- ⇒ How to manage computational cost?

### 3 Initializing the properties of the agents and the environment:

- ⇒ How collecting data from the real world?



**Thank you for your attention...**



# Agent Environment: Definition and Examples

prof.dr.habil. Stéphane GALLAND

During this lecture, I will present:

- 1 the concept of Environment;
- 2 the concept of Agent Environment;
- 3 the Coordination Artifacts and Smart Objects;
  
- 4 a library for a 2D discrete environment;
- 5 a model for a 3D urban environment;
- 6 a multidimensional environment model.

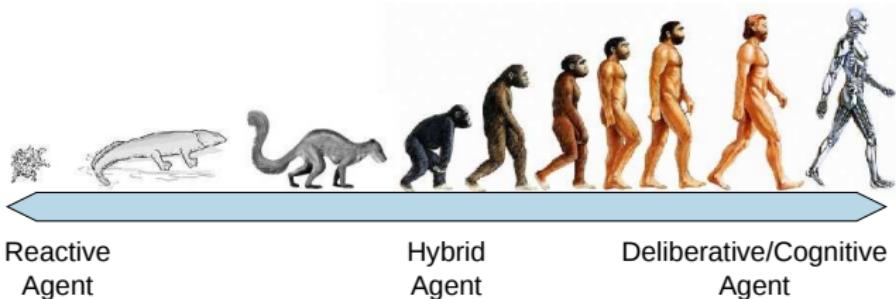
- 1** Remainders of Multiagent Systems
- 2** From Environment to Agent Environment
- 3** Coordination Artifacts
- 4** Smart Objects
- 5** Agent Body
- 6** Jaak Library: simulation with 2D discrete environment
- 7** 3D Urban Environment
- 8** Simulated Multidimensional Environment

- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
- 8 Simulated Multidimensional Environment

An agent is an entity with (at least) the following attributes/characteristics:

- **Autonomy:** Agents encapsulate their internal state, and make decisions about what to do based on this state, without the direct intervention of humans or others.
- **Reactivity:** **Agents are situated in an environment**, and are able to perceive this environment and respond in a timely fashion to changes that occur in it.
- **Pro-activity:** Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.
- **Social Skills:** Agents interact with other agents, and have the ability to engage in social activities in order to achieve their goals.

- **Reactive:** Each agent has a mechanism of **reaction to events**, without having an explanation/understanding of the objectives nor planning mechanisms. **Typical Example:** The ant colony.
- **Cognitive/Deliberative:** Each agent has a **knowledge** base that contains all information and skills necessary for the accomplishment of their **tasks/goals** and the management of his interactions with other agents and his environment: reasoning, planning, normative. **Typical Example:** Multi-Expert Systems.



## Multiagent systems

An MultiAgent Systems (MAS) is a system composed of agents that interact together and through their environment.

### Interactions:

- Direct, agent to agent
- Indirect, **Stigmergy, through the Environment**

Introduced by the French Biologist Pierre-Paul Grassé:

### Stigmergy [Grassé, 1959]

Stimulation of workers by the work they perform.

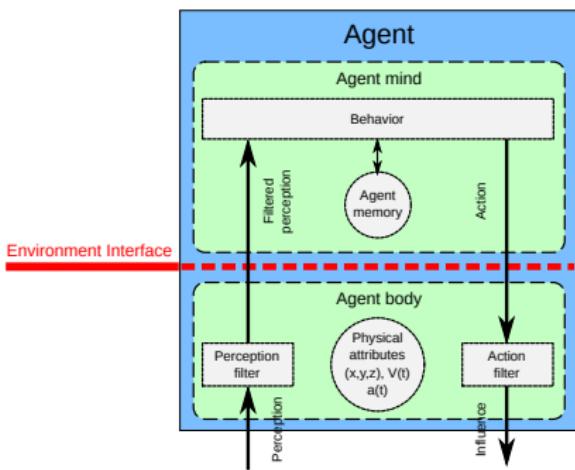
### Stigmergy in multiagent systems [Parunak, 2003]

- Actions by an agent put signs on the environment.
- These signs are perceived by all agents.
- Agents determine their next actions accordingly.

- Direct coordination is too costly in time and resources.
- Enable auto-organization of complex systems without plan, control nor direct interaction.

An agent:

- is located in an environment (situatedness)
- **perceives** the environment through its **sensors**.
- **acts** upon that environment through its **effectors**.
- tends to maximize progress towards its goals by acting in the environment.



## 1 Remainders of Multiagent Systems

## 2 From Environment to Agent Environment

- Definition and Properties of an Environment
- Missions of the Environment
- Dimensions of the Environment

## 3 Coordination Artifacts

## 4 Smart Objects

## 5 Agent Body

## 6 Jaak Library: simulation with 2D discrete environment

## 7 3D Urban Environment

## What is the Environment?

- First-class abstraction of a part of the system that contains all non-agent elements of a multiagent system.
- It provides:
  - a) the surrounding conditions for agents to exist.
  - b) an exploitable design abstraction to build MAS applications.

## Key Ideas

- It is omnipresent.
- Manages access to resources and structures.

## Fully observable (accessible) vs. partially observable (inaccessible)

- Fully observable if agent's sensors detect all aspects of environment relevant to choice of action.
- Could be partially observable due to noisy, inaccurate or missing sensors, or inability to measure everything that is needed.
  - Model can keep track of what was sensed previously, cannot be sensed now, but is probably still true.
- Often, if other agents are involved, their intentions are not observable, but their actions are.
  - Chess: the board is fully observable, as are opponent's moves.
  - Driving: what is around the next bend is not observable.

- **Deterministic:** the next state of the environment is completely predictable from the current state and the action executed by the agent.
- **Stochastic:** the next state has some uncertainty associated with it.
  - Uncertainty could come from randomness, lack of a good environment model, or lack of complete sensor coverage.
- **Strategic environment:** if the environment is deterministic except for the actions of other agents.

- The agent's experience is divided into atomic "**episodes**" (each episode consists of the agent perceiving and then performing a single action).
  - The choice of action in each episode depends only on the episode itself.
  - Examples of episodic are expert advice systems — an episode is a single question and answer.
- **Sequential** if current decisions affect future decisions, or rely on previous ones.
  - Most environments (and agents) are sequential.
  - Many are both — a number of episodes containing a number of sequential steps to a conclusion.

- **Discrete:** time moves in fixed steps, usually with one measurement per step (and perhaps one action, but could be no action).

### Example

a game of chess

- **Continuous:** Signals constantly coming into sensors, actions continually changing.

### Example

driving a car, a pedestrian moving around

- **Dynamic:** if the environment may change over time.
- **Static:** if nothing (other than the agent) in the environment changes.
  - Other agents in an environment make it dynamic.
  - The goal might also change over time.

## Examples

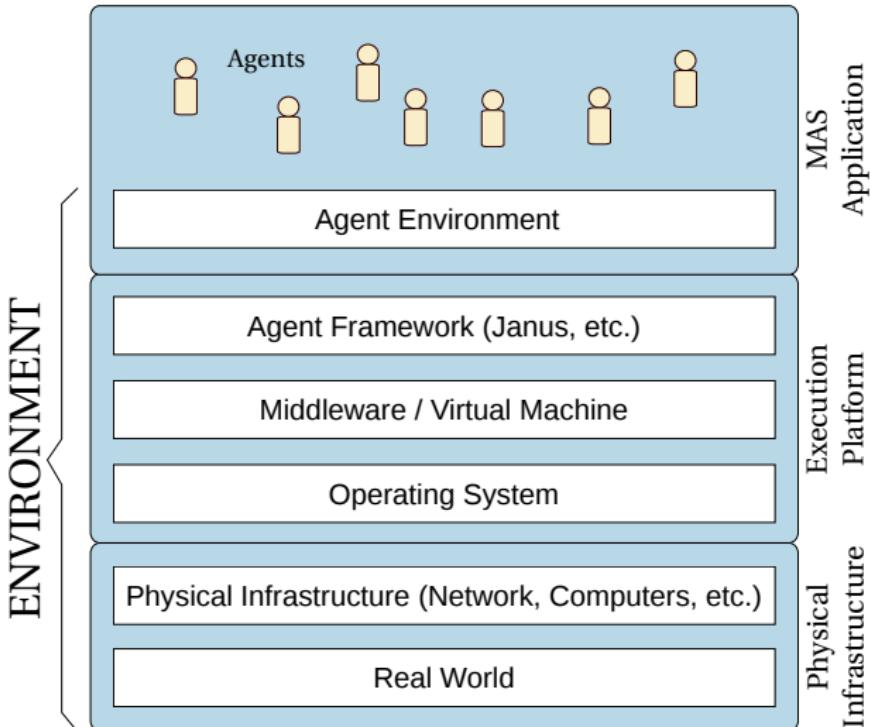
- Playing football, other players make it dynamic.
- Mowing a lawn is static (unless there is a cat).
- Expert systems usually static (unless knowledge changes).

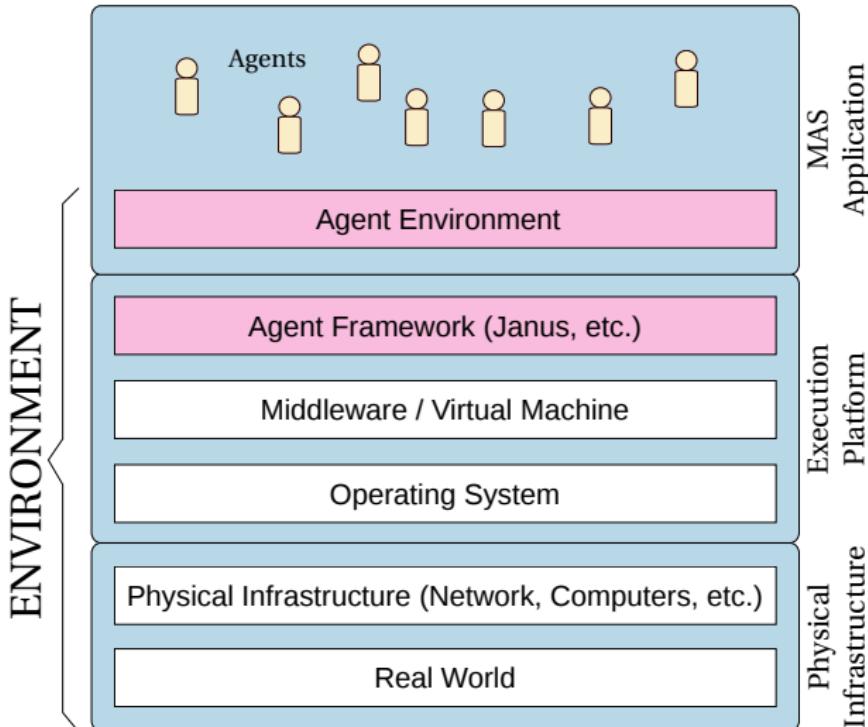
- M1 - Sharing informations: Agent environment is a shared structure for agents, where each of them perceives and acts.
- M2 - Managing agents actions and interactions: It is related to the management of agents' simultaneous and joint actions and to the preservation of the environmental integrity: influence-reaction model  
[Ferber, 1996, Michel, 2004, Galland, 2009].
- M3 - Managing perception and observation: Agents can manage the access to environmental informations and guarantee the partialness and localness of perceptions.
- M4 - Maintaining endogenous dynamics: The environment is an active entity; it can have its own processes, independently of the ones of the agents.

- **M1 - Sharing informations:** Agent environment is a shared structure for agents, where each of them perceives and acts.
- **M2 - Managing agents actions and interactions:** It is related to the management of agents' simultaneous and joint actions and to the preservation of the environmental integrity:  
influence-reaction model  
[Ferber, 1996, Michel, 2004, Galland, 2009].
- **M3 - Managing perception and observation:** Agents can manage the access to environmental informations and guarantee the partialness and localness of perceptions.
- **M4 - Maintaining endogenous dynamics:** The environment is an active entity; it can have its own processes, independently of the ones of the agents.

- **M1 - Sharing informations:** Agent environment is a shared structure for agents, where each of them perceives and acts.
- **M2 - Managing agents actions and interactions:** It is related to the management of agents' simultaneous and joint actions and to the preservation of the environmental integrity:  
influence-reaction model  
[Ferber, 1996, Michel, 2004, Galland, 2009].
- **M3 - Managing perception and observation:** Agents can manage the access to environmental informations and guarantee the partialness and localness of perceptions.
- **M4 - Maintaining endogenous dynamics:** The environment is an active entity; it can have its own processes, independently of the ones of the agents.

- **M1 - Sharing informations:** Agent environment is a shared structure for agents, where each of them perceives and acts.
- **M2 - Managing agents actions and interactions:** It is related to the management of agents' simultaneous and joint actions and to the preservation of the environmental integrity:  
influence-reaction model  
[Ferber, 1996, Michel, 2004, Galland, 2009].
- **M3 - Managing perception and observation:** Agents can manage the access to environmental informations and guarantee the partialness and localness of perceptions.
- **M4 - Maintaining endogenous dynamics:** The environment is an active entity; it can have its own processes, independently of the ones of the agents.





## Requirements

- Provides the features for running agents.
- Must support parallel execution of the agents.

## The Janus Agent Framework

- Fully distributed.
- Dynamic discovery of Kernels.
- Automatic synchronization of kernels' data (easy recovery).
- Micro-Kernel implementation.
- Official website: <http://www.janusproject.io>



Other frameworks may be created. See Seminar #4.

## Definition

The agent environment (or application environment) embeds:

- the environmental logic of the application, and
- the interface to the agents.

## Content of the Agent Environment

- Resources in the agent environment are passive and reactive.  
They are not agents.
- May support a specific dimension of the environment  
[Odell, 2003]:
  - a) Physical, and
  - b) Communicational / Social.

## Physic Environment

Class of real or simulated systems in which agents and objects have an explicit position, and that produce localized actions.

## Properties

- Contains all objects.
- Agents interact with it via dedicated model.
- Agents' bodies are “managed” by the environment.
- Multiple “Views” of the environment can be implemented (1D, 2D, 3D).
- Enforces Universal Laws (e.g. Laws of physics).
- Should be a singleton.

- Multiple ways of agent interaction: event, messages, etc.
- Multiple models of agent relationship: authority, auction, contract-net-protocol, etc.
- Social dimension could influence other dimensions [Galland, 2015].

### Communicational Dimension in SARNL

- Supported by Space in SARNL.
- Default Interaction Space: based on events (may be redefined).
- Programmer can create new Space specifications (and ways of interacting):
  - FIPA,
  - Organizational (MOISE, CRIOS, etc).

- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
- 8 Simulated Multidimensional Environment

[Norman, 1991]

"Artifacts play a critical role in almost all human activity [...]. Indeed [...] the development of artifacts, their use, and then the propagation of knowledge and skills of the artifacts to subsequent generations of humans are among the distinctive characteristics of human beings as a species."

[Amant, 2005]

"The use of tools is a hallmark in intelligent behaviour. It would be hard to describe modern human life without mentioning tools of one sort or another."

- Artifacts and tools are essential elements for mediating and supporting human activities.  
Especially for social activities.
- First characterization of the artifact notion:
  - **Tool Perspective:** artifacts as a kind of device explicitly designed to embed some kind of functionality, that can be properly used by humans to do their work tool perspective.
  - **Activity Target Perspective:** artifacts as objectification of human goals and tasks.
- Basic bricks of human environment.

### Psychology & human sciences

- Activity Theory.
- Artifacts as mediators of any (complex) activity.

### Cognitive science

- Distributed Cognition.
- Cognition spread among humans and the environment where they are situated.

### Computer-Supported Cooperative Work

- Coordinative Artifacts.
- Artifacts as bricks of human fields of works.

### Distributed Artificial Intelligence

- The role of environment for supporting intelligence.
- "Lifeworld Analysis" [Agre, 1997]

If it is such a fundamental concept for human society and human working environments, could it be useful also in agent societies and MAS for conceiving agent “working environments?” [Ricci, 2005]

- Analogy between human and agent societies:

- agent societies / MAS conceived as systems of autonomous cognitive entities working together within some kind of social activities, situated within some kind of working environment, etc.

- A&A conceptual framework:

- defining a notion of artifact and working environment within MAS.
  - foundational & engineering aims: finally useful for building MAS and agent -based systems.

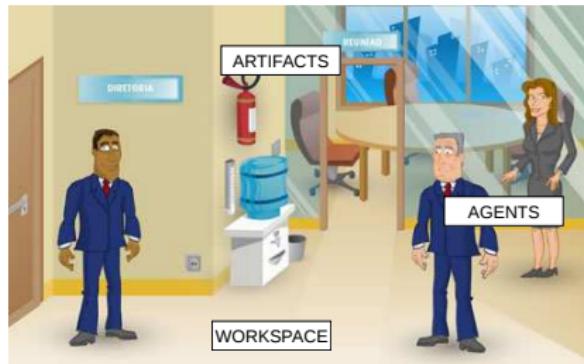
## 1 Abstraction and Generality:

- defining a basic set of abstractions & related theory general enough for designing & engineering general-purpose working environments.  
engineering principles: encapsulation, reuse, extendibility, etc.
- effective/expressive enough to capture the essential properties of specific domains.

## 2 Cognition:

- the properties of such environment abstractions should be conceived to be suitably and effectively exploited by cognitive agents.  
agents as intelligent constructors, users, manipulators of such environment
- analogy with human society.

- MAS: set of agents working together in the same working environment.
  - Working environment: set of artifacts, collected in workspaces.
- 
- Agents dynamically create, share and use artifacts to support their social/individual activities (mediation role of artifacts).



## Building blocks of working environments

- Different kinds / types of artifacts, with possibly multiple instances for each type.  
similar to objects but in an agent world
- Representing any kind of resource or tool that agents can dynamically create, share, use, manipulate.

## Passive, dynamic, function-oriented entities

- Designed to encapsulate some kind of function:  
the intended purpose of the artifact vs. agent as goal/task-oriented entities
- Typically stateful object: result of an action depends on the state history.

## Agents-Artifacts Interaction

- Based on a notion of use.
- Agents use artifacts so as to exploit their functionalities vs. communication-based interaction.  
agents do not communicate with artifacts

## Artifact Usage Interface

- Set of operations that agent can trigger on the artifact.
- It can change depending on the working state of the artifact.

## Artifact Observable State and Events

Artifacts as sources of observable events that can be perceived by agents using or observing them.

## Manual of an Artifact

- Equipping each artifact with the formal description of its functionality and usage modalities:
  - a) functional description: why to use it.
  - b) operating instructions: how to use it.
- Enabling a cognitive use of artifacts with dynamic discovery, selection and learning of artifacts.

- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
- 8 Simulated Multidimensional Environment

## Smart Objects

"When almost every object either contains a computer or can have a tab attached to it, obtaining information will be trivial."  
[Weiser, 1991]

- Objects are smart in the sense that they act as digital information sources.

## Tangible Objects

"Tangibles bits or tangible user interfaces that enable users to grasp & manipulate bits in the center of users' attention by coupling the bits with everyday physical objects and architectural surfaces." [Ishii, 1997]

- Objects could react to user interactions.

[Kallmann, 1998]

An object that can describe its own possible interactions.

- Focus on Smart Object interaction with Humans and Agents in virtual worlds.
- These interaction information enables more general interaction schemes, and can greatly simplify the action planning of an agent.

- 1 **Object properties:** physical properties and a text description.
- 2 **Interaction information:** position of handles, buttons, grips, and the like.
- 3 **Object behavior:** different behaviors based on state variables.
- 4 **Agent behaviors:** description of the behavior an agent should follow when using the object]

### Example

Include 3D animation information in the object information, but this is not considered to be an efficient approach [Jorissen, 2005].

[Poslad, 2009]

A smart physical object:

- is active,
- is digital,
- is networked,
- can operate to some extent autonomously,
- is reconfigurable, and
- has local control of the resources it needs such as energy, data storage, etc.

⇒ may be seen as a particular type of agents.

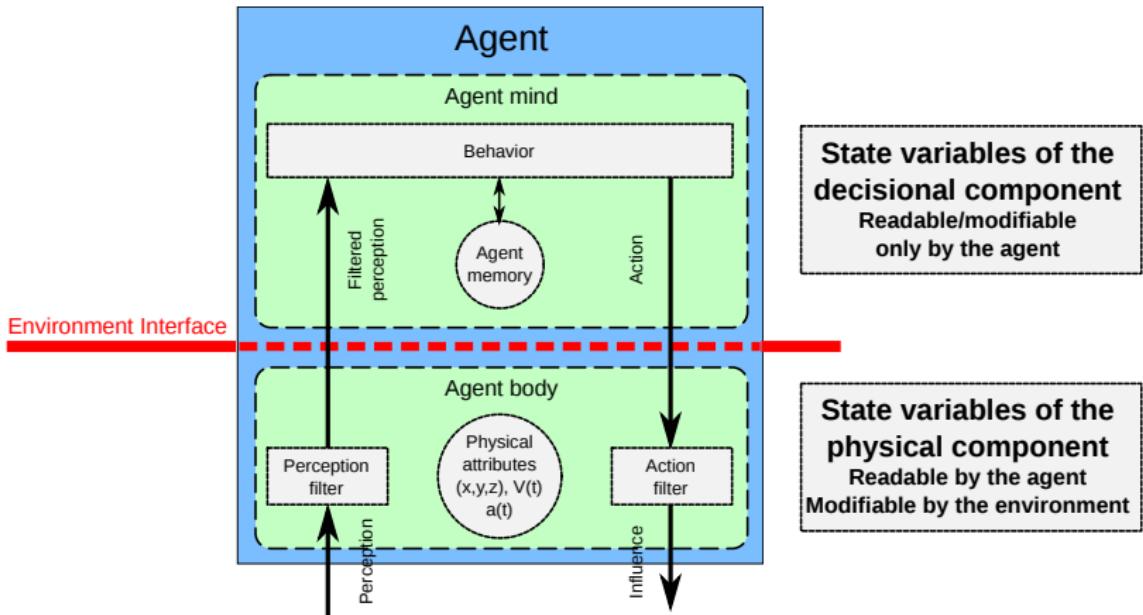
- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
- 8 Simulated Multidimensional Environment

[Saunier, 2015]

The agent body is a component of the multiagent system working as an interface between mind (agent) and environment. It is embedded in the environment to enforce body rules and ecological laws, but is influenced by -and allows introspection for- the mind. An agent may have one or more bodies in the environment(s) it participates in.

### Body Responsibilities

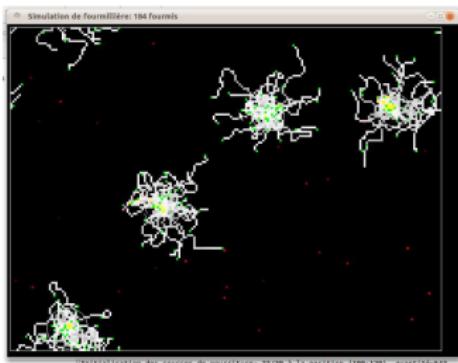
- **Representation of the agent:** It is the observable part of the agent in an environment dimension (physic or social).
- **Perception mediation:** It defines the perception capabilities of the agent.
- **Action mediation:** It defines the action capabilities of the agent.
- **Environment Component:** It respects the laws of the agent environment.



- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
  - Overview
  - Ant Colony Simulation
- 7 3D Urban Environment

## What is Jaak?

- Reactive agent execution framework (but may be cognitive with additional libraries).
- Provides a **discrete 2D environment model**.
- Provides a simplified agent-environment model based on LOGO-like primitives: move forward, turn left, etc.



<https://github.com/gallandarakhneorg/jaak>

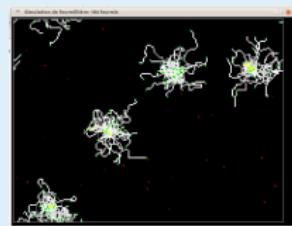
## LOGO

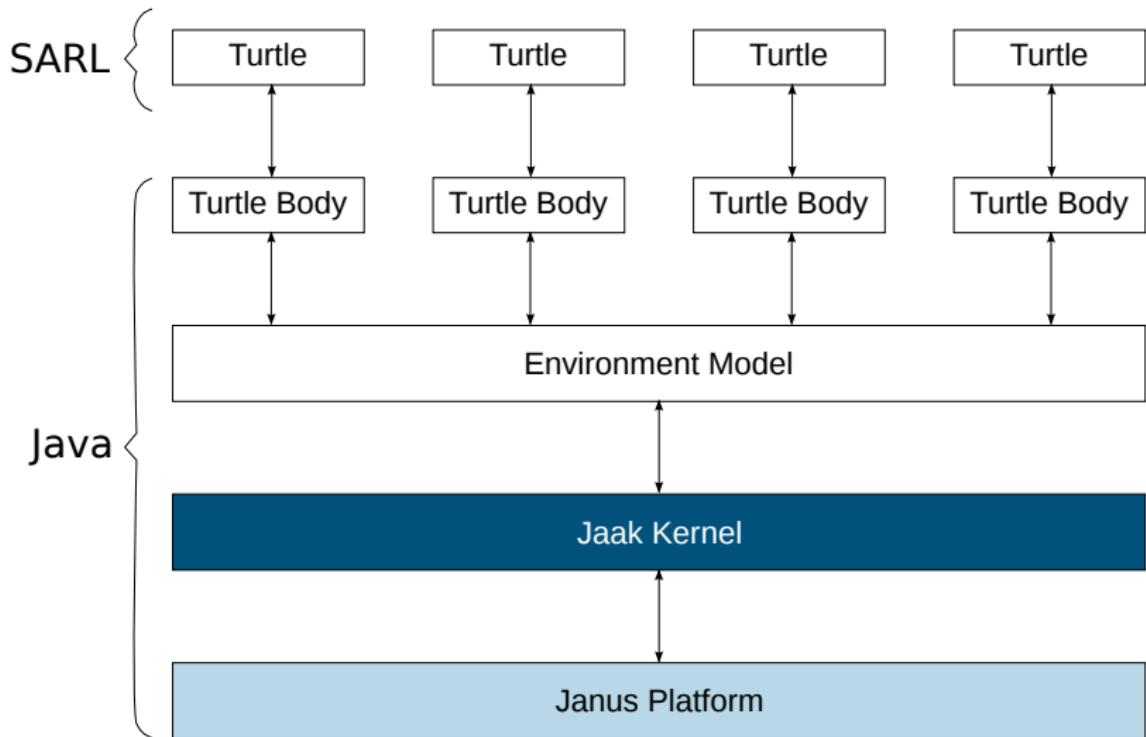
- Reflexive and functional programming language from the MIT.
- Mainly known for its famous **graphical turtle**.
- Turtles move with simple instructions: move forward, turn 45 degrees left, etc.

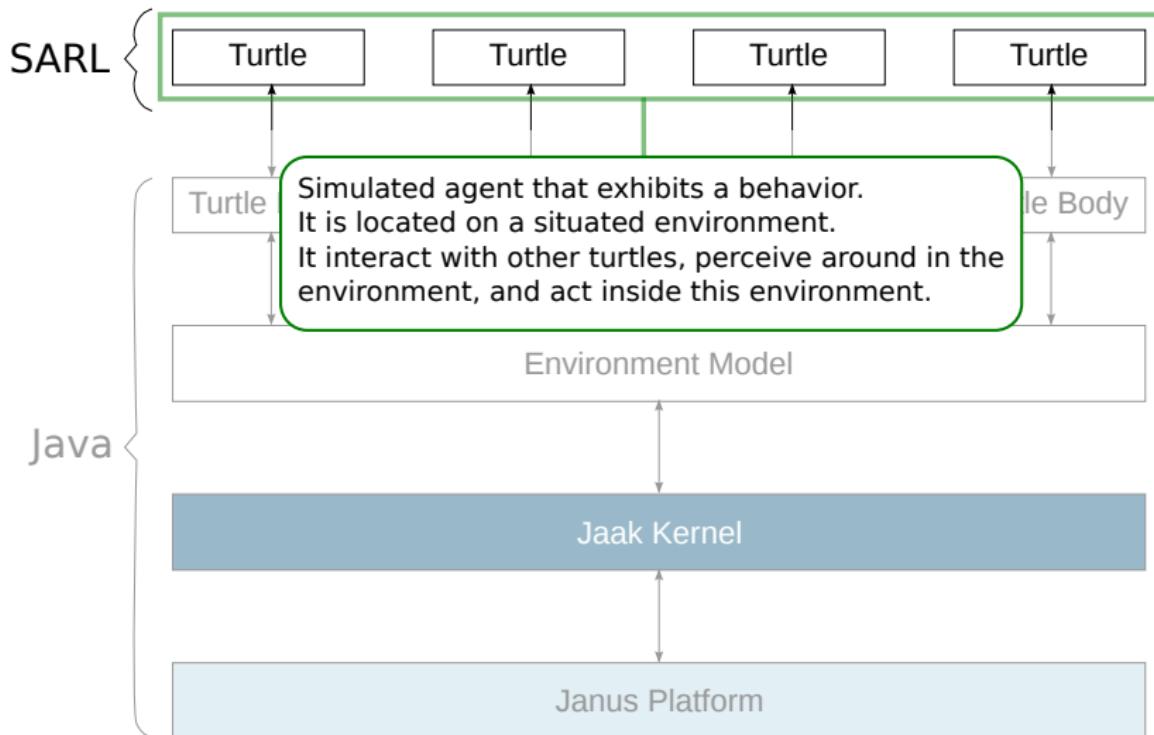


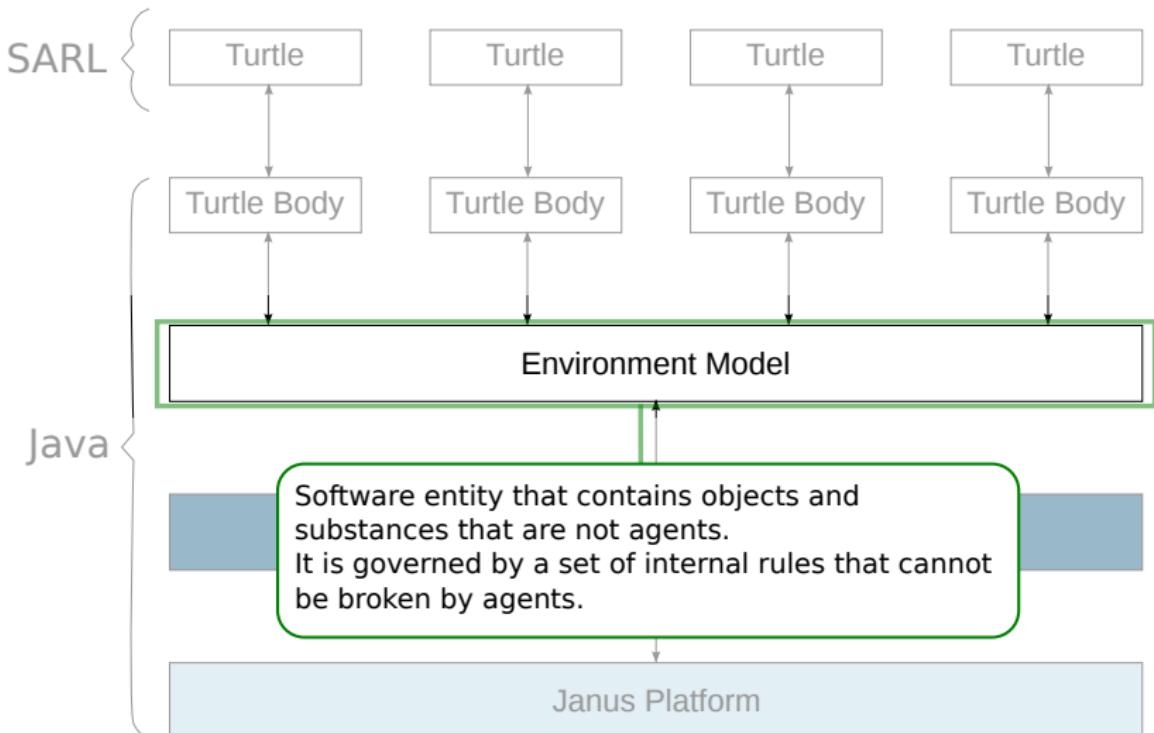
## Jaak

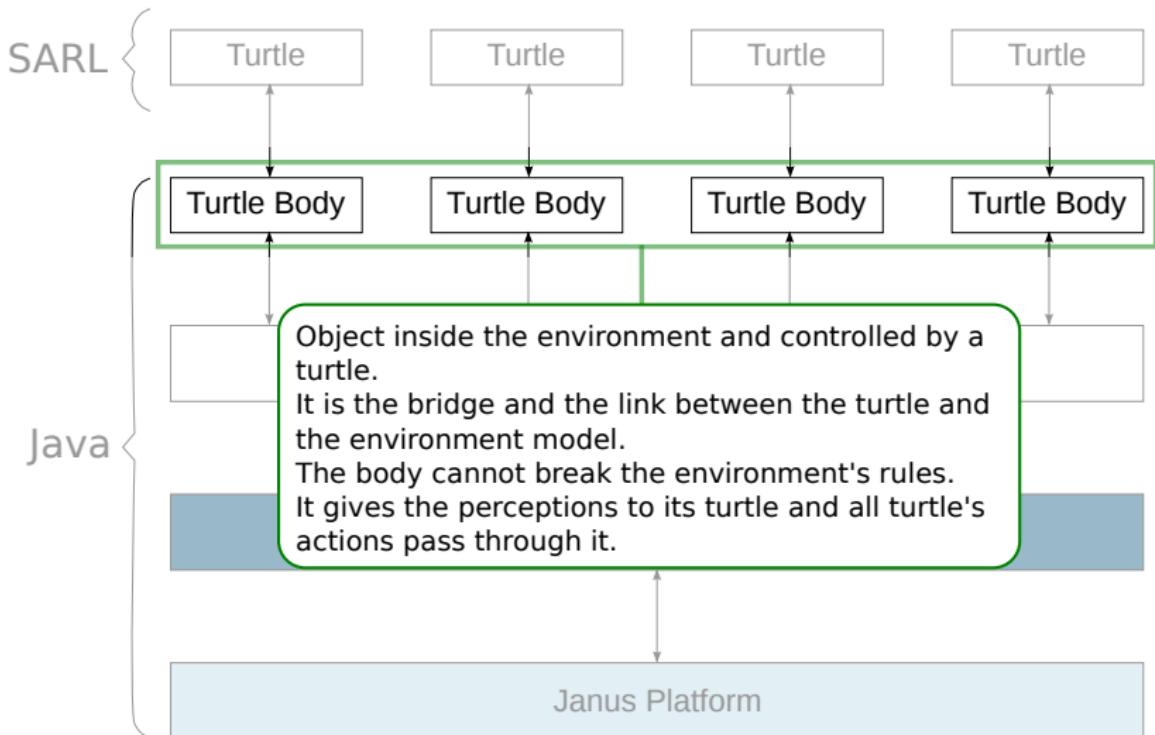
- Turtles are situated agents, which is able to move on a 2D environment.
- Turtles are written in SARC.
- A specific capacity provides the simple instructions for moves.











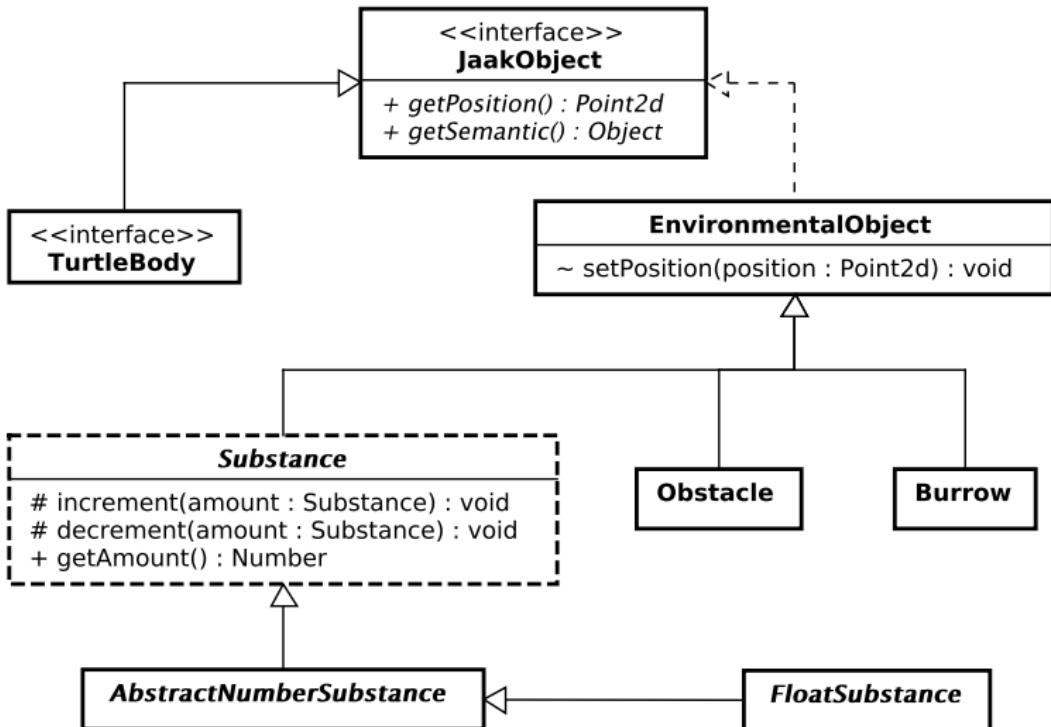
## Environment Objects

- Environment objects are all the entities inside the environment.
- Turtles are not, and cannot be environmental objects.
- But turtle's bodies are special types of environmental objects.
- Each body is associated to one turtle agent, and contains the physical description of that agent: field of vision, maximal speed, weight, etc.

## Substances

Substance is a special type of environmental object:

- It is a particular type of liquid, solid, or gas located in the environment.
- It is countable and may be divided or expanded.



The environment structure is based on a 2D grid (matrix of cells).

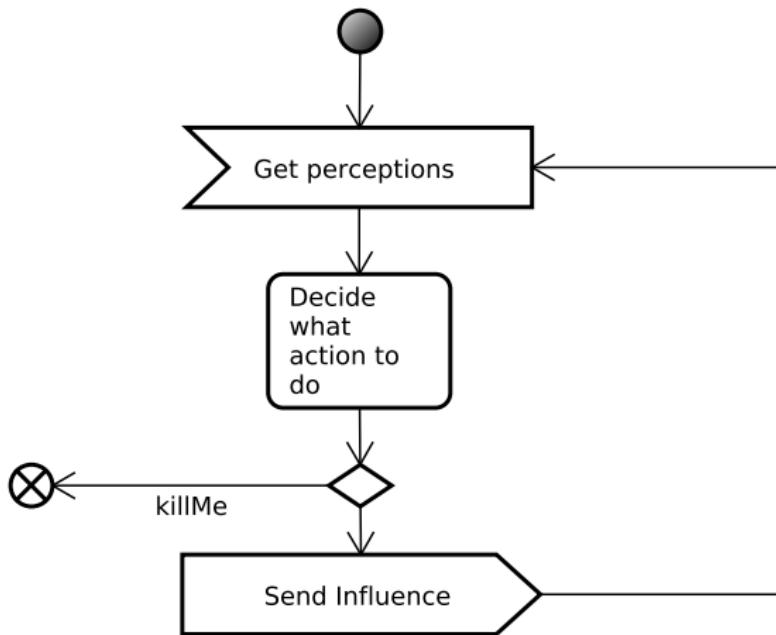
### Properties of the cells

- A cell contains:
  - At most one agent body (see burrow for exception), or
  - at most one obstacle; and
  - many other environmental objects.
- Cell's "graphical" size is application-dependent.

### Properties of the grid

- Size is fixed at start-up.
- Grid could be wrapped: if an agent is located on one side of the grid and is trying to move outside; it will be moved at the opposite side of the grid.



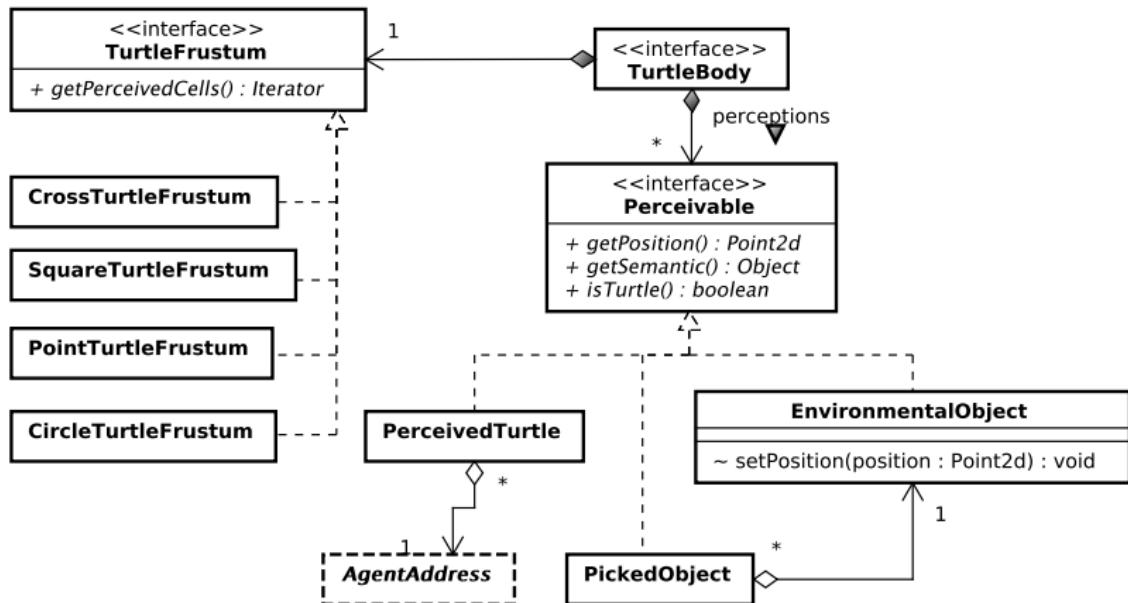


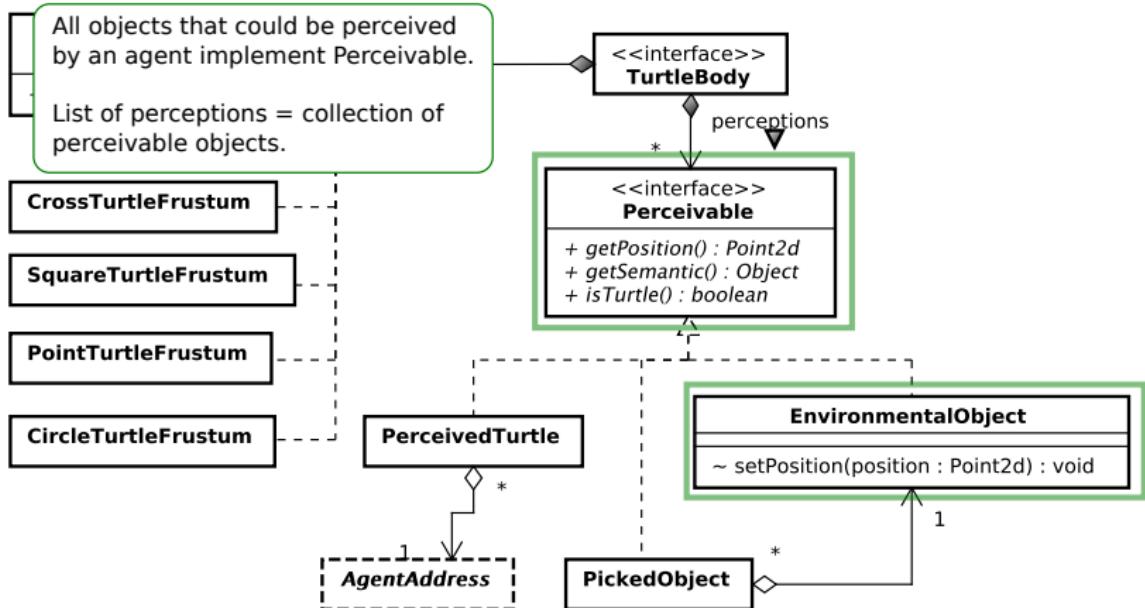
## Definition

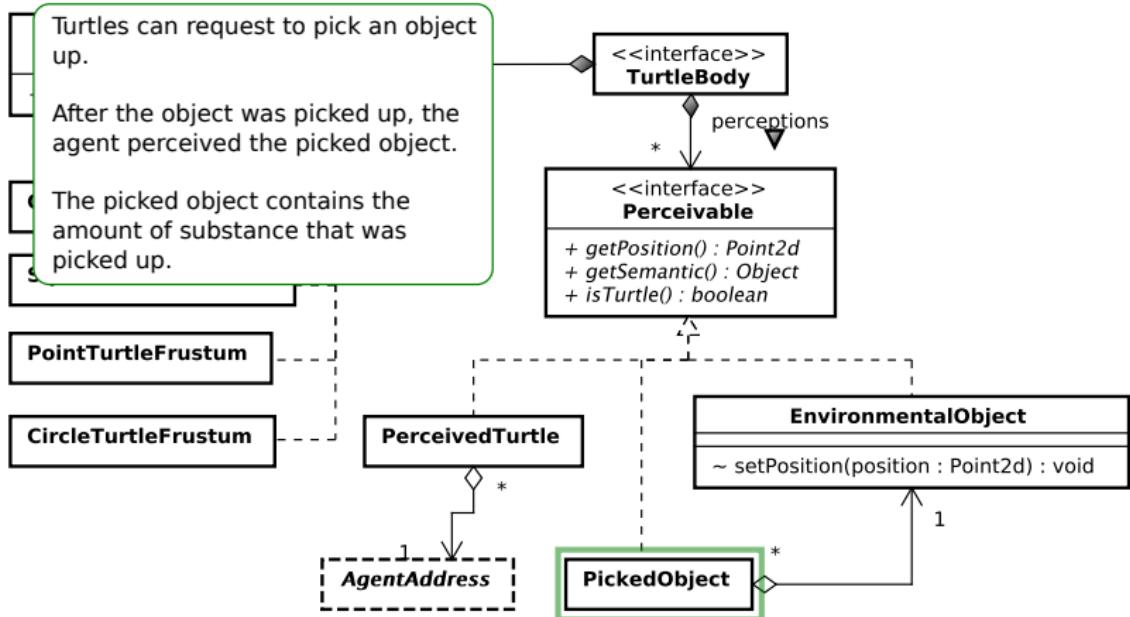
An environmental object, or a collection of environmental objects, that are inside, or intersecting, the field of perception of an agent. Perception is computed and given by the sensors of the agent's body.

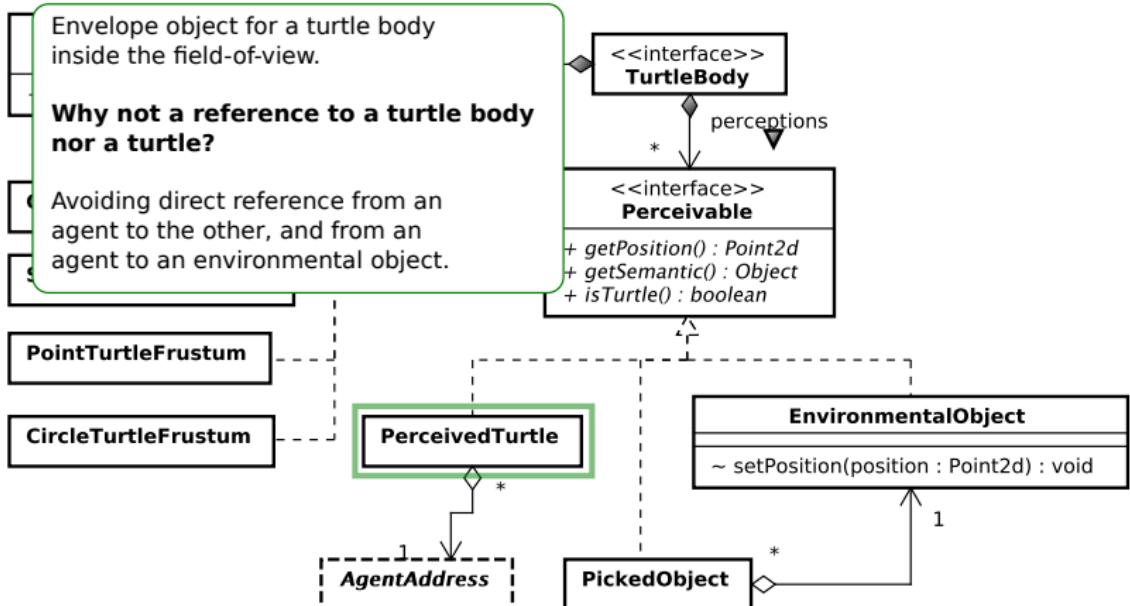
## Hypotheses

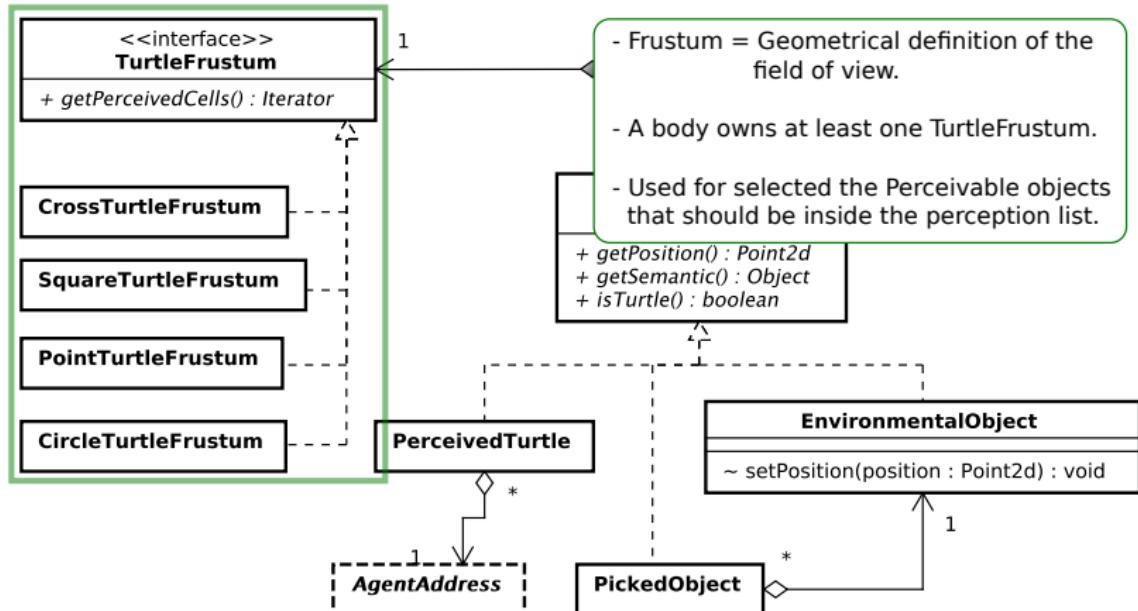
- 1 Agent cannot be omniscient: the scope of its perception is restricted to a limited portion of the environment.
- 2 Perception is for a given time.
- 3 Perception's list may be influenced or changed according to the properties of the sensors, e.g. visual impaired agent.
- 4 An agent can perceive even if it is inside a burrow.

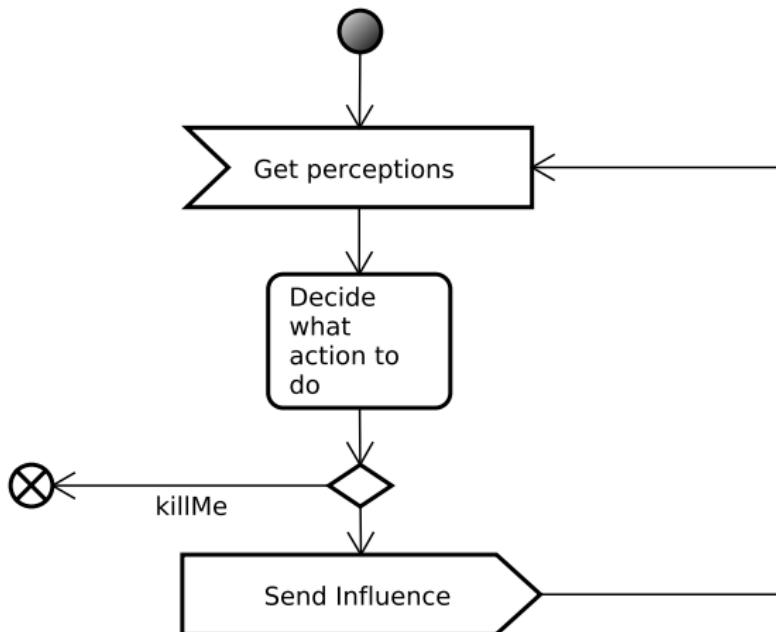












### Problem 1: simultaneous actions

- Actions decided by different agents that may be applied at the same time on the same action space.
- Simultaneous actions may be under conflict. Example: when two agents try to move on the same cell.

### Problem 2: uncertain actions

- An action decided by an agent may be skipped or partially applied according to the internal rules of the environment.
- Example: an agent cannot go inside the same cell of an obstacle.

### Solution: influence

- Actions from agents are not directly applied in the environment.
- Conflicts among actions are detected and solved.
- The resolution result is applied in the environment.
- **Influence:** the expected action by the agent.

### Problem 1: simultaneous actions

- Actions decided by different agents that may be applied at the same time on the same action space.
- Simultaneous actions may be under conflict. Example: when two agents try to move on the same cell.

### Problem 2: uncertain actions

- An action decided by an agent may be skipped or partially applied according to the internal rules of the environment.
- Example: an agent cannot go inside the same cell of an obstacle.

### Solution: influence

- Actions from agents are not directly applied in the environment.
- Conflicts among actions are detected and solved.
- The resolution result is applied in the environment.
- Influence: the expected action by the agent.

### Problem 1: simultaneous actions

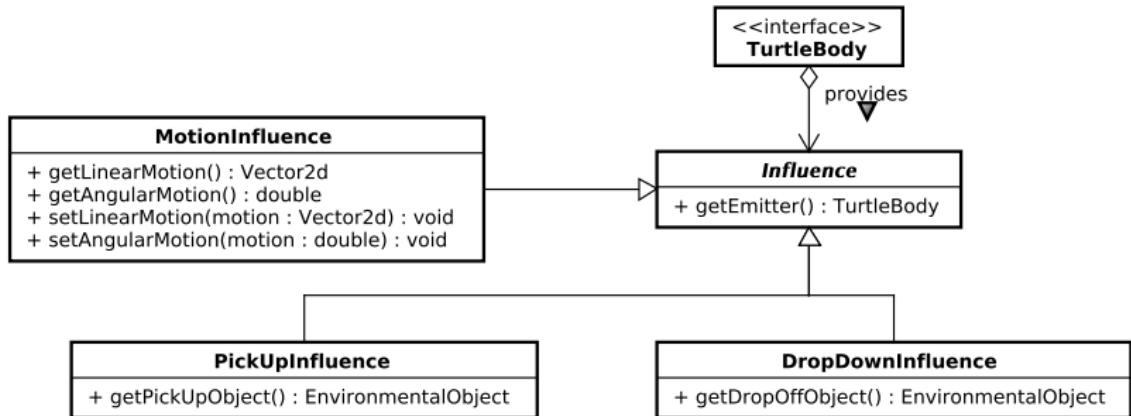
- Actions decided by different agents that may be applied at the same time on the same action space.
- Simultaneous actions may be under conflict. Example: when two agents try to move on the same cell.

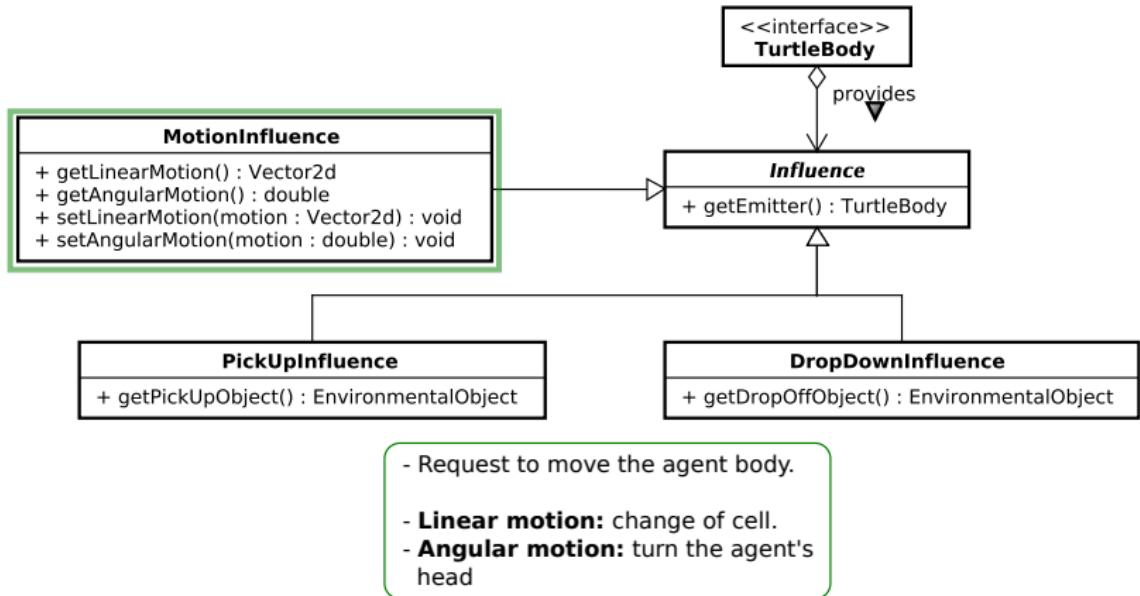
### Problem 2: uncertain actions

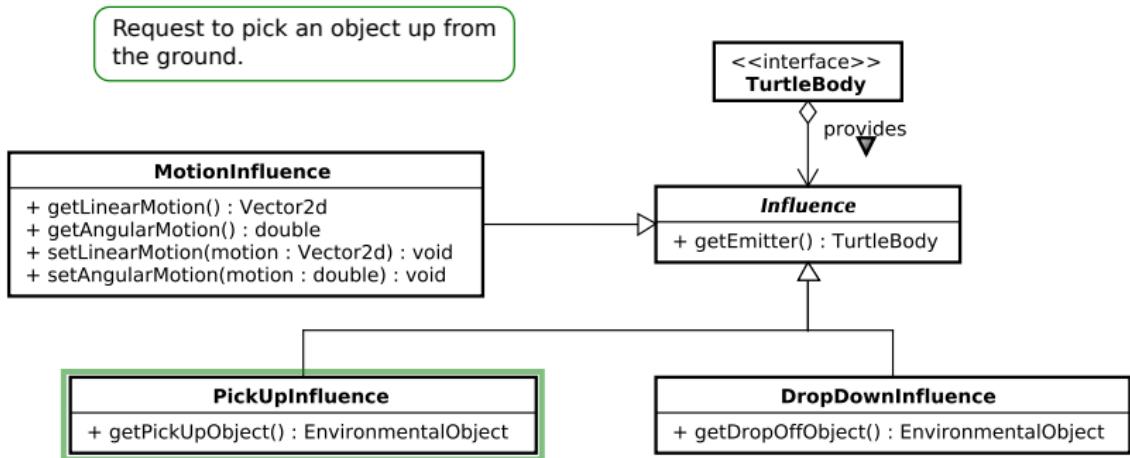
- An action decided by an agent may be skipped or partially applied according to the internal rules of the environment.
- Example: an agent cannot go inside the same cell of an obstacle.

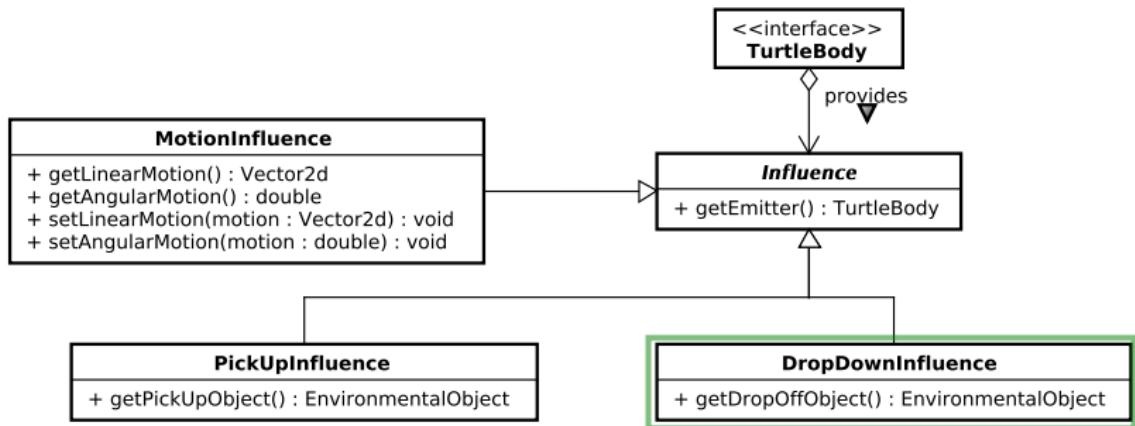
### Solution: influence

- Actions from agents are not directly applied in the environment.
- Conflicts among actions are detected and solved.
- The resolution result is applied in the environment.
- **Influence: the expected action by the agent.**









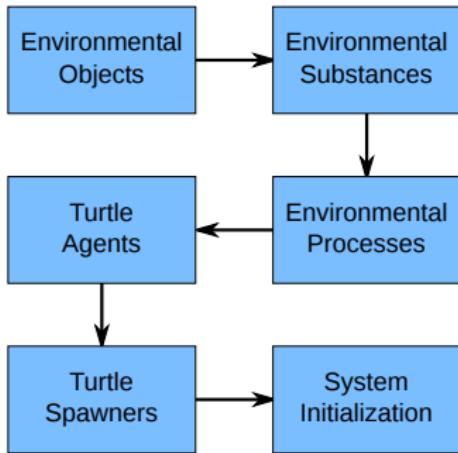
Request to drop an object down  
on the ground.

## Problem

Simulating the global behavior of a colony of ants for finding the shortest routes from the nest to a food source.

## What is a Ant?

- Simple insects with no global knowledge and limited memory.
- Capable of performing simple actions:
  - move
  - get food from the cell
  - sense pheromone in neighbor cells
  - put pheromone into the current cell



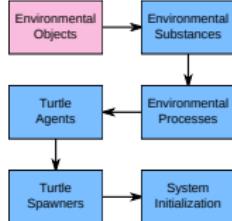
## General Principle

For each type of object inside the environment, a subclass of `EnvironmentalObject` must be defined.

## In the ant colony problem

- Nest: location where many ants may be at the same place. It is a kind of Burrow to enable this behavior.

```
class Nest extends Burrow {  
  
    val colonyId : int  
  
    new (colonyId : int) {  
        this.colonyId = colonyId  
    }  
  
    def getColonyId : int {  
        this.colonyId  
    }  
}
```

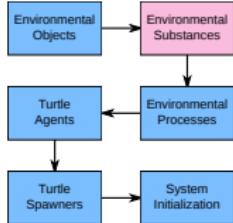


## General Principle

For each substance (countable/measurable) inside the environment, a subclass of Substance must be defined.

## In the ant colony problem

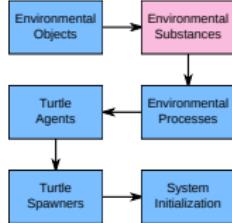
- ColonyPheromone: put when the ant is going far and far from the ant colony.
- FoodPheromone: put when the ant is going far and far from a food source.
- Food: a source of food.



## Definition of a food source

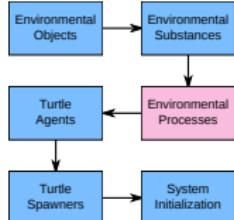
- Contains the quantity of available food.
- Quantity cannot increase.

```
class Food extends FloatSubstance {  
    new (foodQuantity : float) {  
        super(foodQuantity, typeof(Food));  
    }  
  
    def decrement(s : Substance) : Substance {  
        var oldValue = floatValue()  
  
        decrement(s.floatValue())  
  
        var c = clone  
        c.value = abs(floatValue() - oldValue)  
        return c  
    }  
  
    def increment(s : Substance) : Substance {  
        // The food source could not be increased  
        null  
    }  
}
```



## General Principle

- Endogenous environmental activities are the processes that produce an evolution of the environment state outside the control of any agent.
- Two types of endogenous environmental activities:
  - a) a autonomous endogenous process associated to an object
  - b) the global endogenous engine



## In the ant colony problem

Pheromones are substances that are evaporating over the time.

## Update of the definition of a pheromone

Implements the AutonomousEndogenousProcess interface.

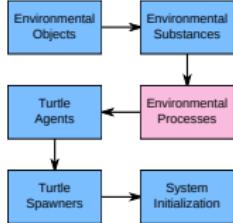
```
class Pheromone extends Substance
    implements AutonomousEndogenousProcess {

    def runAutonomousEndogenousProcess(
        currentTime : float,
        simulationStepDuration : float)
        : Influence {

        decrement(simulationStepDuration * this.evaporationAmount)

        if (floatValue() <= 0) {
            /* Provided by EnvironmentalObject */
            return createRemovalInfluenceForItself()
        }

        return null
    }
}
```

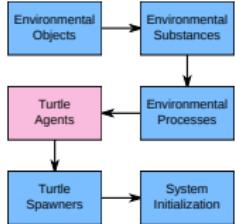


## General Principle

Define the different behaviors associated to the turtle agents.

### In the ant colony problem

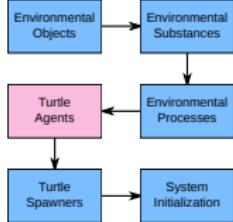
- Foragers: Search for food, and carrying it to the nest.
- Patrollers: patrolling around the nest, and defend against others.



## Definition of a patroller

- Define the initialization (similar for all turtles agents).
- Define the specific behavior.

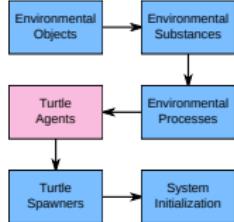
```
agent PatrollingAnt {  
  
    uses Lifecycle  
  
    on Initialize {  
        var body = new PhysicBodySkill  
        setSkill(typeof(PhysicBody), body)  
        setSkill(typeof(PheromoneFollowingCapacity), new  
            PheromoneFollowingSkill)  
        setSkill(typeof(FoodSelectionCapacity), new  
            FoodSelectionSkill)  
    }  
  
    on BodyCreated {}  
  
    on SimulationStopped {  
        killMe  
    }  
}
```



## Definition of a patroller

- Define the initialization (similar for all turtles agents).
- **Define the specific behavior.**

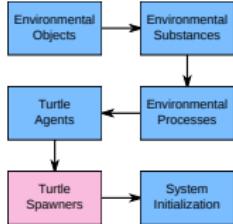
```
agent PratrollingAnt {  
    ...  
    uses PhysicBody  
  
    var state = ForagerState::SEARCH_FOOD  
    var bag : Food  
  
    on Perception [state == ForagerState::SEARCH_FOOD] {  
        var body = occurrence.body  
  
        moveForward(1)  
        dropOff(new ColonyPheromone)  
        ...  
        synchronizeBody  
    }  
    ...  
}
```



## General Principle

Two ways are available to add turtle agents in the system:

- manual instantiation of a turtle agent through the standard SARL API.
- definition of a spawner: a point or an area where turtle agents could be automatically created at a given generation rate.



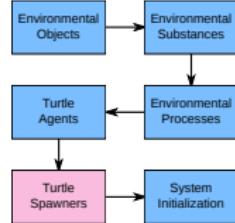
## In the ant colony problem

A spawner is defined for each nest since the Queen is inside.

## Definition of nest spawner

- Located at a specific point on the ground.
- Maximal number of ants to generate.

```
class NestSpawner extends JaakPointSpawner {  
    var budget : int  
    new (environment : EnvironmentArea,  
        budget : int, x : int, y : int) {  
        super(environment, x, y)  
        this.budget = budget  
    }  
    def isSpawnable(timeManager : TimeManager) : boolean {  
        (this.budget > 0)  
    }  
    def computeSpawnedTurtleOrientation(  
        timeManager : TimeManager) : float {  
        RandomNumber::nextFloat() * 2 * Math::PI;  
    }  
    def turtleSpawned(turtle : UUID, body : TurtleBody,  
        timeManager : TimeManager) {  
        this.budget = this.budget - 1  
        body.semantic = typeof(Forager.class)  
    }  
}
```

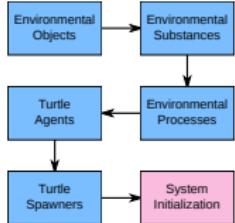


## General Principle

Setting up the system by defining initialization functions.

### In the ant colony problem

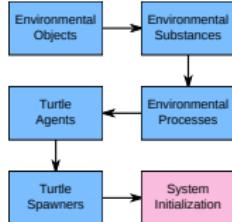
- Extend the Jaak kernel agent with initialization of the nests, spawners.
- Provide the spawner-agent mapping that is used when agents are manually created.



### Initializing the system

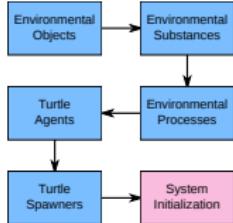
- Create subtype of JaakKernelAgent.
- Create the environment grid.
- Create the spawners on the ground.
- Provide the spawner-agent mapping for manually created agents.
- Initialize the UI and start the simulation.

```
agent AntColonyProblem extends JaakKernelAgent {  
  
    def createEnvironment(tm : TimeManager) : JaakEnvironment {  
        var environment = new JaakEnvironment(WIDTH, HEIGHT)  
        environment.timeManager = tm  
  
        var actionApplier = environment.actionApplier;  
        for(i : 0..99) {  
            actionApplier.putObject(random, random, new Food(50))  
        }  
  
        return environment  
    }  
}
```



### Initializing the system

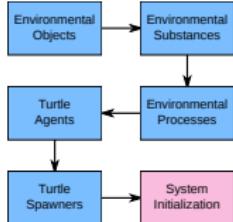
- Create subtype of JaakKernelAgent.
- Create the environment grid.
- **Create the spawners on the ground.**
- Provide the spawner-agent mapping for manually created agents.
- Initialize the UI and start the simulation.



```
agent AntColonyProblem extends JaakKernelAgent {  
    ...  
    def createSpawners : JaakSpawner[] {  
        var spawners = <JaakSpawner>newArrayOfSize(  
            ANT_COLONY_COUNT)  
        for(i : 0 .. spawners.length) {  
            spawners.set(i, createColony(i+1))  
        }  
        return spawners;  
    }  
  
    def createColony(colonyId : int) : JaakSpawner {  
        ...  
    }  
}
```

### Initializing the system

- Create subtype of JaakKernelAgent.
- Create the environment grid.
- Create the spawners on the ground.
- **Provide the spawner-agent mapping for manually created agents.**
- Initialize the UI and start the simulation.

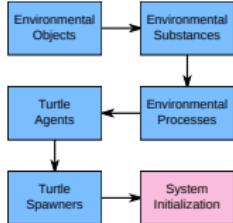


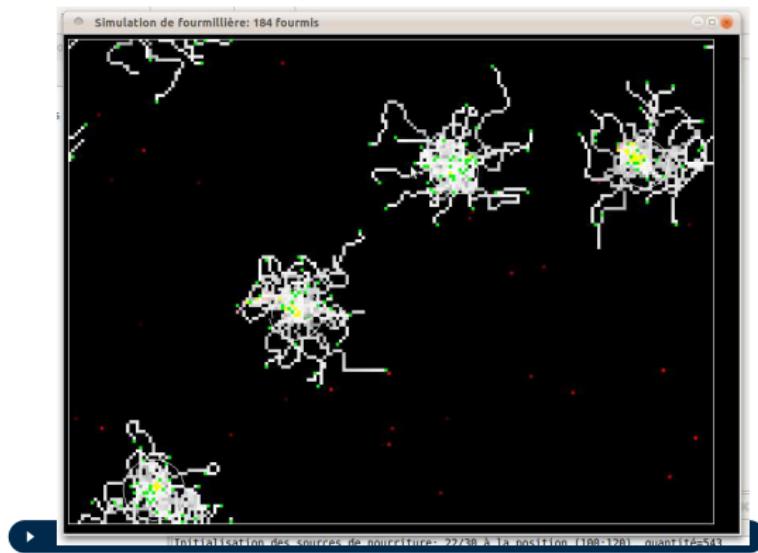
```
agent AntColonyProblem extends JaakKernelAgent {  
    ...  
    def getSpawnableAgentType(spawner : JaakSpawner)  
        : Class<? extends Agent> {  
        return typeof(Ant)  
    }  
}
```

### Initializing the system

- Create subtype of JaakKernelAgent.
- Create the environment grid.
- Create the spawners on the ground.
- Provide the spawner-agent mapping for manually created agents.
- **Initialize the UI and start the simulation.**

```
agent AntColonyProblem extends JaakKernelAgent {  
    ...  
    on Initialize {  
        super._handle_Initialize_0(occurrence)  
        var ui = new UI(controller)  
        addJaakListener(ui)  
        fireEnvironmentChange  
        ui.visible = true  
  
        controller.startSimulation  
    }  
}
```





Official Demo of the Jaak Library  
<https://github.com/gallandarakneorg/jaak>

- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
  - Context and Problems
  - Existing Environment Models
  - Hybrid Environment Model
  - Simulation of an Urban Environment

## Modeling and simulation of individuals in a large-scale virtual world

Reproduction of the dynamics of individuals and groups of individuals evolving in a large-scale virtual world (building, street...).

## Application Domains

- Development of urban sites, security study, flow analysis...
- Interactive staff training, serious games.
- Video games, virtual animation...



- 1 Generating a population of virtual individuals.
- 2 Reproduction of the movements of the individuals.
- 3 Generation of individual and collective behaviors.
- 4 Modeling of a virtual environment.
- 5 Interaction with the virtual population.
- 6 Realistic displaying of the population and the univers.

- 1 Generating a population of virtual individuals.
- 2 Reproduction of the movements of the individuals.
- 3 Generation of individual and collective behaviors.
- 4 Modeling of a virtual environment.
- 5 Interaction with the virtual population.
- 6 Realistic displaying of the population and the univers.

## Crowds and Traffic Simulation

- enables the reproduction of the mobility behavior of individuals and groups of individuals
- may be applied on various types of environments (indoor/outdoor, open/closed space...)

## Motivations

- To provide accurate simulation for risk estimation and infrastructure design.
- To enable to build communication media for stakeholders.

## Adopted Approach

- Simulation of individuals ⇒ Agent-based simulation.

## Goals

Provide the tools for modeling and simulating entities of different types (vehicules, bicycles, pedestrians...) in a virtual world.

## Constraints

- Modeling and simulation relations [Zeigler, 2000].
- Minimize the simulator latency perceived by the immersed user (serious game).
- **Minimize the design time of the environment model** that is built from geographical information systems and airplane photographies.

## Adopted approach

- **Modeling:** Adopt a graph-based model, which avoid redundancy in the model, and easy to edit.
- **Simulation:** Provide accurate and realtime implementation of the environment model, and of the individual behaviors.

### The environment is a grid

- A cell contains a single object.
- The agents are moving in 4 directions.
- The agents may perceive on near cells.

Examples: NetLogo, Jaak (Janus extension)...

But...

- Discrete modeling of the environment: discrete perceptions and actions (motion, etc.)
- Limited size of the environment.
- Not "realistic."



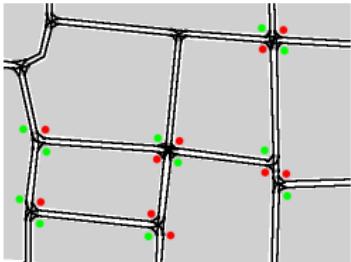
## The environment is a graph

- The road network is modeled with a graph.
- The agents are moving along the graph (in 1D).
- The agents may perceive on near segments.

Examples: S-Paramics, Aimsun, ArchiSim, Vissim, MatSim, Gama, JaSim (Janus extension) . . .

But . . .

- Only for entities that follow paths (car, cycle, train . . .)
- Difficult to simulate motions in 2D.
- Difficult to simulate vehicle overtaking.



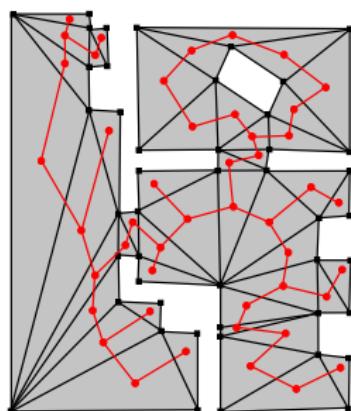
The environment is collection of inter-connected planar zones

- Planar zones are built from the environment topology.
- The agents are moving freely on a zone (in 2D).
- The agents perceive in the zones in intersection with their frustums.

Examples: Place/Portal, PVS, Orca, ClearPath, JaSim (Janus extension)...

But...

- Difficult to restrict motion.
- Need complex algorithm to be built.
- Need information related to the topology: road direction...



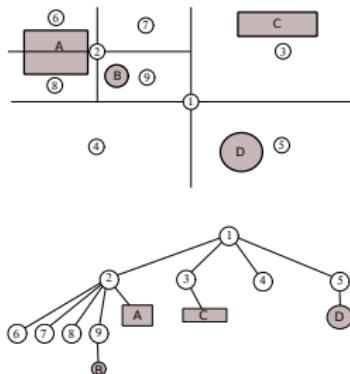
The environment is hierarchically decomposed into zones

- Planar zones are built from a composition of a bigger zone.
- The agents are moving freely on a zone (in 2D/3D).
- The agents perceive in the zones in intersection with their frustums.

Examples: JaSim (Janus extension), Simulate...

But...

- Tree data structure management is costly.
- May be difficult to represent environment topology.
- Need complex algorithm to be built.



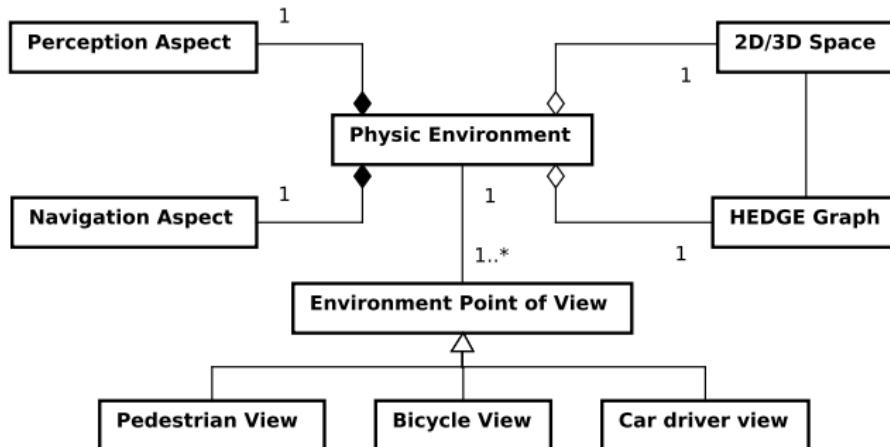
## Problems

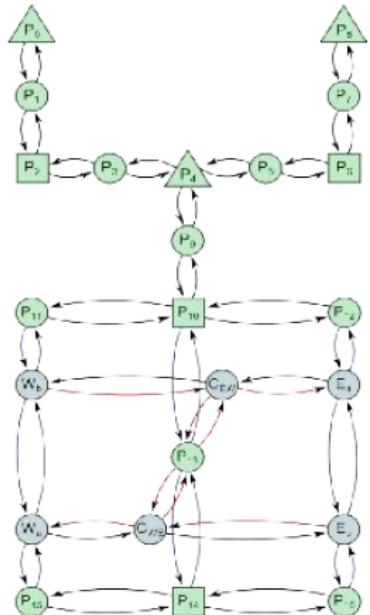
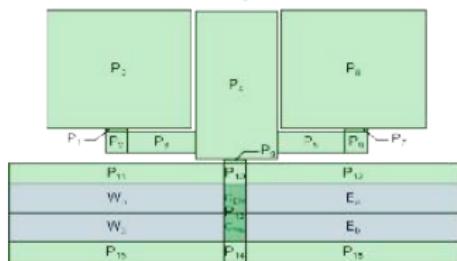
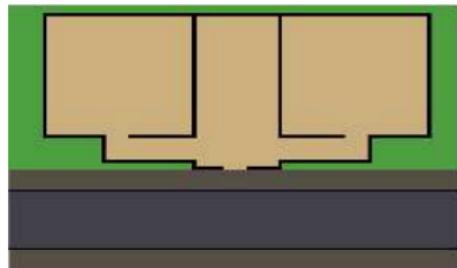
- 1 Difficult for an entity to use different types of zones: how may a bicyclist follow a road, or traverse an open place with the same motion behavior?
- 2 Difficult to use the same zones for different types of entities: how to manage a bicyclist and a pedestrian on the same side-walk?
- 3 Difficult to manage perception and navigation, separately.
- 4 It is time consuming for designing.

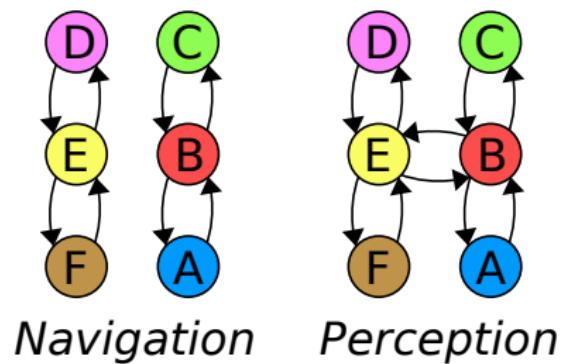
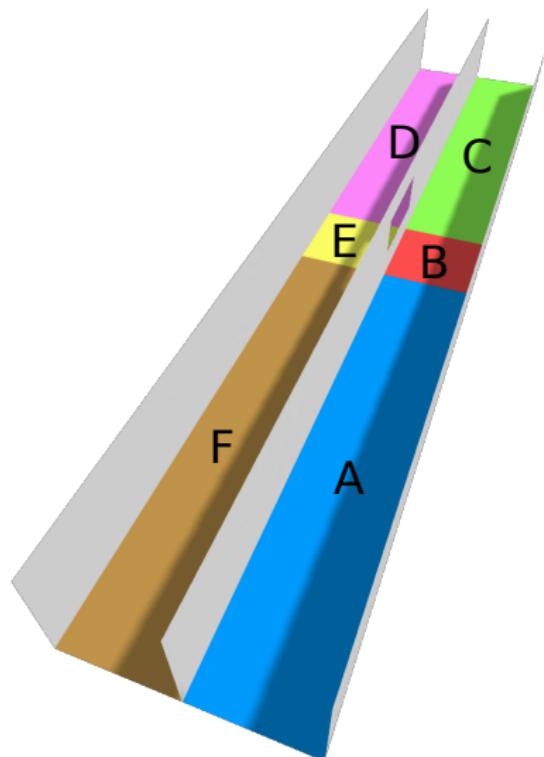
## Proposal [Buisson, 2014]

- Definition of the environment model based on the an hybrid zonal/graph representation.
- Provide tools for easy edition (based on the SIMULATE® framework).

- Different points of view on the Environment.
- Distinction between navigation and perception.
- Hybrid approach: the HEDGE Graphe, and a 2D/3D tree-based space.

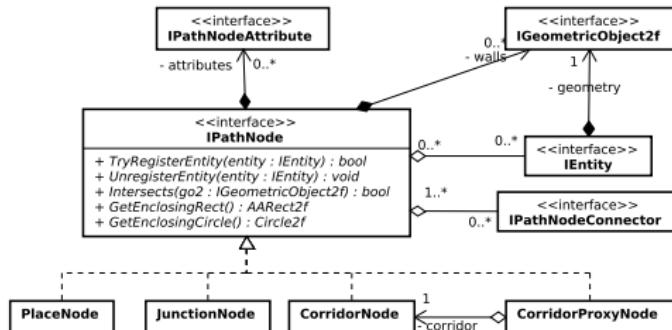






## Types of nodes

- **Corridor:** zone with a general direction (roads, corridors...)
- **Place:** spatial zone that corresponds to an open space.
- **Junction:** zone that permits to connect nodes together.

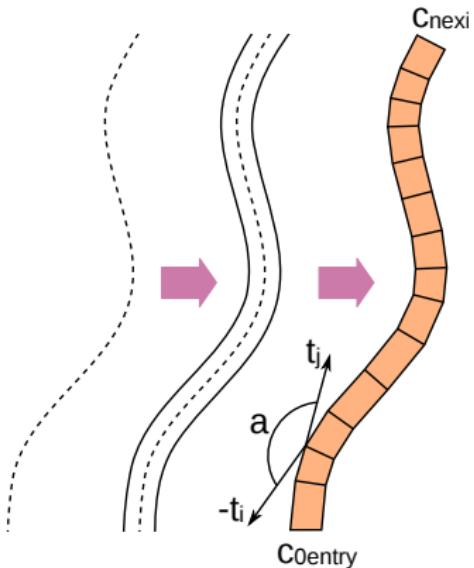


## Definition

- Sequence of shapes:  $\langle s_0, s_1 \dots \rangle$
- $s_i = \langle c_i^{\text{left}}, c_i^{\text{right}}, c_i^{\text{entry}}, c_i^{\text{exit}}, t_i \rangle$
- $c_i^\alpha$ : side of the shape,  $t_i$ : tangent
- $\square \left( (s_i \cap s_{k \neq i}) = \emptyset \wedge c_i^{\text{left}} \parallel c_i^{\text{right}} \wedge c_{i-1}^{\text{exit}} = c_i^{\text{entry}} \wedge c_{i+1}^{\text{entry}} = c_i^{\text{exit}} \right)$
- $c_0^{\text{entry}}$  and  $c_n^{\text{exit}}$  are connectable to one node each.

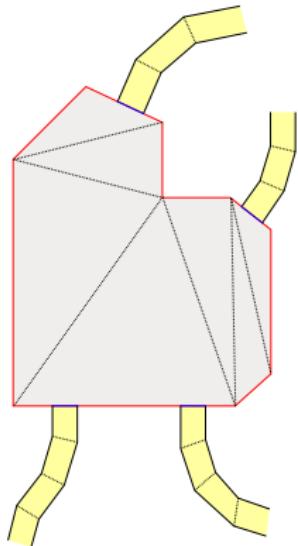
## Building

- 1 Extrusion of a spline or a path.
- 2 Discretization such that  $t_j \cdot t_i \leq \epsilon$



## Definition

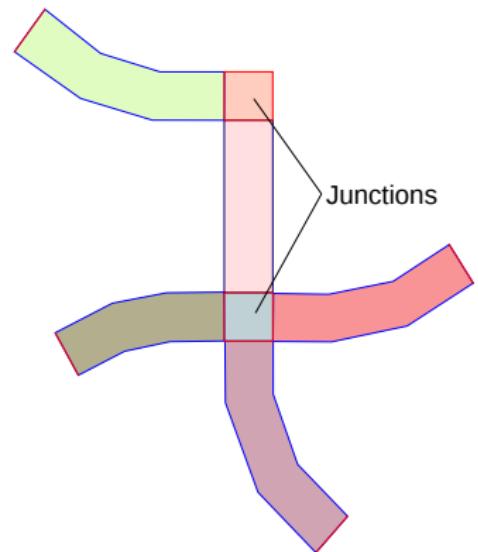
- Set of triangles forming a bounded 2D manifold:  
 $\{t_0, t_1 \dots\}$
- $t_i = \{s_i^0, s_i^1, s_i^2\}$ ,  $s_i^\alpha$  = triangle side
- $\square(t_i \cap t_{k \neq i}) = \emptyset$
- $\square(\nexists k, b/s_i^a = s_{k \neq i}^b \Rightarrow s_i^a \in B)$
- $B$ : sequence of sides that forms the bounds of the place, such that the first segment is connected to the last segment.
- Each element of  $B$  may be connectable to many nodes, without overlapping.



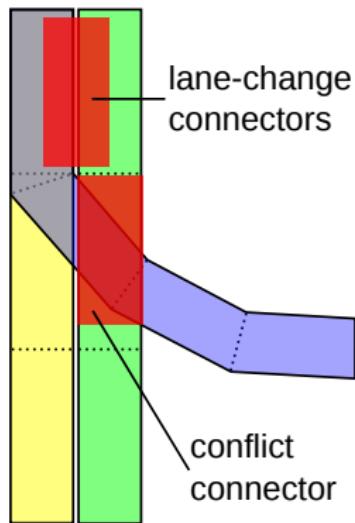
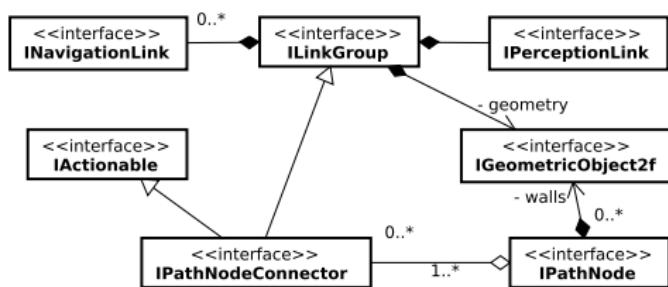
## Definition

Specialization of a place, in which:

- $|B| = 4$
- $\forall b \in B$ , the connected nodes are always the junction and a set of corridors.

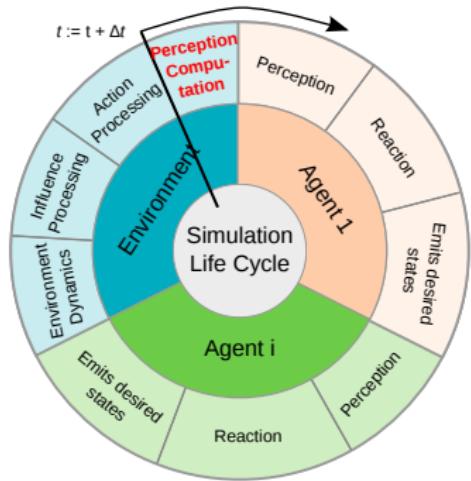


- **Standard connector:** connects two nodes.
- **Lane-change connector:** enables to connect two adjacent corridors.
- **Conflict connector:** add a perception link between two overlapping nodes.



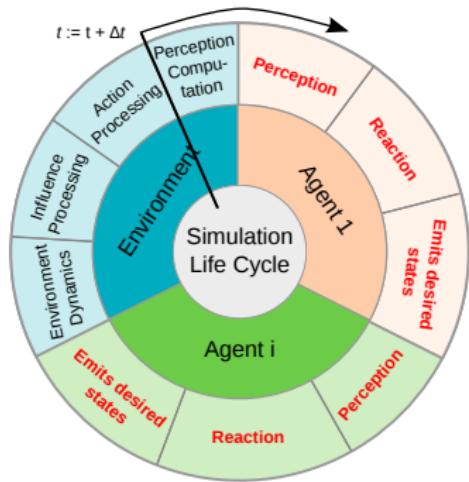
Perception computation for each body  
(parallel algorithm)

- Retreive the physical properties: position, view frustum...
- Traverse the perception graph from the position.
- Apply body's filters on each encountered object.
- Put the collection of perceived objects in the body.



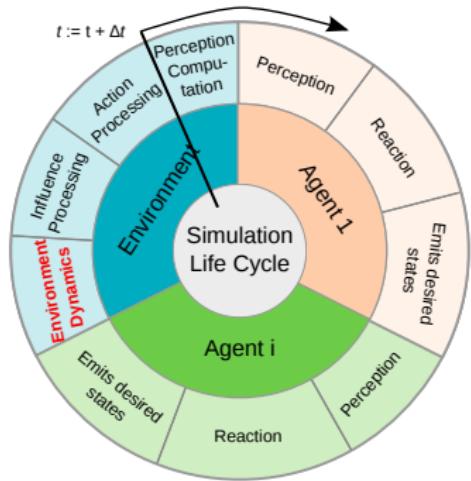
### Agent execution (in parallel)

- Retrive the perception set from the body.
- Compute the next actions  $a$ .
- Convert  $a$  to influences for object motion, or object actioning.
- Put the influences in the body.
- The body is filtering the agent's influences.



## Environment Dynamics (in parallel to agents)

- Run the endogenous processes of the environment: traffic lights, weather...
- For joint actions between the environment and the agents  $\Rightarrow$  the environment must not change its own state directly.
- Generate a collection of influences.



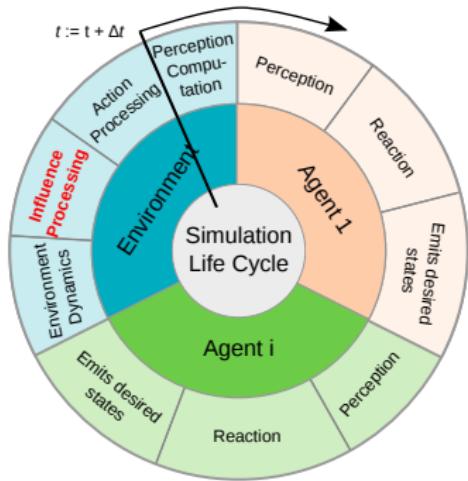
## Influence Processing

- Detect conflict among influences:
  - a) use physic engine<sup>a</sup>, or
  - b) use realtime collision detection algorithm<sup>b</sup>.
  - c) use semaphores or priority queues for selecting the actionner of an object<sup>c</sup>.
- Generate the motion vectors  $m_j$  for all objects in the environment.

<sup>a</sup>PhysX by NVidia...

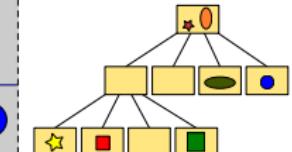
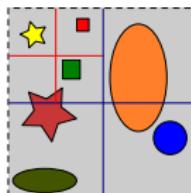
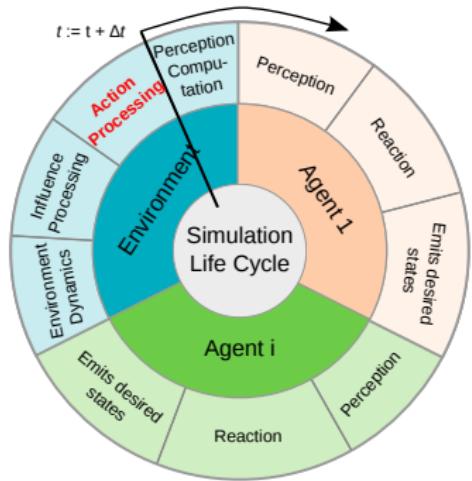
<sup>b</sup>COLLIDE...

<sup>c</sup>Artifacts, Smart Objects...

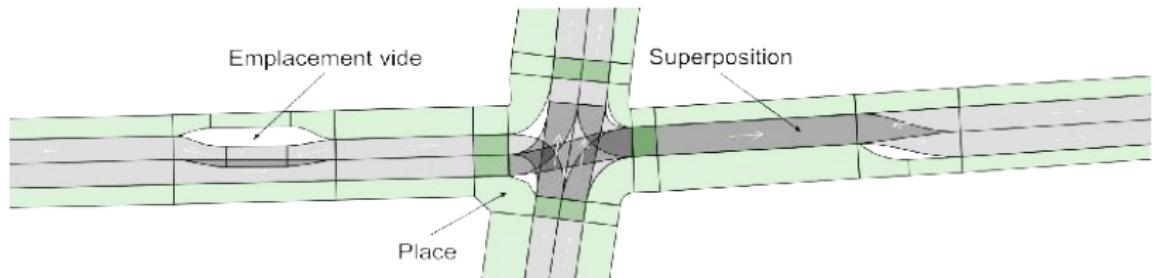
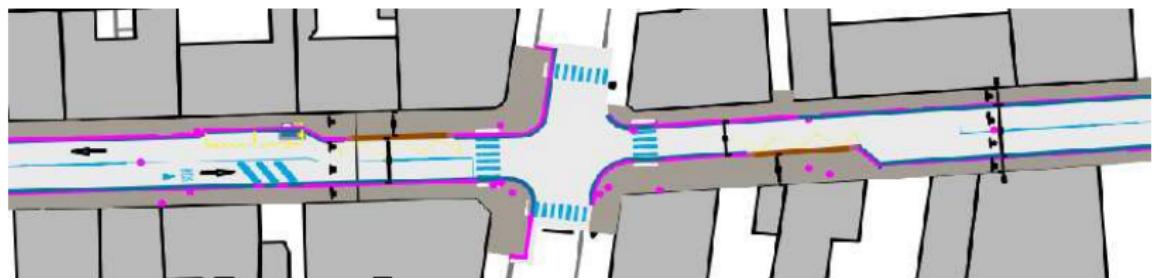


## Action Processing

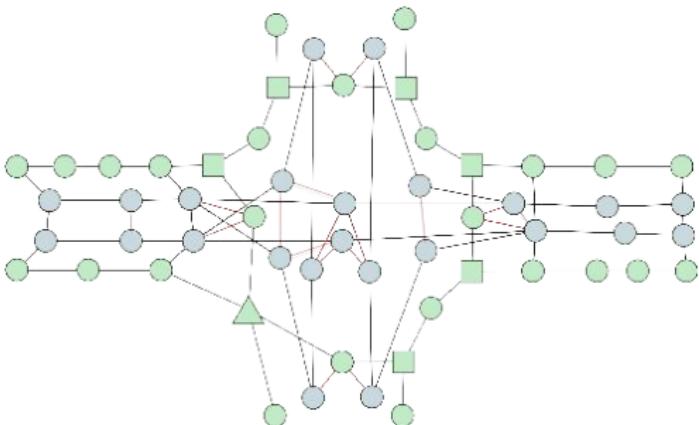
- Apply the motion vector  $m_j$  on the attributes of the body  $j$ .
- Relocate the body on the graph:
  - **Body falls on a node:** Efficient,  $O(2)$ .
  - **Body outside a node:** Relocate to the near node with a spatial tree,  $O(\log_4(n))$ .
- Put the body inside the internal data-structure of the node:  
a spatial tree is used to classify the objects in a node,  $O(\log_4(n))$ .



- 1 Realistic behaviors: use for infrastructure design by experts and stakeholders.
- 2 Quick respond to change requests: less than 2 working days for (re-)designing the simulation environment and a scenario.
- 3 Realtime: use during public meetings with citizens.



- 56 nodes
- 264 links
  - 156 perception links
  - 108 navigation links





These videos were realized on the SIMULATE® tool © Voxelia S.A.S

- 1 Remainders of Multiagent Systems
- 2 From Environment to Agent Environment
- 3 Coordination Artifacts
- 4 Smart Objects
- 5 Agent Body
- 6 Jaak Library: simulation with 2D discrete environment
- 7 3D Urban Environment
- 8 Simulated Multidimensional Environment

- Physical:

- Principles and processes that govern and support a population of entities
- Each agent has a body corresponding to its physical representation [Saunier, 2015].

- Communication:

- Principles, processes and structures to transport information between agents.

- Social:

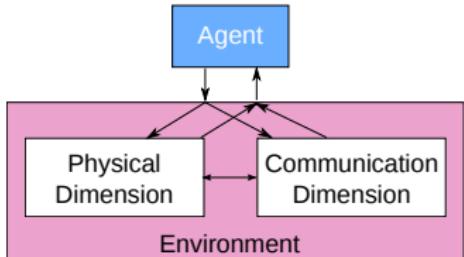
- Principles, processes and structures to support coordinated interaction between agents in a communication environment.

## Hypothesis

A change of state in a dimension can cause a change in another or several dimensions.

## Solution 1

Agent as a propagation vector.



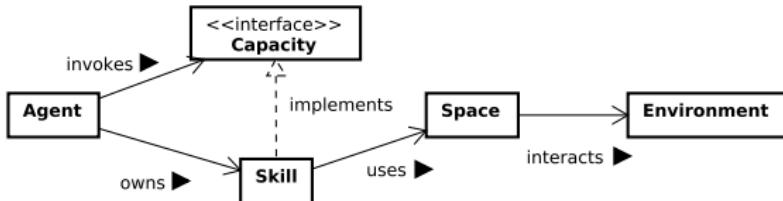
## Solution 2

Interactions between dimensions.

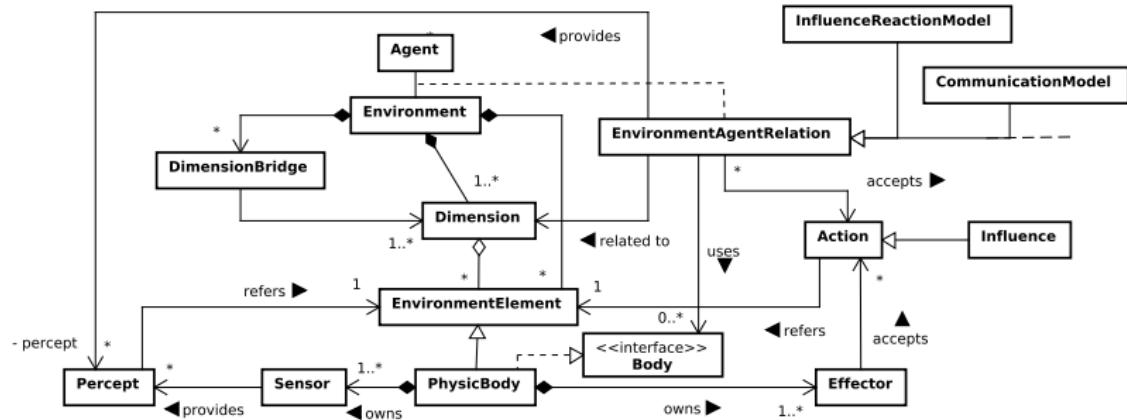
How to model interactions between the dimensions?

- Multi-dimension diffusion / polymorphism:
  - An event/message can be simultaneously interpreted (differently) by several dimensions.
  - Example: GPS Alert may change the social status and spatial indicator of dangerousness.
- Propagation of interactions:
  - An event/message in a given dimension generates another event/message in the other dimension.
  - Example: the detection of physical collision may trigger an emergency message in the communication dimension.
- Constrained perception:
  - A perception of an event/message in a dimension may be constrained by the properties associated with the other dimensions.
  - Example: a traffic light (physical dimension) perceives only emergency vehicles (social status) that are close (physical dimension) to the light.

Modeling based on the concepts of the SARL's metamodel.



- **Space**: interaction place between agents or between agents and their environment.
- **Agent**: autonomous entity with a set of skills to realize the capacities it exhibits.
- **Capacity**: specification of a collection of actions.
- **Skill**: possible implementation of a capacity.  
Skills realized by behaviors mapping a collection of perceptions represented by Events to a sequence of Actions. Event is the specification of some occurrence in a Space that may potentially trigger effects by a listener



“PhysicalSpace” supporting the interaction between agents and the physical dimension

- Emit influences from a specific body.
- Destroy the body related to an agentification.

```
space PhysicalSpace {
    var env : Environment
    def getBodyFactory: PhysicBodyFactory
    {}
    def putEnvironment(body: AgentBody,
                       perceptionListener:
                           Agent) {
        env.add(body).for(perceptionListener)
    }
    def influence(body : AgentBody,
                  influences : Influence*)
    {
        for (i : influences) emit(i, env.
            scope)
    }
    def destroyBody(body : AgentBody) {
        env.destroyObject(body)
    }
}
```

```
skill RoadEnvironmentSkill
implements RoadEnvironmentCapacity
{
    var body : AgentBody
    def install {
        body = bodyFactory.newInstance
        getSpace(PhysicSpace)
            .putEnvironment(body, owner)
    }
    def influence(inf: influence) {
        getSpace(PhysicSpace).influence(body
            , inf)
    }
    def uninstall {
        getSpace(PhysicSpace).destroyBody(
            body)
    }
    ...
}
```

## Exchanges of messages on the Internet

```
space InternetSpace extends EventSpace {  
    val env : Environment  
    def emit(e : Message, scope : Scope) {  
        e.destination = scope  
        super.emit(e, new Scope(env))  
    }  
    def register(agent : Agent) : Address  
    {  
        super.register(agent)  
    }  
    def unregister(agentAddress : Address)  
    {  
        super.unregister(agent)  
    }  
}  
  
event Message {  
    var destination : Scope  
}  
  
capacity InternetCapacity {  
    def emit(e : Message,  
            scope : Scope = null)  
}  
  
skill InternetSkill  
    implements InternetCapacity {  
    def install {  
        getSpace(InternetSpace)  
        .register(owner)  
    }  
    def emit(e : Message, scope : Scope=  
            null) {  
        getSpace(InternetSpace).emit(e,  
                                    scope)  
    }  
    def uninstall {  
        getSpace(InternetSpace).unregister (  
            owner)  
    }  
}
```

## Content

- One instance of the models for each environment dimension.
- Rules of interaction between the dimensions, defined with:
  - a predicate  $p$ : rule activation condition,
  - a function  $f$ : actions to perform when the rule is activated.

## Missions

- To compute the environment reactions from the agent influences.
- To compute the perceptions for each agent in the physical dimension.
- To propagate messages within the communication dimension.

- When receiving an influence, the rules are applied, and the influence is preserved if no rule is deleting it.
- When receiving a message, a similar algorithm is applied.
- Influences and saved messages are stored for later use in the lifecycle.

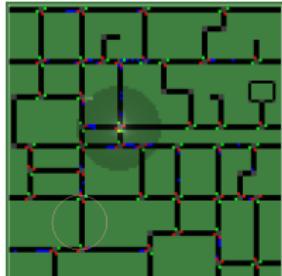
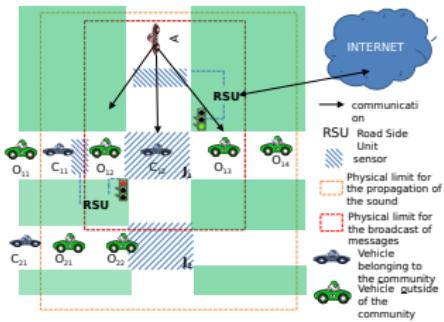
```
behavior Environment {
    var roads : RoadNetwork
    var physicSpace : space
    var communicationSpace: space
    ...
    on Influence {
        if (applyRules(occurrence, occurrence.object)) { saveInfluence(occurrence) }
    }
    on Message {
        for (participant : this.socialSpace.participants) {
            if (occurrence.scope.matches(participant) \&\& applyRules(occurrence,
                participant))
                { saveMessage(occurrence) }
        }
    }
}
```

## Principles

- Simulation of traffic and car crashes.
- Simulation of the displacements of the emergency cars.
- “Green wave” for emergency cars.

## Dimensions of the environment

- Physical: Road network, traffic lights, road sensors.
- Communicational: Wireless Network, RSU, Internet.

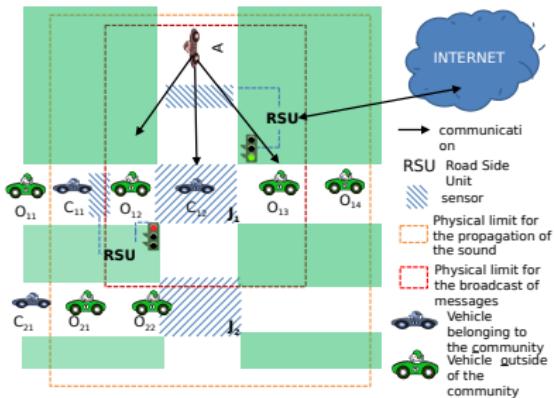


Agent A broadcasts a priority request within its community:

- Priority request sent in the communication dimension, but the broadcast is limited according to the position of vehicles to the physical environment by the V2X propagation model.

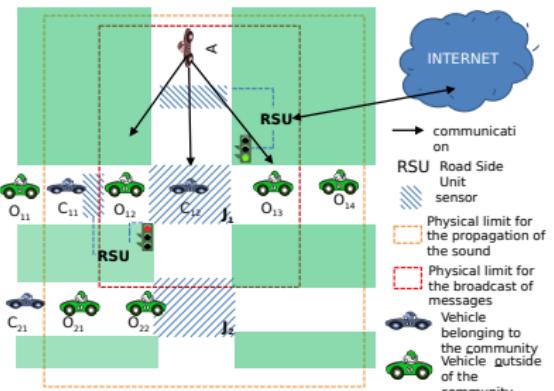
```

rules +=
[ env, e, o |
  e instanceof PriorityRequestMessage
]
=>
[ env, e, o |
  e.scope = Scopes.addresses(
    env.roads.vehiclesAtDistance(
      e.source, env.physicSpace.
      V2X_distance)
  )
]
  
```



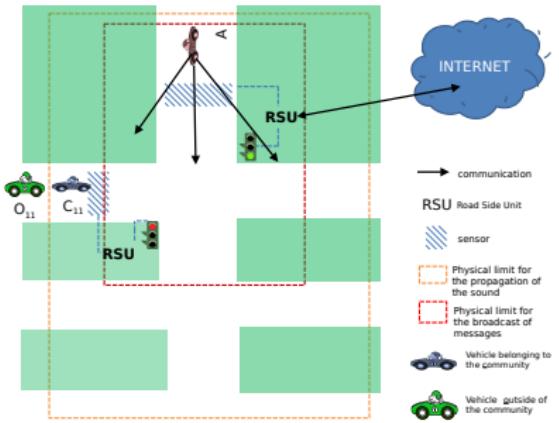
Agent A sends a priority request (resp. Siren influence) that is transformed into Siren influence (resp. priority request).

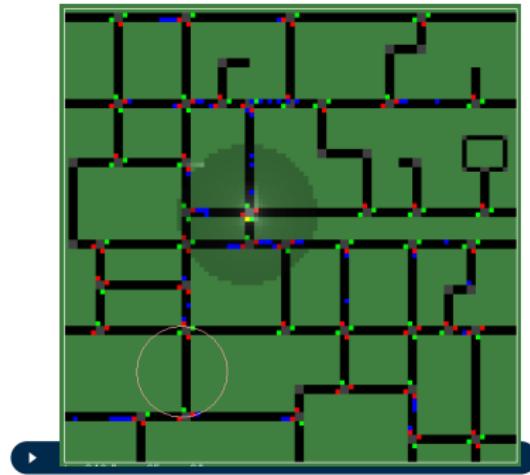
```
rules += [ env, e, o |
           e instanceof Siren ]
=> [ env, e, o |
       env.communicationSpace.emit(
           new PriorityRequestMessage(e.
               source)
       )
   ]
rules += [ env, e, o |
           e instanceof
             PriorityRequestMessage  ]
=> [ env, e, o |
       env.physicSpace.influence(
           new Siren(e.source)
       )
   ]
```



A physical collision involves sending an alert message in the communication dimension.

```
rules +=  
[ env, e, o |  
e instanceof PhysicalCollision ]  
  
=>  
[ env, e, o |  
env.emit( new Alert(e.position))  
]
```





This demo is realized with the Jaak Library

## 1 Methodology:

⇒ How to model the Environment?

## 2 Model:

⇒ Include detailed semantics of objects.

⇒ Towards the use of ontologies.

## 3 Tools:

⇒ Automatic generation of the environment structures.

## 4 Large-scale Environment:

⇒ Supporting large-scale environments with high level of details.



**Thank you for your attention...**



# Organizational and Holonic Modeling

prof.dr.habil. Stéphane GALLAND

During this lecture, I will present:

- 1 the concept of Complex System;
- 2 the organizational modeling principles;
- 3 the definitions of holon and holarchy;
  
- 4 organization examples: broking, carpooling, industrial plant.
- 5 holon examples: industrial plant.

## 1 Complex Systems

## 2 Organizational Modeling

## 3 Holons and Hierarchies

## 1 Complex Systems

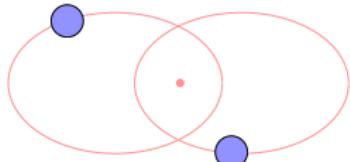
- Examples of Systems
- What is a Complex System?
- The Need of Computational Models

## 2 Organizational Modeling

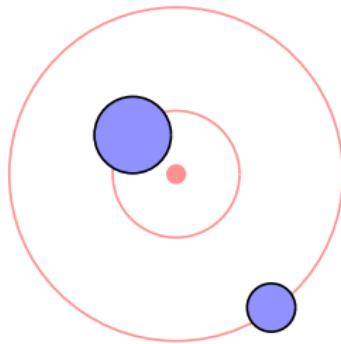
## 3 Holons and Holarchies

- Example: Two-body problem
- Fully solvable and regular trajectories for inverse-square force laws (e.g. gravitational or electrostatic)

$$\begin{cases} F_{12}(x_1, x_2) = m_1 \ddot{x}_1 & \text{(Equation 1)} \\ F_{21}(x_1, x_2) = m_2 \ddot{x}_2 & \text{(Equation 2)} \end{cases}$$

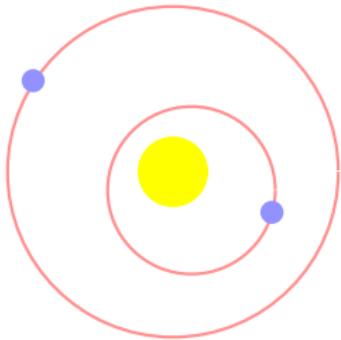


Two bodies with similar mass

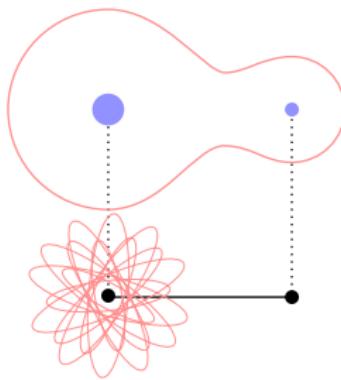


Two bodies with different mass

- Example: Three-body problem
- Generally no exact mathematical solution, even in “restricted” case  $m_1 \ll m_2 \approx m_3$
- must be solved numerically → **chaotic trajectories**

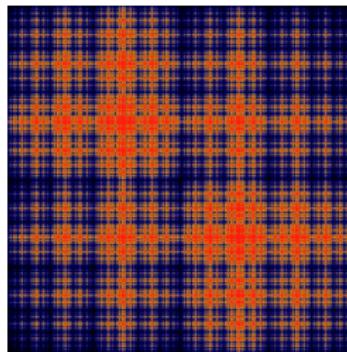


Physics NetLogo Simulation



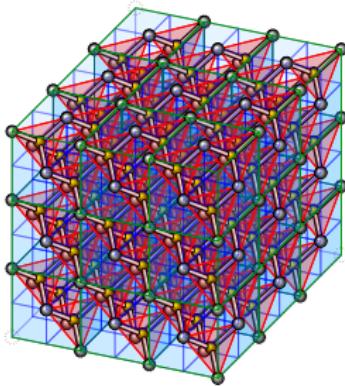
Transit orbit of the planar circular restricted problem

- Example: More Chaos (baker's/horseshoe/logistic maps)
- **chaos** generally means a **bounded, deterministic** process that is **aperiodic** and **sensitive on initial conditions** → small fluctuations create large variations ("butterfly effect")
- Even one-variable iterative functions:  $x_{n+1} = f(x_n)$  can be "complex."

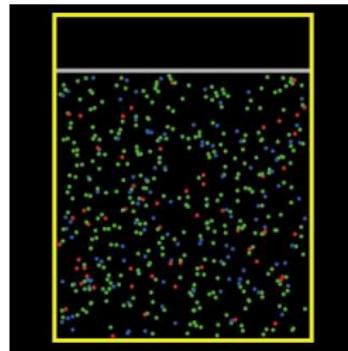


Baker's map

- Examples: Crystal and Gas (Covalent bonds or electrostatic forces)
- Either highly ordered, regular states (crystal).
- or disordered, random, statistically homogeneous states (gas): a few global variables ( $P$ ,  $V$ ,  $T$ ) suffice to describe the system.

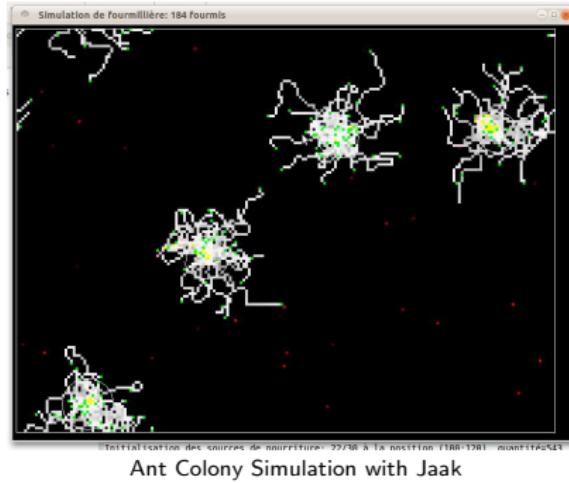


Diamond crystal structure

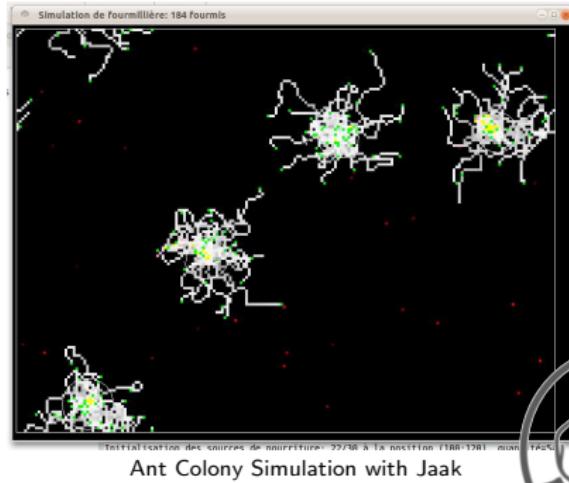


GasLab Isothermal Piston

- Examples: Cellular Automata, Pattern Formation, Swarm Intelligence (insect colonies, neural networks), complex networks, spatial communities
- the typical example of complex systems in this lecture!



- Examples: Cellular Automata, Pattern Formation, Swarm Intelligence (insect colonies, neural networks), complex networks, spatial communities
- the typical example of complex systems in this lecture!



- **Examples:** Natural organisms (cells), societies (individuals + techniques)
- Agent rules become more “complicated”, e.g., **heterogeneous** depending on the element’s **type** and/or **position** in the system.
- Behavior is also complex but, paradoxically, can become more **controllable**, e.g., **reproducible** and **programmable**.



Termite mounds



Companies



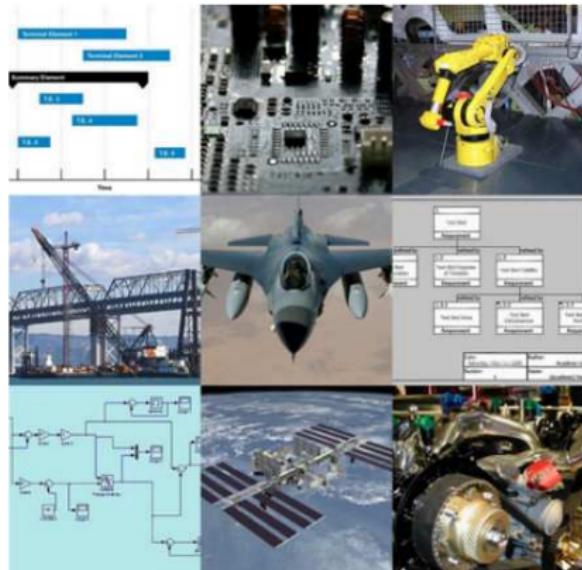
Techno-Networks



Cities

**Examples:** classical engineering, electronics, machinery, aviation, civil construction

- Artifacts composed of a immense number of parts.
- Still designed globally to behave in a limited and **predictable** (reliable, controllable) number of ways
- **Not “complex” systems** in the sense of: little decentralization, no emergence, no self-organization.



Systems' Engineering

- **Examples:** Spectators, orchestras, military, administrations
- People reacting similarly and/or simultaneously to cues/orders coming from a **central cause**: event, leader, plan.
- **Hardly “complex” systems:** little decentralization, little emergence, little self-organization.



Examples	Parts	Local Rules	Emergency	Is a CS?
 2-body problem	few	simple	simple	NO
 3-body problem, low-D chaos	few	simple	complex	NO too small
 crystal, gas	many	simple	simple	NO few params for describing it
 patterns, swarms, complex networks	many	simple	complex	YES but mostly random and uniform
 structured morphogenesis	many	complicated	complex	YES reproducible and heterogeneous
 machines, crowds with leaders	many	complicated	deterministic / centralized	COMPLICATED not self- organized

## 1 Complex Systems

- Examples of Systems
- **What is a Complex System?**
- The Need of Computational Models

## 2 Organizational Modeling

## 3 Holons and Holarchies

### Complexity

measuring the length to describe, time to build, or resources to run, a system

- information theory (Shannon, entropy)
- computational complexity (P, NP)
- Turing machines & cellular automata

### Dynamics

behavior and activity of a system over the time

- nonlinear dynamics & Chaos
- stochastic processes
- systems dynamics (macro variables)

### Adaptation

change in typical functional regime of a system

- evolutionary methods
- genetic algorithms
- machine learning

### Systems Sciences

holistic (non-reductionist) view on interacting parts

- systems theory (von Bertalanfly)
- systems engineering (design)
- cybernetics (Wiener, goals, feedback)
- control theory (negative feedback)

### Multitude, Statistics

large-scale properties of systems

- graph & networks theories
- statistical physics
- agent-based modeling
- distributed AI systems

# COMPLEX SYSTEMS

A complex system is a system that exhibits some (and possibly all) of the following characteristics:

- 1 A big/huge number of interactions among entities.
- 2 Different levels of abstraction.
- 3 Hierarchical organization.
- 4 Emergent organization.
- 5 Local interactions and nonlinear relationships.
- 6 System's components may be considered as complex systems.
- 7 Feedback loops.

## 1 Complex Systems

- Examples of Systems
- What is a Complex System?
- The Need of Computational Models

## 2 Organizational Modeling

## 3 Holons and Holarchies

- We are typically interested in obtaining an explicit description of the behavior of a whole system over time.
- In the case of dynamical systems, this means **solving** their evolution rules, traditionally a set of **differential equations** (DEs).
- Either **ordinary DEs** of **macro-variables** in **well-mixed** systems:
  - **Example:** in chemical kinetics, the law of mass action governing concentrations

$$\alpha A + \beta B \rightarrow \gamma C \text{ described by } d[A]/dt = -\alpha k[A]^\alpha [B]^\beta$$

- **Example:** in economics, (simplistic) laws of gross domestic product (GDP) change

$$dG(t)/dt = \rho G(t)$$

- or **partial DEs** of **local variables** in **spatially extended** systems
  - **Example:** heat equation:  $\delta y/\delta t = \alpha \nabla^2 u$ , wave equation:  $\delta^2 u/\delta t^2 = c^2 \nabla^2 u$
  - **Example:** Navier-Stokes in fluid dynamics, Maxwell in electromagnetism, etc.

### Systems that **no macroscopic quantity** suffices to explain (**ODE**)

- no law of “concentration”, “pressure”, or “gross domestic product.”
- even if global metrics can be designed to give an indication about the system’s dynamical regimes, they rarely obey a given equation or law.

### Systems that require a **non-Cartesian** decomposition of space (**PDE**)

- network of irregularly placed or mobile agents.

### Systems that contain **heterogeneity**

- segmentation into different types of agents.
- at a fine grain, this would require a “patchwork” of regional equations, e.g. embryo.

### Systems that are dynamically **adaptive**

- the topology and strength of the interactions depend on the short-term activity of the agents and long-term “fitness” of the system in its environment.

## Decomposition

- Dividing the problem into sub-problems. Each sub-problem being easier to study, relatively isolation of each sub-pb.
- Limiting the view of the designer to only one part of the problem.

## Abstraction

- Simplifying model of the system by focusing only on certain details or properties.
- Limiting the view of the designer to the main aspects of the problem.

## Organization

- Identifying the relationships between the different components of the problem.
- Grouping a number of basic elements, and treating them as a “whole” at a higher level of abstraction.
- Describing the high-level relationships between the various components. e.g. a number of elements may need to work together to provide a particular functionality.

## Decomposition

- Dividing the problem into sub-problems. Each sub-problem being easier to study, relatively isolation of each sub-pb.
- Limiting the view of the designer to only one part of the problem.

## Abstraction

- Simplifying model of the system by focusing only on certain details or properties.
- Limiting the view of the designer to the main aspects of the problem.

## Organization

- Identifying the relationships between the different components of the problem.
- Grouping a number of basic elements, and treating them as a “whole” at a higher level of abstraction.
- Describing the high-level relationships between the various components. e.g. a number of elements may need to work together to provide a particular functionality.

## Decomposition

- Dividing the problem into sub-problems. Each sub-problem being easier to study, relatively isolation of each sub-pb.
- Limiting the view of the designer to only one part of the problem.

## Abstraction

- Simplifying model of the system by focusing only on certain details or properties.
- Limiting the view of the designer to the main aspects of the problem.

## Organization

- Identifying the relationships between the different components of the problem.
- Grouping a number of basic elements, and treating them as a “whole” at a higher level of abstraction.
- Describing the high-level relationships between the various components. e.g. a number of elements may need to work together to provide a particular functionality.

[Henderson-Sellers, 2005]

MAS are considered as adapted for modeling complex systems.

Multiagent systems are well suited for:

- managing the heterogeneous nature of the system components.
- modeling the interactions between these components.
- trying to understand the emergent phenomena that result from these interactions.



**CS Science: understand "natural" CS**  
→ *Agent-Based Modeling (ABM)*



## Computational Complex Systems



**CS Engineering: design a new generation  
of "artificial" CS**  
→ *Multi-Agent Systems (MAS)*



## 1 Complex Systems

## 2 Organizational Modeling

- Principles
- Example 1: Market-like Community
- Example 2: Carpooling Simulation
- Example 3: Simulation of an Industrial Plant

## 3 Holons and Holarchies

## 1 Complex Systems

## 2 Organizational Modeling

### ■ Principles

- Example 1: Market-like Community
- Example 2: Carpooling Simulation
- Example 3: Simulation of an Industrial Plant

## 3 Holons and Holarchies

## Change of Paradigm [Argente, 2006]

- **From:** an approach focusing on agents and individual aspects.
- **To:** an approach focusing on organizations in which agents form groups and hierarchies.

## Principle [Jennings, 2000, Zambonelli, 2003]

Each agent plays specific roles, and interacts with other agents in the context of an organization.

## Used for:

- Describing the organizational structures of complex systems.
- Decreasing the modeling complexity using organizations as design patterns/templates.

- Recently, a particular interest has been given to the use of **organizational concepts** within MAS where the concepts of “organizations”, “groups”, “communities”, “roles”, “functions”, etc. play an important role.
- They permits to break the design complexity down by focusing on the organizations, the roles, and the interactions between the roles.
- Several works from literature:
  - Agent-Group-Role (AGR) [Ferber, 1998, Ferber, 2004]
  - Moise [Hannoun, 2000, Hübner, 2002]
  - **Capacity-Role-Interaction-Organization** [Hilaire, 2000, Cossentino, 2010]

## Organization

- An organization is defined by a **collection of roles** that take part in systematic institutionalized patterns of interactions with other roles in a common context.
- This context consists in shared knowledge and social rules/norms, social feelings, etc and is defined according to an ontology.
- The aim of an organization is to fulfill some requirements.

Source: <http://www.aspecs.org>, and [Cossentino, 2010]

## Role

- An **expected behavior** (a set of role tasks ordered by a plan) and a set of rights and obligations in the organization context.
- The goal of each role is to contribute to the fulfillment of (a part of) the requirements of the organization within which it is defined.
- A role can be instantiated either as a Common Role or Boundary Role.

Source: <http://www.aspecs.org>, and [Cossentino, 2010]

## Group

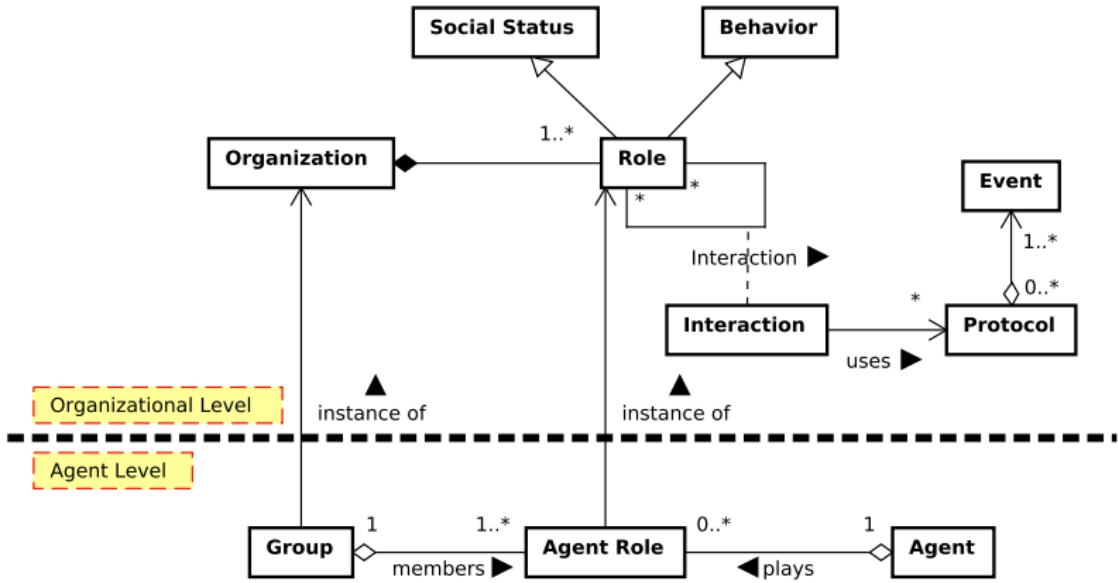
- An instance [...] of an Organization [...].
- It is used to model an aggregation of roles played by agents.

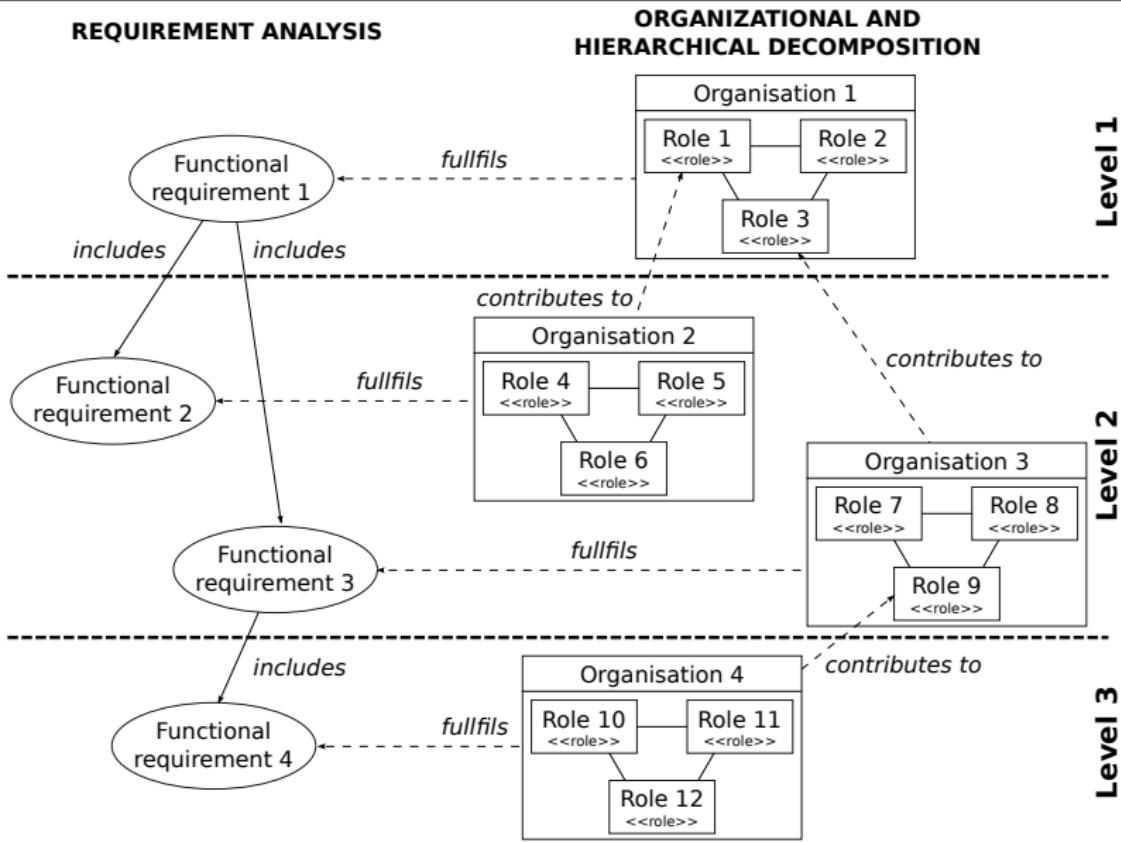
Source: <http://www.aspecs.org>, and [Cossentino, 2010]

## Agent

- An entity that adopts a decision in order to obtain the satisfaction of one or more of its own goals.
- An agent may play a set of roles within various groups.
- These roles interact each other in the specific context provided by the entity itself.
- The entity context is given by the knowledge, the capacities owned by the entity itself.
- Roles share this context by the simple fact of being part of the same entity.

Source: <http://www.aspecs.org>, and [Cossentino, 2010]



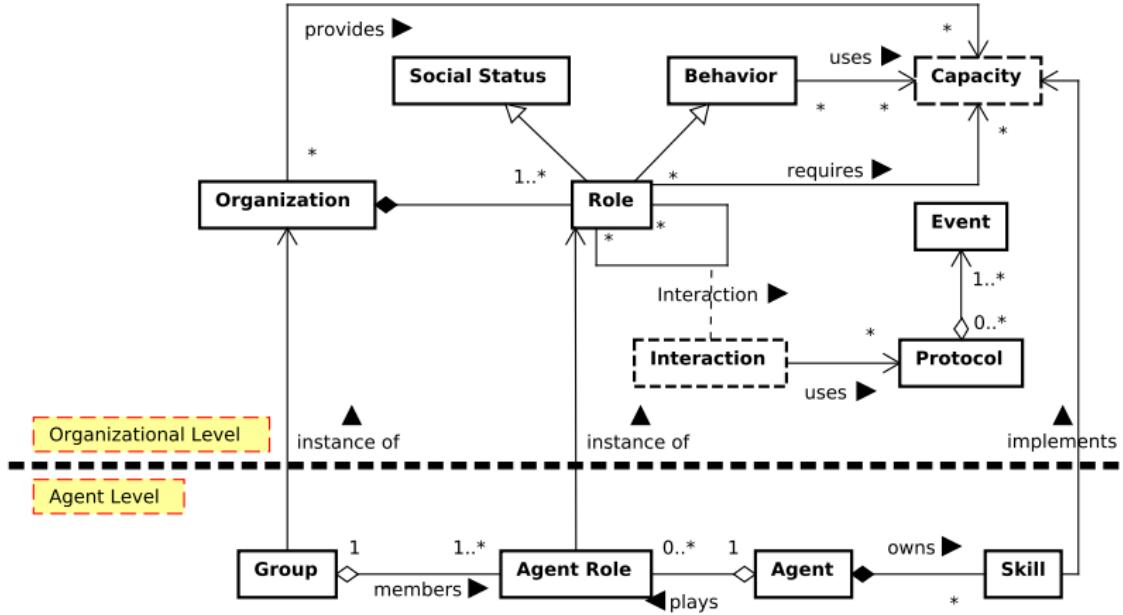


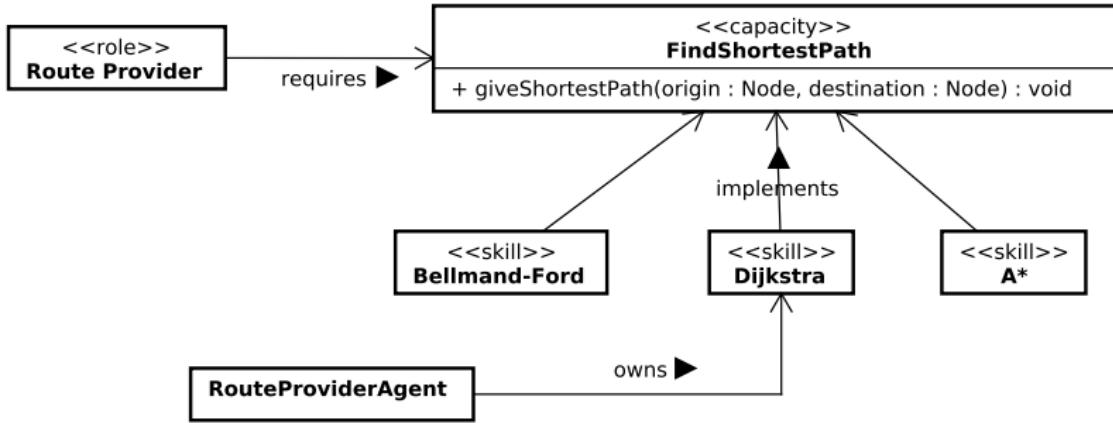
## Capacity

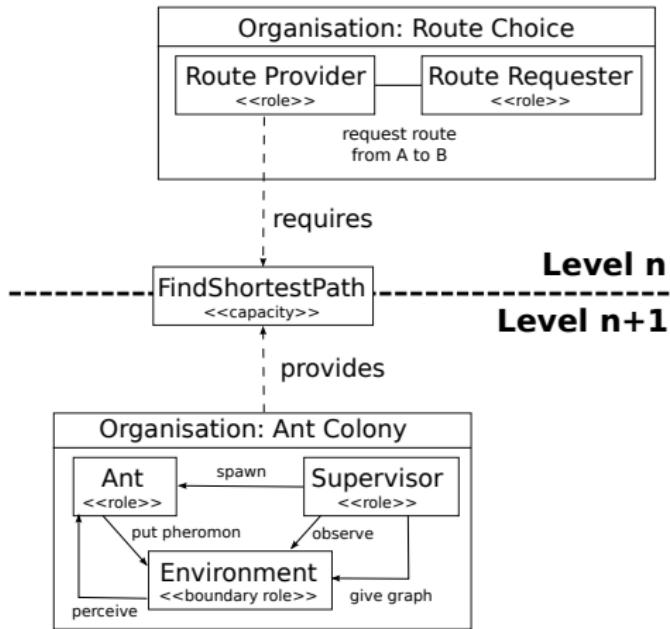
- describes what a behavior is able to do or what a behavior may require to be defined
- Requires by a role
- Provides by an agent or an organization

Double function within CRIo:

- 1 interface between the agent and the role: to define the role behavior disregarding the architecture of the agent.
- 2 interface between two adjacent levels of abstraction in the organizational hierarchy of the system.







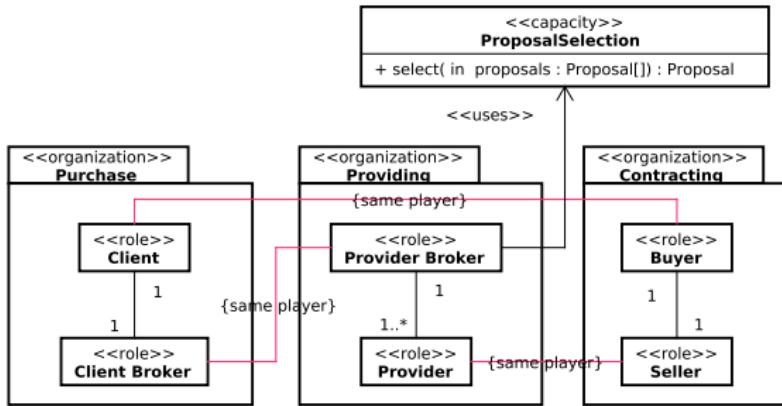
## 1 Complex Systems

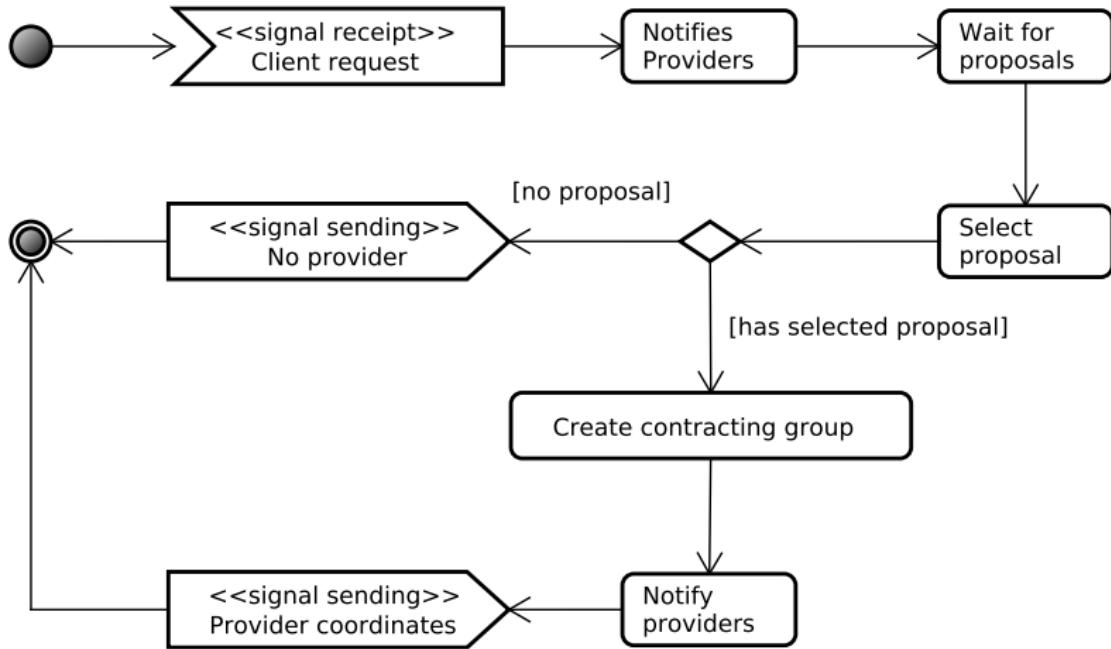
## 2 Organizational Modeling

- Principles
- Example 1: Market-like Community
- Example 2: Carpooling Simulation
- Example 3: Simulation of an Industrial Plant

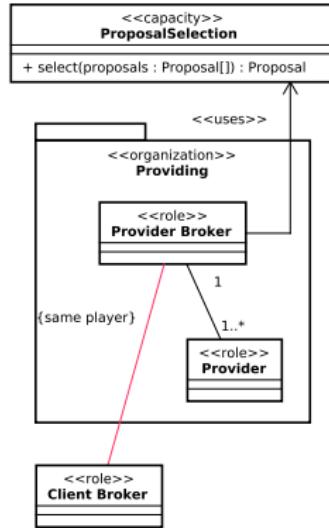
## 3 Holons and Holarchies

- 1 Client is requesting to Broker the best offer for a service.
- 2 Broker is contacting Providers for offer submission and select the best one.
- 3 Broker is giving back to Client the best offer.
- 4 Client and Provider are contracting.





```
behavior ProviderBroker {  
  
    uses ProposalSelection  
  
    var proposals : Collection<Proposal>  
    var providing : ProvidingSpace  
  
    on ClientRequest {  
        providing.emit( new Call(occurrence) )  
  
        in(1.hours) [  
            var best = select(proposals)  
            var replyEvent = new Contracting {  
                request => occurrence,  
                partner => best.source  
            }  
            wake( replyEvent )  
        ]  
    }  
  
    on Proposal {  
        proposals = proposals + occurrence  
    }  
}
```



## 1 Complex Systems

## 2 Organizational Modeling

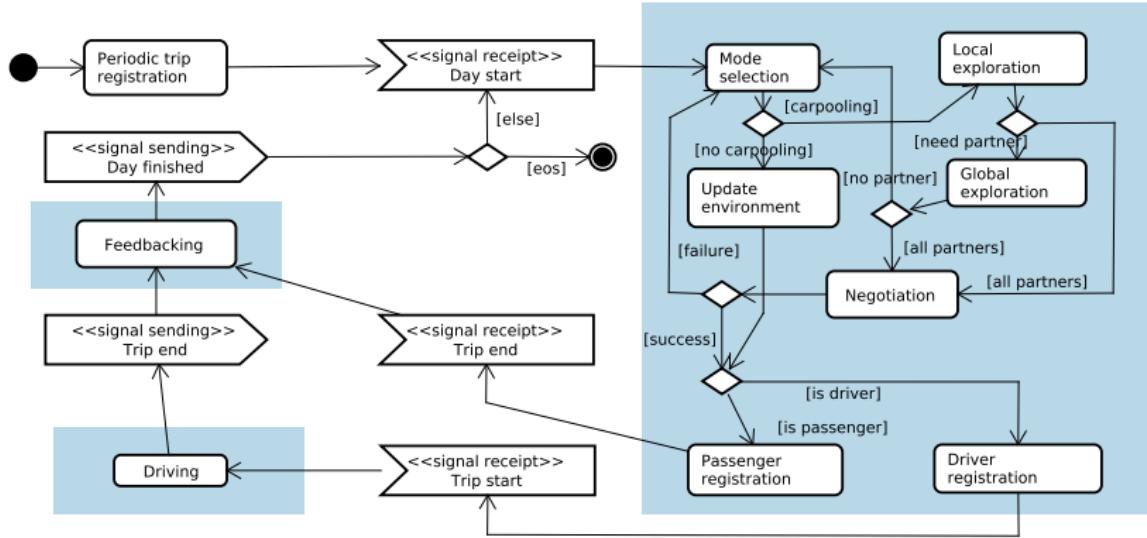
- Principles
- Example 1: Market-like Community
- **Example 2: Carpooling Simulation**
- Example 3: Simulation of an Industrial Plant

## 3 Holons and Holarchies

For each day, for each individual

- 1 Select the best transport mode according to the individual characteristics.
- 2 Create a motive to carpool.
- 3 Communicate this motive with other agents.
- 4 Negotiate a plan with the interested agents.
- 5 Execute the agreed plans.
- 6 Provide a feedback to all concerned agents.

[Galland, 2013b]



## Goal

- Explore the social network of the agent to determine potential carpooling partners.
- If no partner was found, explore the global trip database (website...)

## Partner Selection

Three similarities are used for potential matchings:

- 1 Profile similarity,
- 2 Path similarity, and
- 3 Time window similarity.

## Profile of the agent $A$

Set of attributes, named  $a_A$  with  $|a_A| = N_A$

## Profile Similarity

- Distance between two attribute sets  $a_0$  and  $a_1$ :

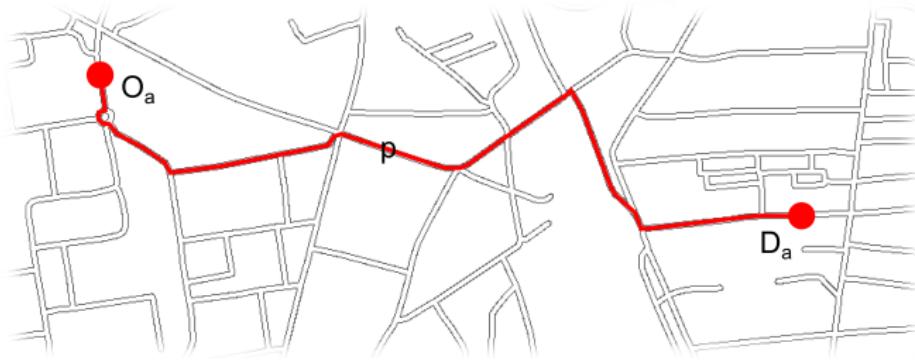
$$d(a_0, a_1) = \sqrt{\frac{\sum_{i \in a_0 \cap a_1} (a_0[i] - a_1[i])^2}{|a_0 \cap a_1|}}$$

- Continuous variables are combined into a single distance value  $d_C$
- Discrete variables are combined into a single value  $d_D \in [0, 1]$
- Similarities:
  - $s_C = 1 - d_C$
  - $s_D = 1 - d_D$

- A sequence of segments of road, starting from  $O_a$  and finishing at  $D_a$ :

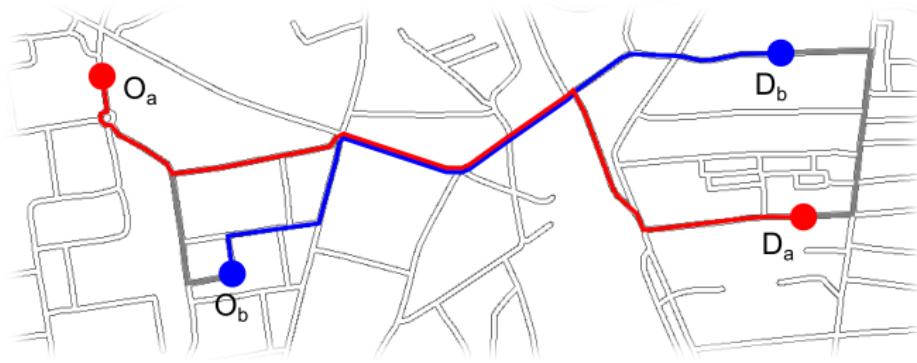
$$p_a = \langle O_a \rangle.p.\langle D_a \rangle$$

- Most of the time, the shortest paths between  $O_a$  and  $D_a$ .



$$\text{pathSim}(p_a, p_b) = \frac{c(O_a, D_a)}{c(O_a, O_b) + c(O_b, D_b) + c(D_b, D_a)}$$

where  $c(i, j)$  denote the cost from the length of the path from  $i$  to  $j$  and the corresponding travel duration.

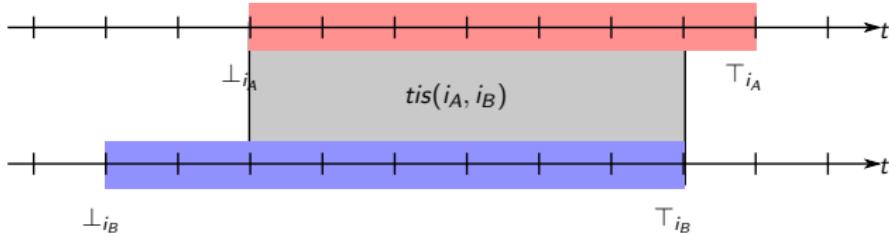


### Time window of a trip

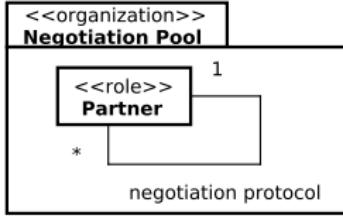
- a trip  $A$  is defined by a path and a time window  $i_A = [\perp_{i_A}, \top_{i_A}]$

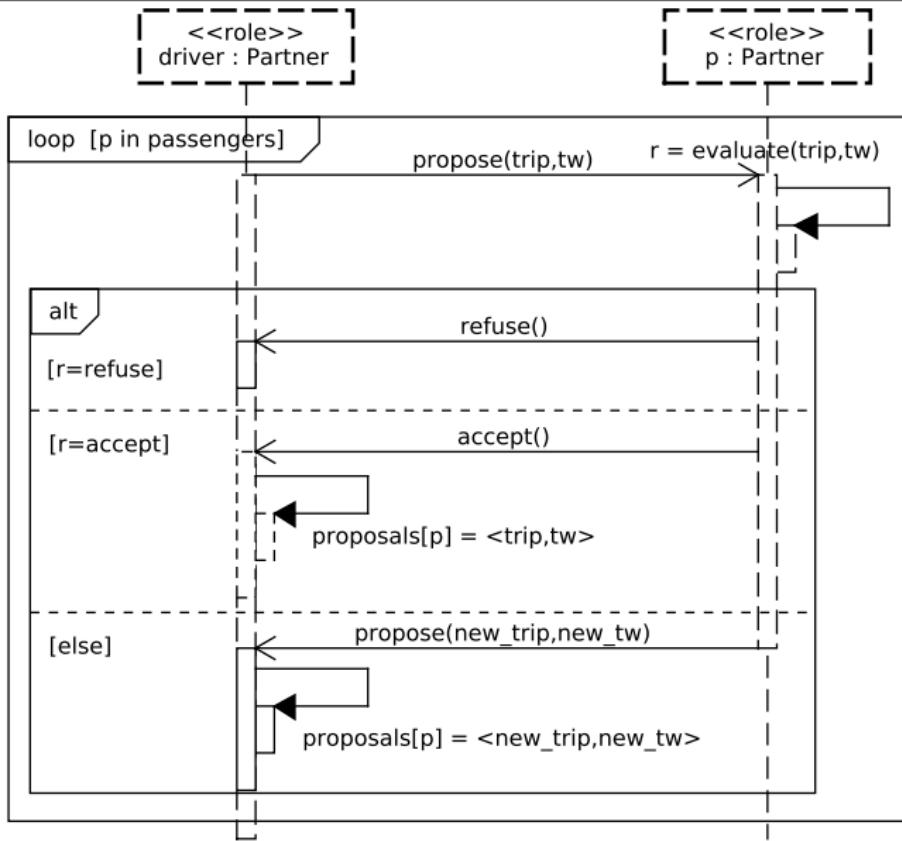
### Time Interval Similarity

$$tis(i_A, i_B) = \min(\top_{i_A}, \top_{i_B}) - \max(\perp_{i_A}, \perp_{i_B})$$



- Each potential partner is invited to play the role Partner in the Negotiation Pool organization.
  - Organizational metamodel  
Capacity-Role-Interaction-Organization (CARIO)  
[Cosentino, 2010]
- They interact together to negotiate the time window of the trip.





```

event Proposal {
    var preferences : Profile
    var path : Path
    var timeWindow : TimeWindow
}

event Refuse
event Accept {
    var proposal : Proposal
}

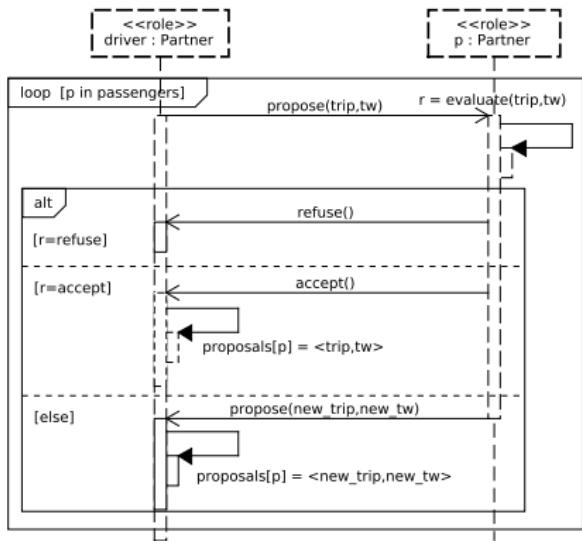
behavior DriverPartner {
    val carpoolingSpace : EventSpace

    on StartNegotiation {
        var proposal = new Proposal(
            owner.profile, occurrence.
            path,
            occurrence.path)
        this.carpoolingSpace.emit(proposal)
    }

    on Refuse {
        releaseRole()
    }

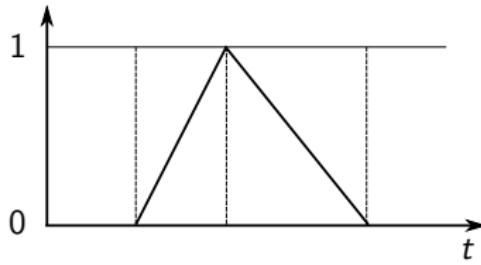
    on Accept {
        owner.addProposal(occurrence.
            proposal)
    }
...
}

```



- Each agent  $A$  gives its preferences for the time of boarding/alighting:

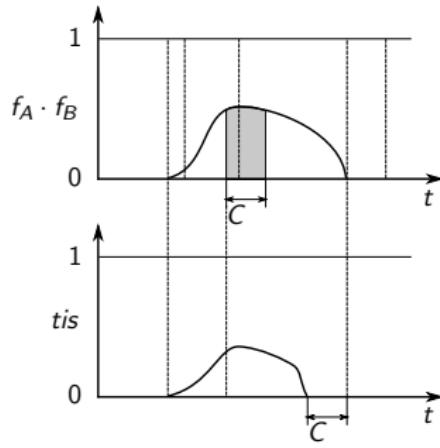
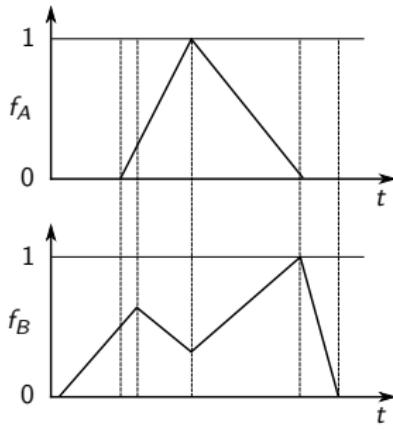
$$f_A : \mathbb{R} \Rightarrow \mathbb{R} : t \mapsto f_A(t) \in [0, 1]$$



- Evaluate the suitability of times for boarding/alighting:

$$\begin{cases} \int_t^{t+C} f_A(x) \cdot f_B(x) dx & \text{if } t \in [\max(\perp_{i_A}, \perp_{i_B}), \min(\top_{i_A}, \top_{i_B}) - C] \\ 0 & \text{otherwise} \end{cases}$$

- Each agent uses this suitability to slightly adapt to the time window.

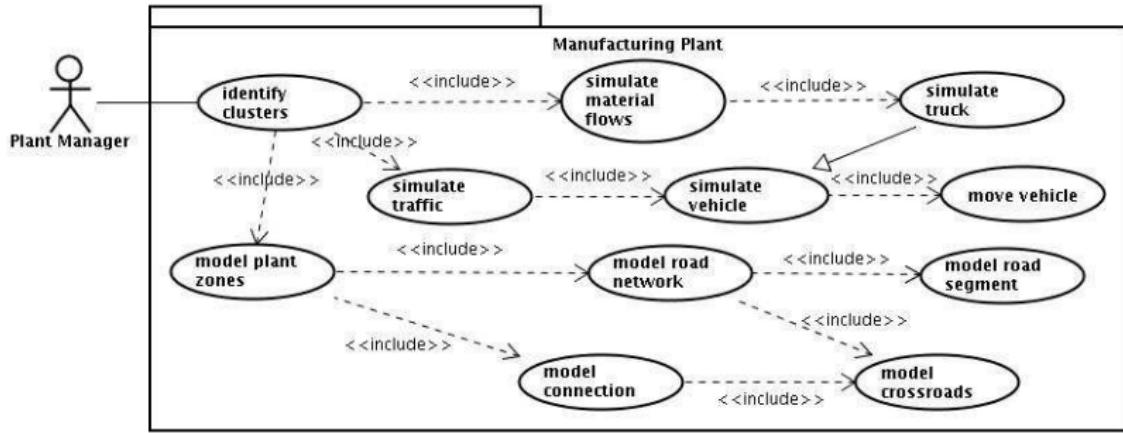


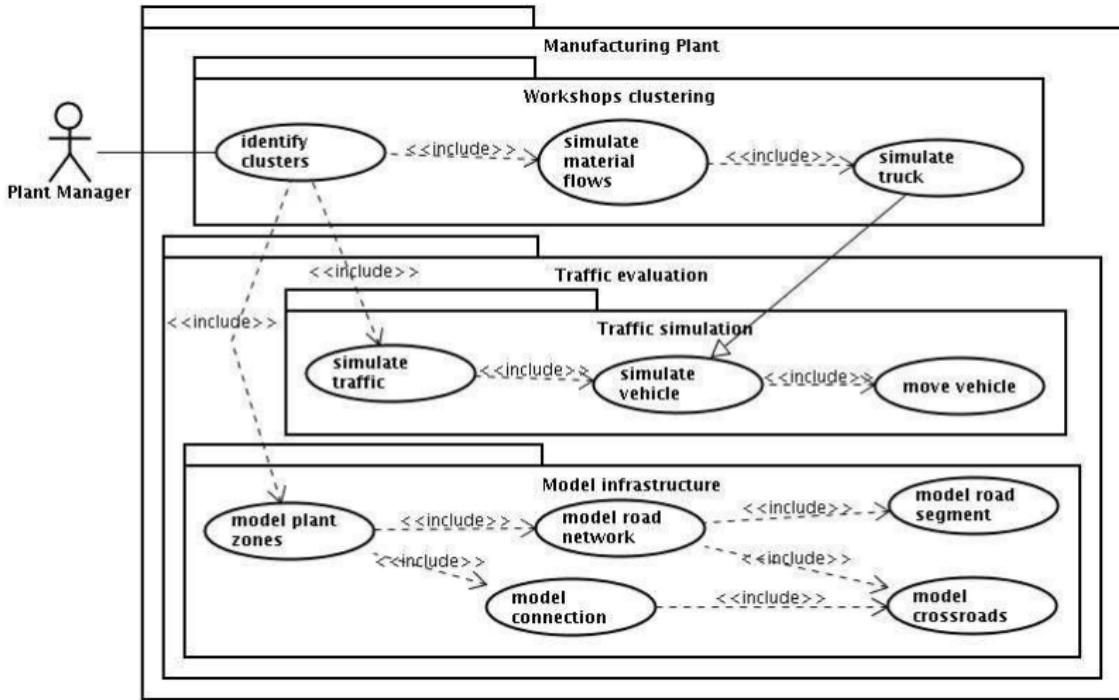
## 1 Complex Systems

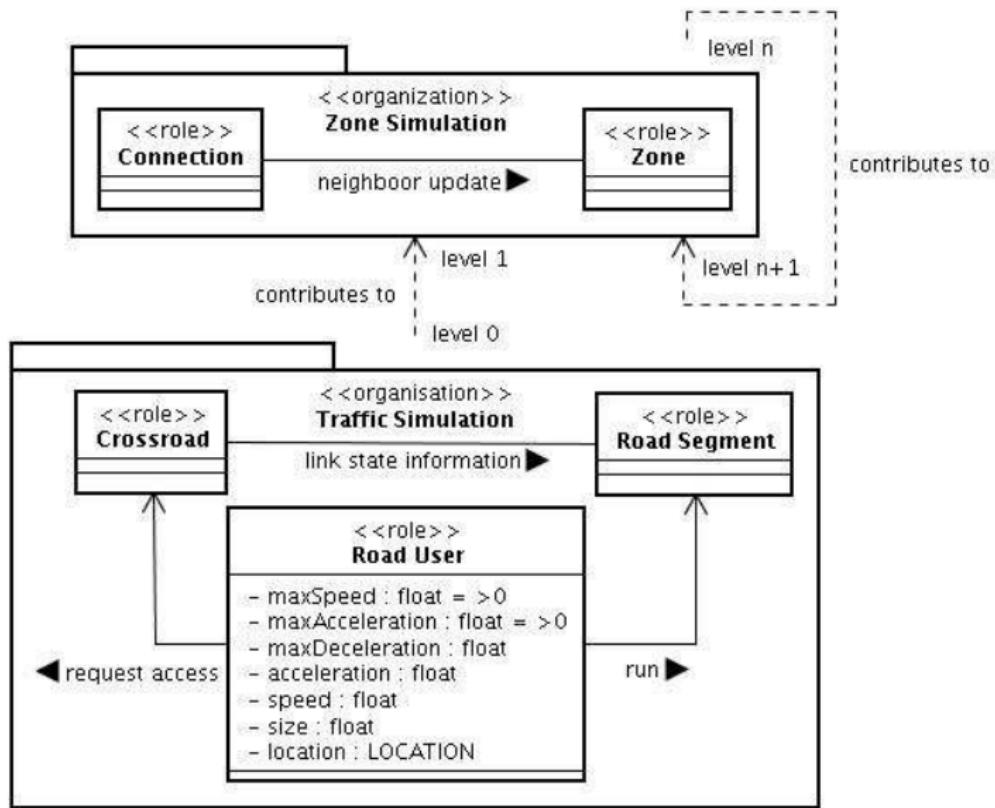
## 2 Organizational Modeling

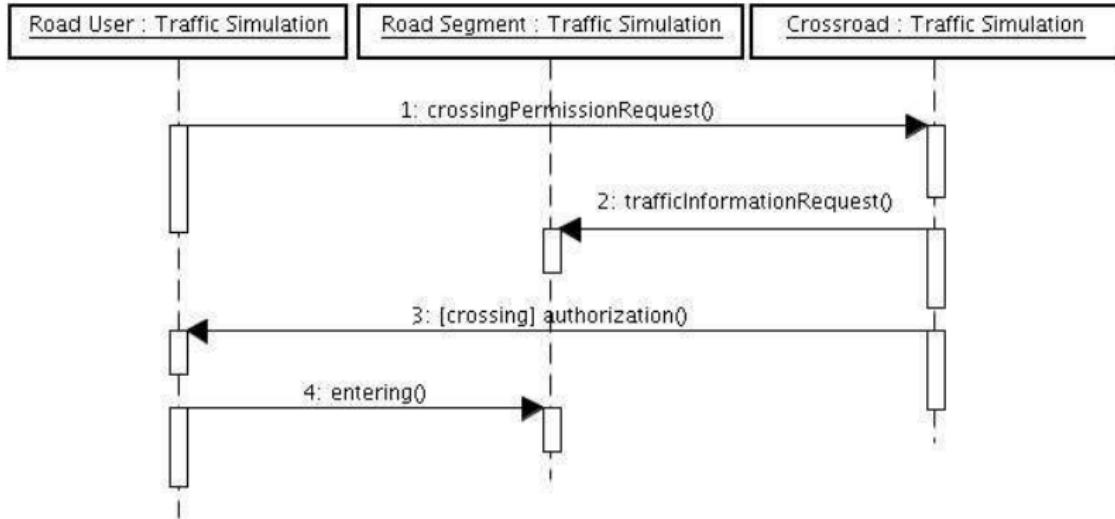
- Principles
- Example 1: Market-like Community
- Example 2: Carpooling Simulation
- Example 3: Simulation of an Industrial Plant

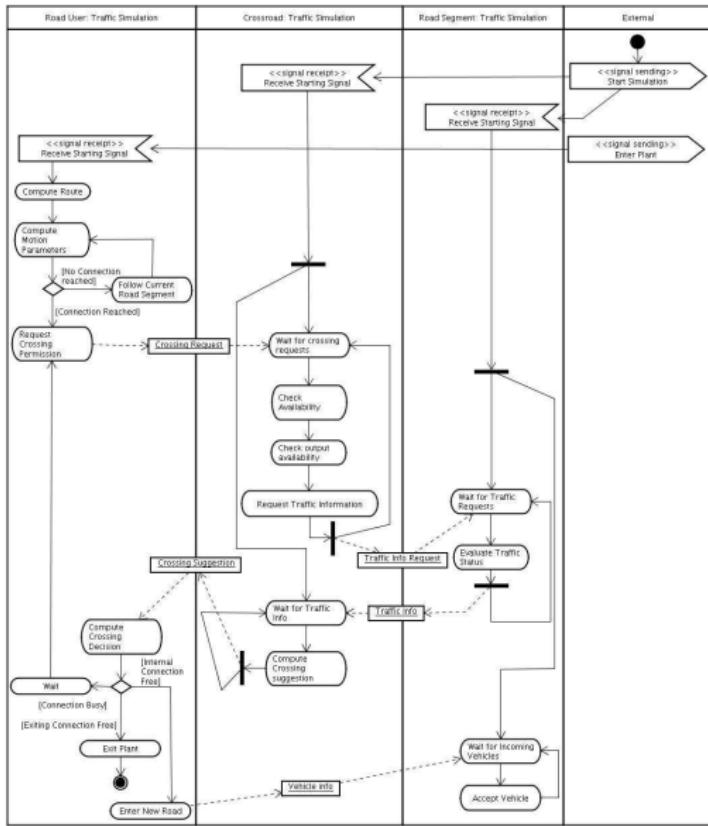
## 3 Holons and Holarchies

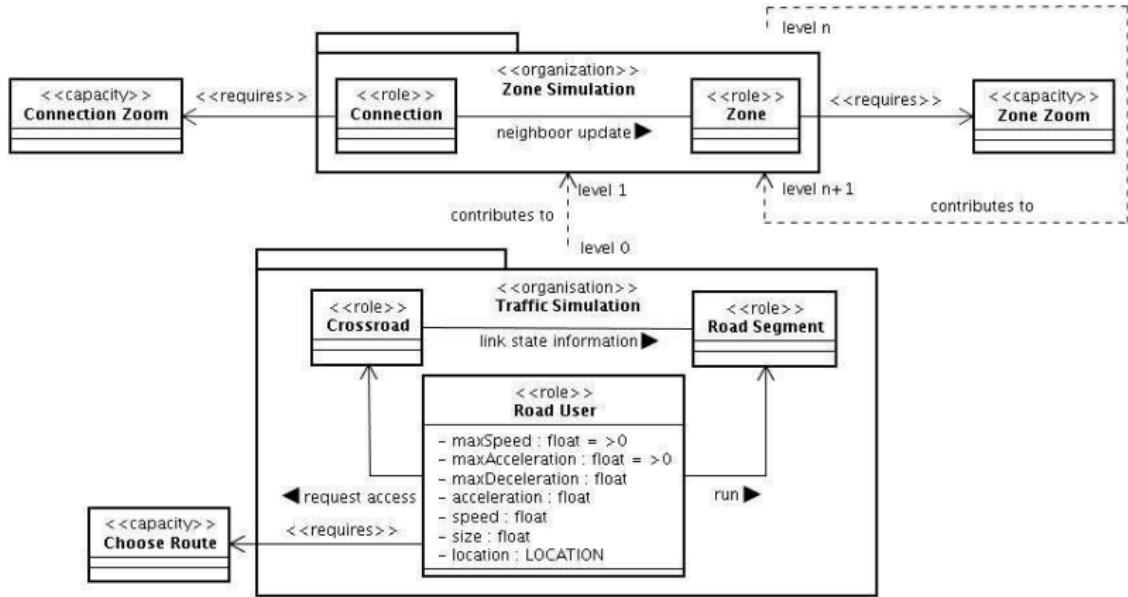












## 1 Complex Systems

## 2 Organizational Modeling

## 3 Holons and Holarchies

- Hierarchical Multiagent Systems
- Holons
- Example: Virtual Enterprise
- Example 1: Simulation of an Industrial Plant
- Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation

## 1 Complex Systems

## 2 Organizational Modeling

## 3 Holons and Holarchies

### ■ Hierarchical Multiagent Systems

- Holons
- Example: Virtual Enterprise
- Example 1: Simulation of an Industrial Plant
- Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation

Agents are not necessarily atomic.

The idea of “Agent composed of agents” is recurrent in the scientific community:

- Collective Agents [Ferber, 1999]
- Meta-Agents [Holland, 1995b]
- Recursive agents [Correa e Silva Fernandes, 2001]
- Agentified-Groups [Odell, 2005]
- **Holon and holarchies**  
[Rodriguez, 2005, Gaud, 2007, Galland, 2013a]

## Two different kinds of hierarchies

Hierarchical MAS are increasingly used.

- **Hierarchy of Control:** a relationship of authority between the agents.
- **Hierarchy of Composition:** Agents are composed of agents.

- Analysis, Modeling and Simulation of **Complex Systems**
- Representing multiple levels of abstraction
- Large Scale System Engineering

## 1 Complex Systems

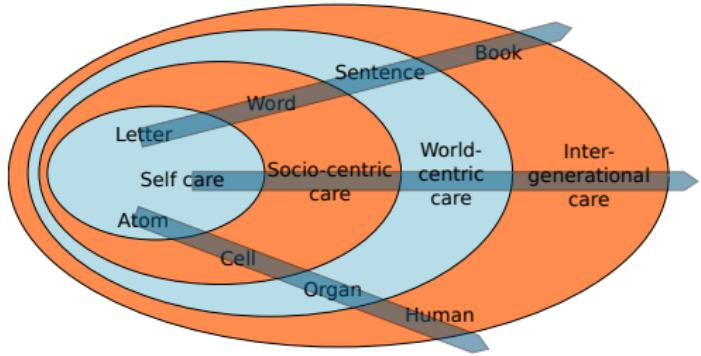
## 2 Organizational Modeling

## 3 Holons and Holarchies

- Hierarchical Multiagent Systems
- **Holons**
- Example: Virtual Enterprise
- Example 1: Simulation of an Industrial Plant
- Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation

[Koestler, 1967]

A holon is a self-similar structure, composed of holons as sub-structures.

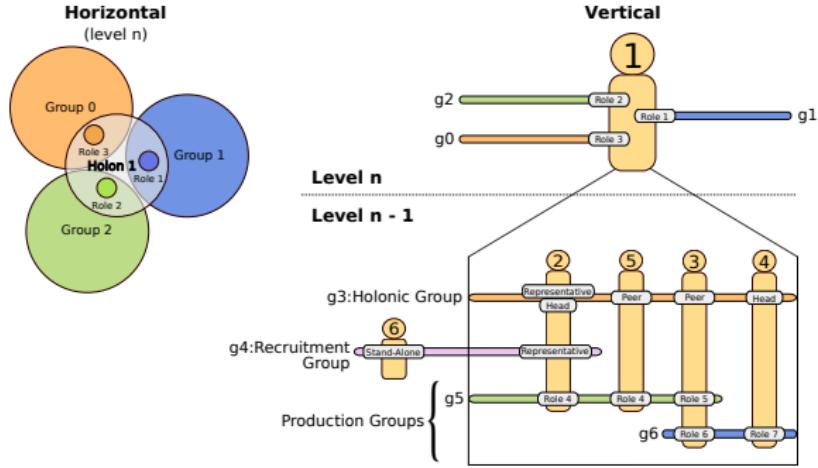


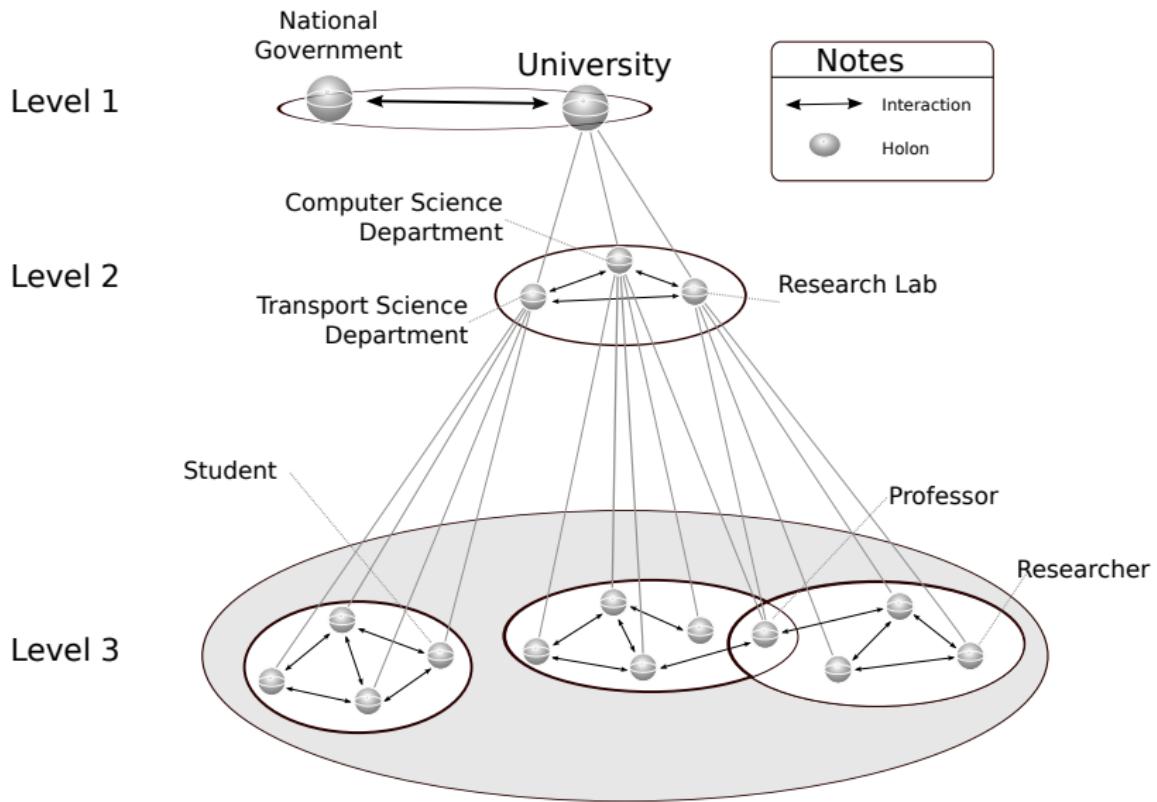
The concept of holon allows us to:

- to view a collection of interacting entities as a single higher-level holon.
  
- to decompose a entity seen as atomic at a certain level as a set of interacting entities at a lower level.

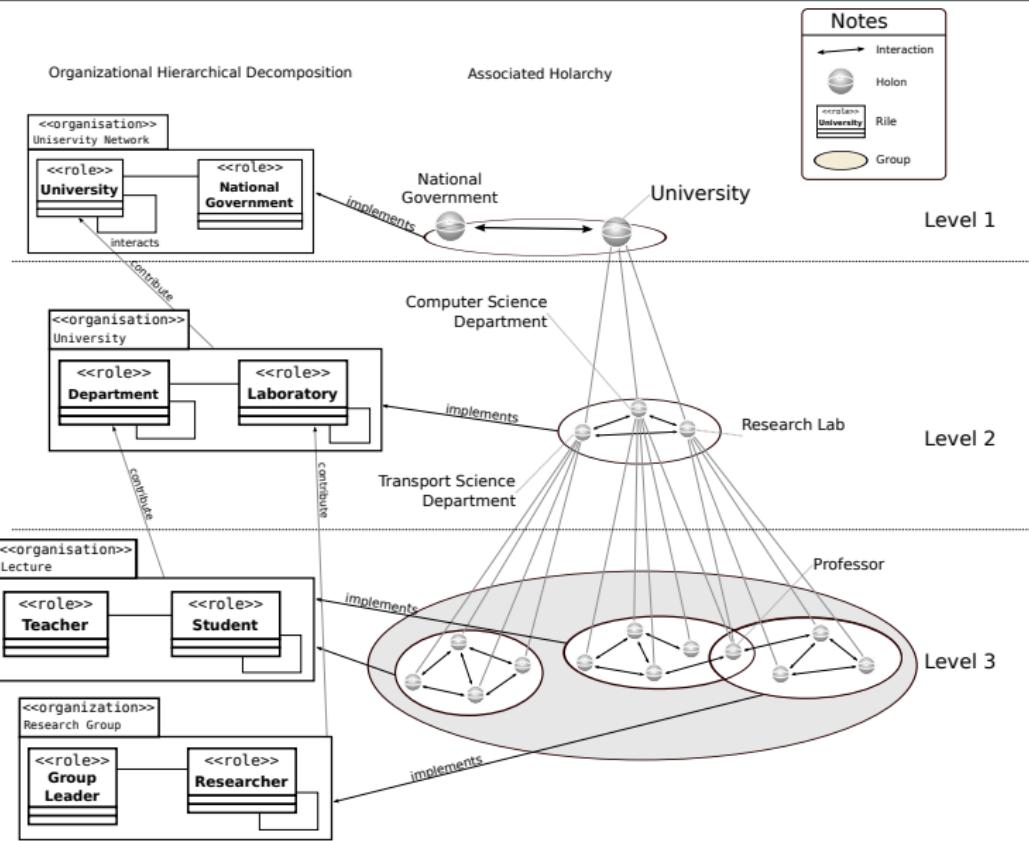
[Koestler, 1967]

The hierarchical structure composed of holons is called a **Holarchy**





# Mapping Organizations to an Holarchy



- Defined by A. Koestler to explain natural and social phenomena. [Koestler, 1967]
- Used in philosophy e.g. [Wilber, 1995, Edwards, 2003]
- Introduced in the Manufacturing System in the late 80's. [Kriz, 1995]
- Introduced in the MAS in late 90's. [Gerber, 1999]

- Study of natural and social phenomena:  
[Koestler, 1967, Wilber, 1995]
- Manufacturing System: ANEMONA [Giret, 2005], PROSA  
[Brussel, 1998], [Kriz, 1995, Wyns, 1999, Maturana, 1999]
- Transport Systems: [Bürckert, 1998]
- Robot Soccer: [Candea, 2000, Barbat, 2001]
- Collaborative Work: SOHTCO [Adam, 2000]
- Health: [Ulieru, 2002]
- MAS: [Gerber, 1999, Fischer, 2003, Rodriguez, 2005,  
Gaud, 2007, Galland, 2013a]

- Well adapted to the analysis and modeling of CS.
- Capable of reproducing the characteristics of CS.
- Large number of application domains.
- Dynamic and adaptive granularity/decomposition.
- Different levels of abstraction.

- What does it mean that an entity to be an Agent and a group of Agents at the time?
- How the information is transformed and transferred between abstraction levels ?
- Who makes the decisions? and how?
- How to model such systems?
- How to integrate/welcome new members within a super-holon? Is it even possible?

- What does it mean that an entity to be an Agent and a group of Agents at the time?
- How the information is transformed and transferred between abstraction levels ?
- Who makes the decisions? and how?
- How to model such systems?
- How to integrate/welcome new members within a super-holon? Is it even possible?

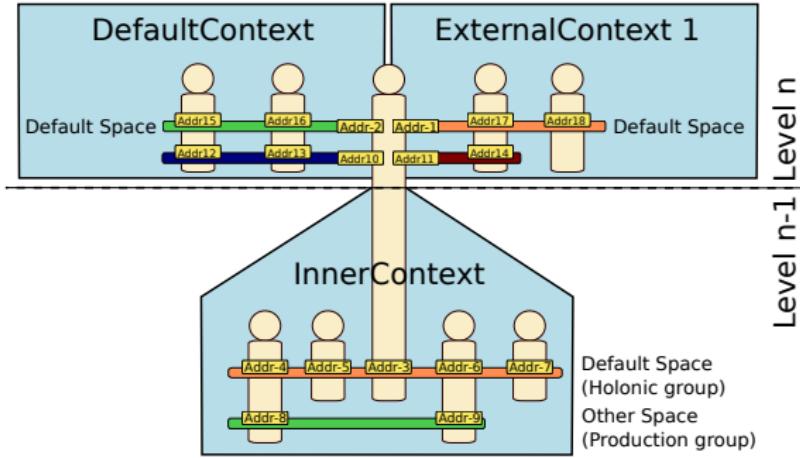
- What does it mean that an entity to be an Agent and a group of Agents at the time?
- How the information is transformed and transferred between abstraction levels ?
- Who makes the decisions? and how?
- How to model such systems?
- How to integrate/welcome new members within a super-holon? Is it even possible?

- What does it mean that an entity to be an Agent and a group of Agents at the time?
- How the information is transformed and transferred between abstraction levels ?
- Who makes the decisions? and how?
- How to model such systems?
- How to integrate/welcome new members within a super-holon? Is it even possible?

- What does it mean that an entity to be an Agent and a group of Agents at the time?
- How the information is transformed and transferred between abstraction levels ?
- Who makes the decisions? and how?
- How to model such systems?
- How to integrate/welcome new members within a super-holon? Is it even possible?

## Holonic Perspective of an Agent

- Every agents are holons: an agent may be composed of agents.
- Composed agent and composing agents are living in the same context: the **inner context**.

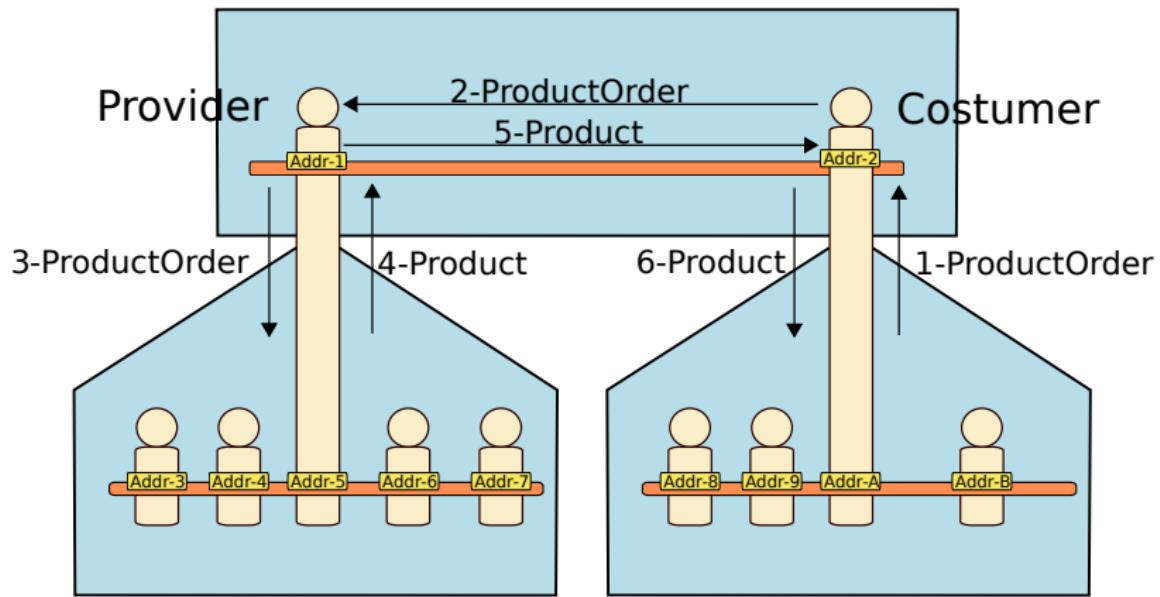


## 1 Complex Systems

## 2 Organizational Modeling

## 3 Holons and Holarchies

- Hierarchical Multiagent Systems
- Holons
- **Example: Virtual Enterprise**
  - Example 1: Simulation of an Industrial Plant
  - Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation



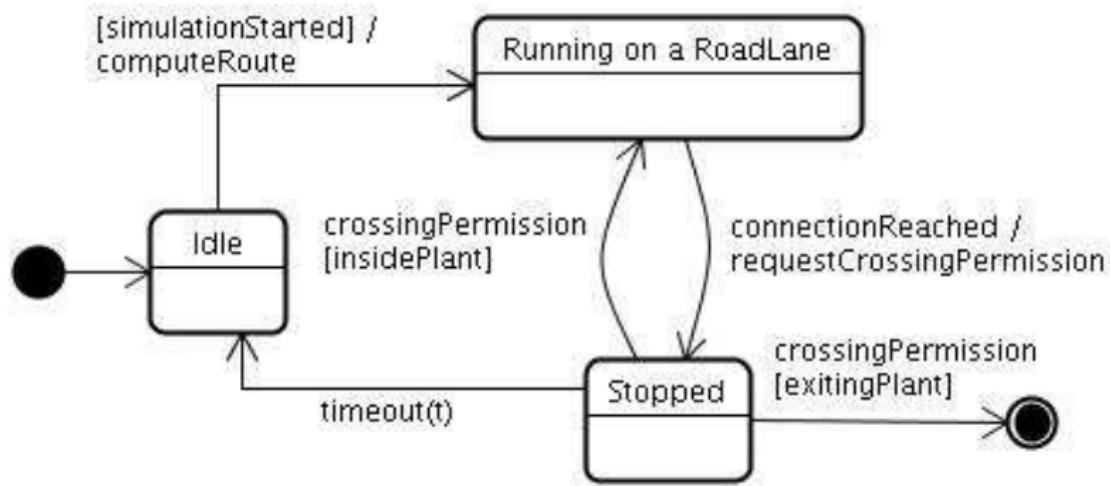
```
agent Provider {  
  
event ProductOrder {  
    val quantity : int  
    val id : String  
    new(q : int, i : String) {  
        this.quantity = q  
        this.id = i  
    }  
}  
  
event Product {  
    val quantity : int  
    val id : String  
    new(q : int, i : String) {  
        this.quantity = q  
        this.id = i  
    }  
}  
  
    var products : Map<String,Address>  
  
    on ProductOrder {  
        products.put( occurrence.id,  
                      occurrence.source)  
        wake( occurrence,  
              Scopes.notAddresses(innerSpace  
                               .address) )  
    }  
  
    on Product {  
        var adr = products.remove(  
                                  occurrence.id )  
        if (a !=== null) {  
            emit( occurrence,  
                  Scopes.addresses(adr) )  
        }  
    }  
}
```

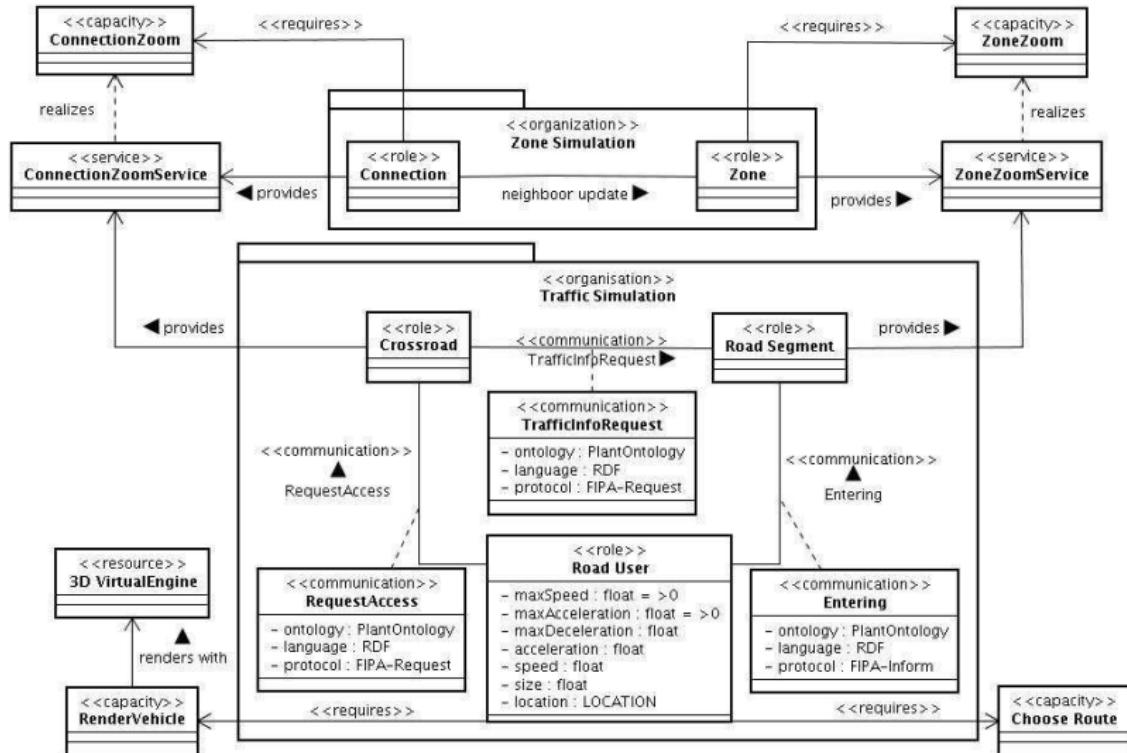
## 1 Complex Systems

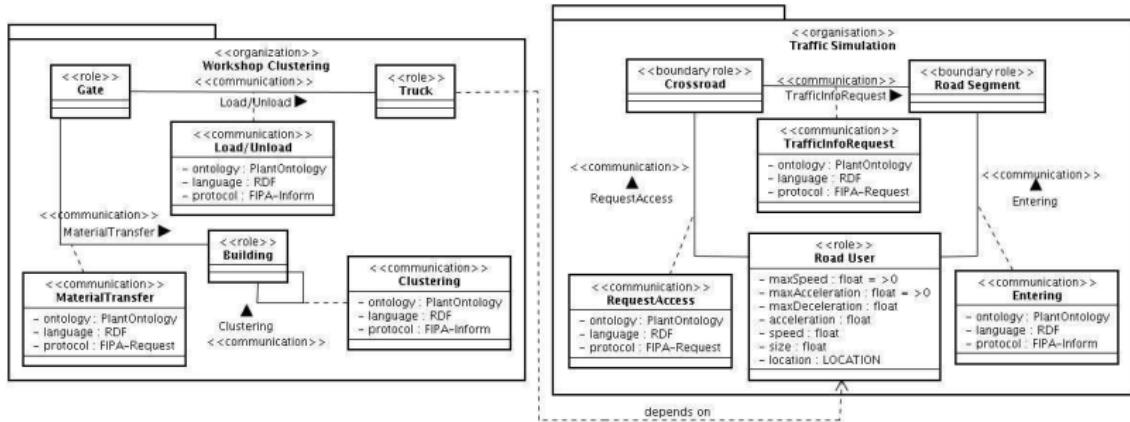
## 2 Organizational Modeling

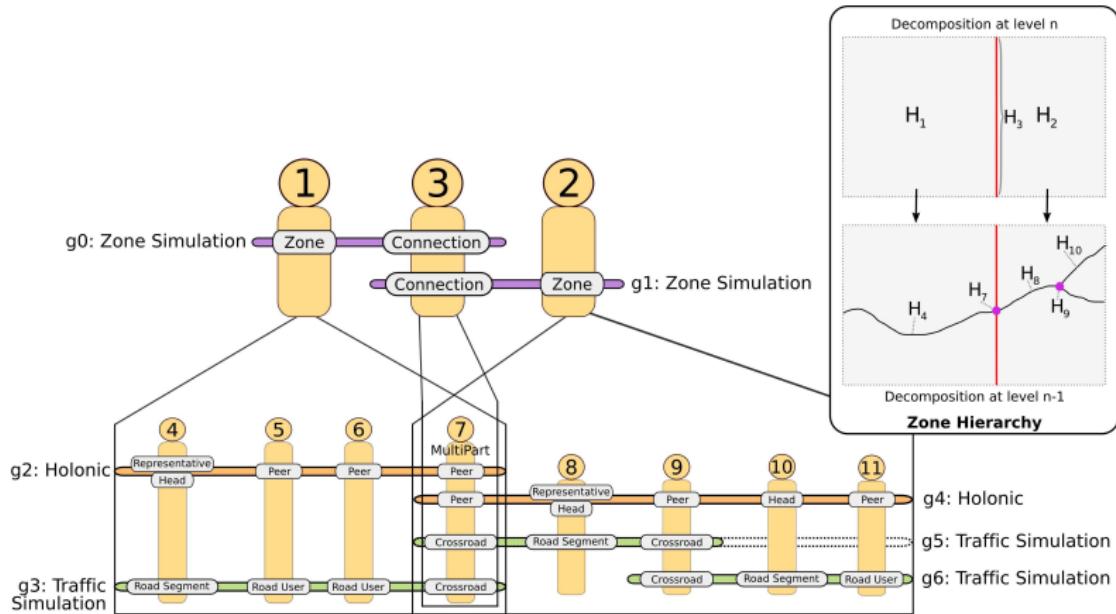
## 3 Holons and Holarchies

- Hierarchical Multiagent Systems
- Holons
- Example: Virtual Enterprise
- **Example 1: Simulation of an Industrial Plant**
- Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation









## 1 Complex Systems

## 2 Organizational Modeling

## 3 Holons and Holarchies

- Hierarchical Multiagent Systems
- Holons
- Example: Virtual Enterprise
- Example 1: Simulation of an Industrial Plant
- Example 2: Holonic Model of a Virtual 3D Indoor Environment for Crowd Simulation

## Modeling and simulation of individuals in a large-scale virtual world

Reproduction of the dynamics of individuals and groups of individuals evolving in a large-scale virtual world (building, street...).

### Application Domains

- Development of urban sites, security study, flow analysis...
- Interactive staff training, serious games.
- Video games, virtual animation...



- 1 Generating a population of virtual individuals.
- 2 Reproduction of the movements of the individuals.
- 3 Generation of individual and collective behaviors.
- 4 Modeling of a virtual environment.
- 5 Interaction with the virtual population.
- 6 Realistic displaying of the population and the univers.

- 1 Generating a population of virtual individuals.
- 2 Reproduction of the movements of the individuals.
- 3 Generation of individual and collective behaviors.
- 4 **Modeling of a virtual environment.**
- 5 Interaction with the virtual population.
- 6 Realistic displaying of the population and the univers.

The system composed by the population and the environment is a complex system [Thalmann, 2007]

## Goals

Provide the tools for modeling and simulating a large scale virtual world.

## Constraints

- Modeling and simulation relations [Zeigler, 2000].
- Support the scalability of the models.
- Minimize the simulator latency perceived by the immersed user (serious game).

## Adopted approach

- **Modeling:** Adopt an organizational and holonic modeling approach.
- **Simulation:** Dynamically adapt the level of the simulated behaviors, and of the environment, while trying to stay as accurate and as close as possible to the original model.

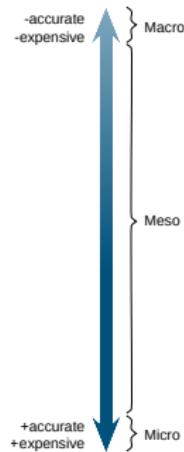
## Problems

Modulate the complexity of a simulation according to specific constraints (population density, available computing resources...)

## Hypothesis

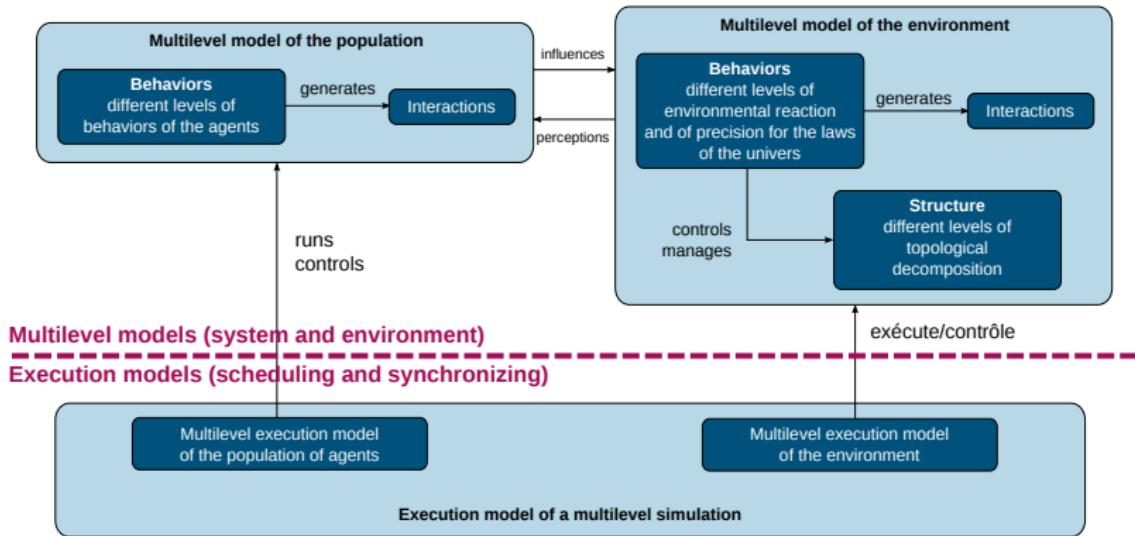
There is a correlation between the levels of abstraction in a simulation model and:

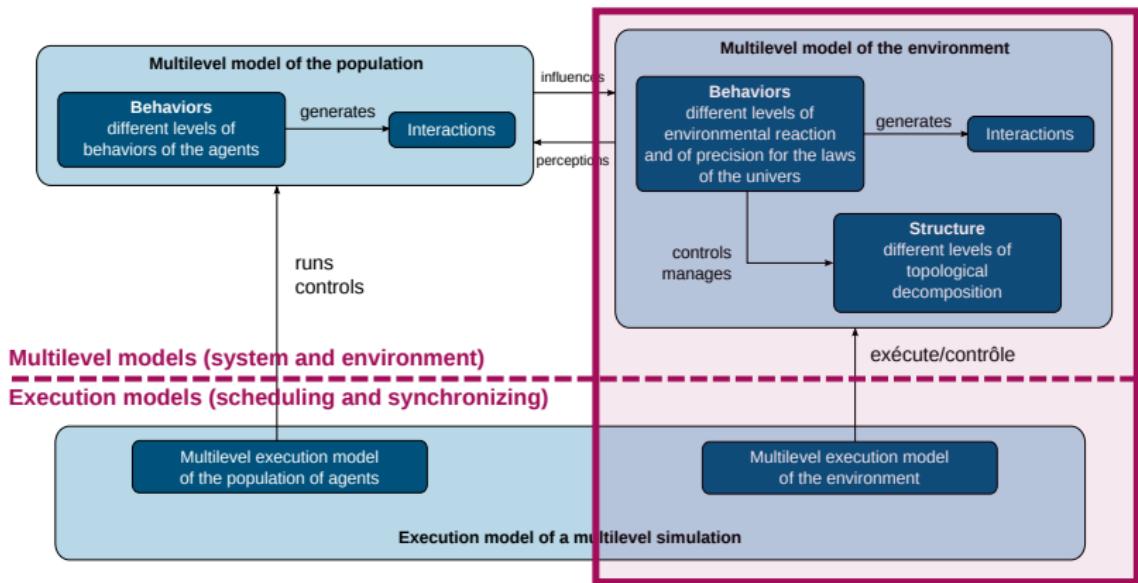
- the accuracy of the results;
- the resources needed for running.



## Multilevel simulation [Gaud, 2007]

- Integrates different levels of abstraction (at least two).
- Allows the coexistence of different levels within the same execution.
- Provides dynamic transitions between them.



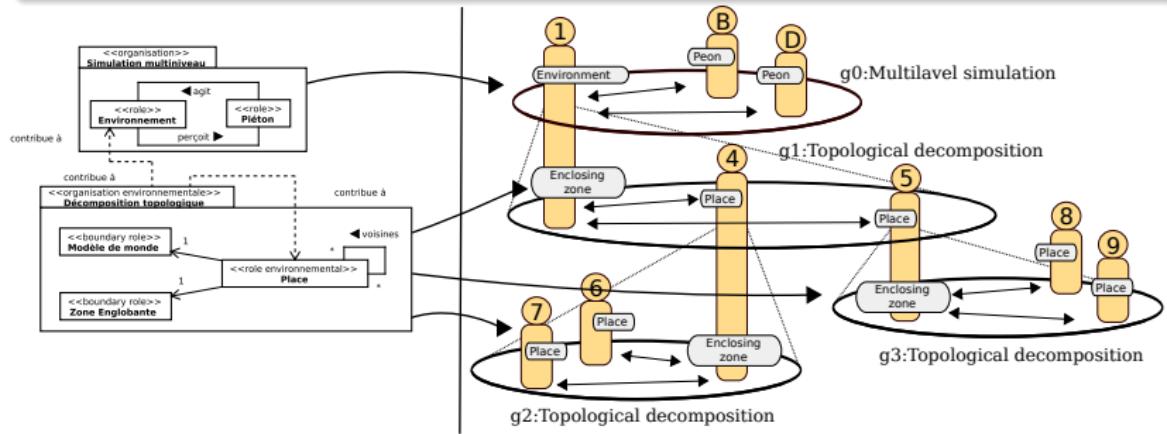


## Structure of the holarchy

The structure of the holarchy corresponds to the natural topological decomposition of the environment.

## Dynamics of the holarchy

Each holon decides to execute the missions of the environment for its level of decomposition, or to delegate to its member holons.

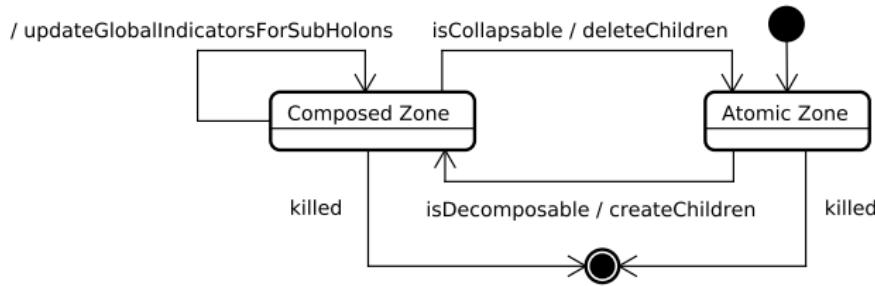


### Case 1

If the agent manages an atomic place, must it decompose this place and create sub-agents?

### Case 2

If the agent manages a decomposed place, must it combine the sub-places, and destroy the sub-agents managing these sub-places?



### Goals

- Choose the best level of abstraction for the best compromise between the simulation results and the performance of the simulator.

### Problems

- Transitions between levels of simulation, and consistency of the simulation.
- Ensure the transition between two levels of abstraction implies that the models used by each of them are consistent.
- Wider problem in MAS: Evaluating the accuracy or effectiveness of a system with respect to the task it has to perform and to the local mechanisms involved in this task.

### Proposal

Develop indicators that attempt to measure this effectiveness.

Three types of measures inspired by various classical energies in Physics

- Kinetic energy  $E_d$ : related to the dynamics of a holon (speed...)
- Goal potential energy  $E_o$ : linked to the objective of the holon (position of the goal...)
- Constraint potential energy  $E_c$ : for the elements hindering the progress of the holon (obstacles...)

Global energy of a holon  $k$

$$E_k = E_d + E_o + E_c$$

Similarity between two adjacent levels

$$s_{(i,j)} = E_i - E_k$$

Other indicators

Functions of Gibbs, of Z partition...

- 1 Top-down indicator that provides the amount of available resources .

$$R_a = (R_z - k_z) \times \frac{M_a}{\sum_{b \in D_z} M_b} \quad \forall a \in D_z$$

- $R_z$ : amount of resources given by the super-holon of  $z$ .
- $k_z$ : estimation of the resource consumption by  $z$ .
- $M_\alpha$ : mass/importance of the zone managed by  $\alpha$  in the simulation.
- $D_z$ : set of the subholons of  $z$ .

- 2 Mass of the zone  $z$ :

$$M_z = \alpha_z \cdot w_z + \sum_{a \in D_z} \alpha_a \cdot M_a + \sum_{e \in O_z} \alpha_e \cdot w_e$$

- $O_z$ : the objects located in the zone managed by  $z$ .
- $w_\alpha$ : the weight of  $\alpha$ .

$$\left[ \left( \exists a \in D_z, |E_z - E_a| > \epsilon \right) \vee \left( \forall R, R_z \geq \sum_{p \in D_z} g_R(p) + k_z \right) \right] \wedge \\ (\max_z < i \vee \min_z > i_z) \wedge \\ (O_z \neq \emptyset)$$

- $D_z$ : set of the subholons of  $z$ .
- $R_z$ : a resource indicator given by the super holon of  $z$ .
- $g_R(p)$ : estimate the amount of resources  $R$  for executing  $p$ .
- $k_z$ : estimation of the resource consumption by  $z$ .
- $O_z$ : the objects located in the zone managed by  $z$ .
- $\min_z, i_z, \max_z$ : min, current, max level of  $z$  in the holarchy.

$$\left( \forall a \in D_z, |E_z - E_a| \leq \epsilon \right) \wedge \left( \forall R, R_z < \sum_{p \in D_z} g_R(p) + k_z \right) \wedge \min_z < i_z$$

- $D_z$ : set of the subholons of  $z$ .
- $R_z$ : a resource indicator given by the super holon of  $z$ .
- $g_R(p)$ : estimate the amount of resources  $R$  for executing  $p$ .
- $k_z$ : estimation of the resource consumption by  $z$ .
- $\min_z, i_z$ : min, current level of  $z$  in the holarchy.

## Passenger Behavior

- **Activity-based model:** registration, boarding [Demange, 2012]
- **Force-based collision avoidance:** model based on shifting forces [Buisson, 2013]

## Environment Structure

- Two hall connected by security inspection posts.
- Two registration desks in the first hall, and two boarding gates in the second hall.
- Each hall is decomposed according to the 4D-tree heuristic.



- Kinetic energy: density of population

$$E_d = \frac{\alpha_c |O_z|}{\text{area of } z}$$

- Goal potential energy: distance for moving the pedestrians, with the set of influences  $I_z$

$$E_o = \alpha_{po} \sum_{i \in I_z} |i|$$

- Constraint potential energy: influences that is not applied due to conflicts, with  $m : I_z \mapsto R_z$  the influence-reaction mapping

$$E_c = \begin{cases} \alpha_{pc} \sum_{i \in I_z} (|i| - |m(i)|) & \text{if } O_z \neq \emptyset \\ 0 & \text{else} \end{cases}$$

- **Kinetic energy:** surfaces covered by the perception fields: added area  $s^\oplus$ , removed area  $s^\ominus$ , retained area  $s^\odot$

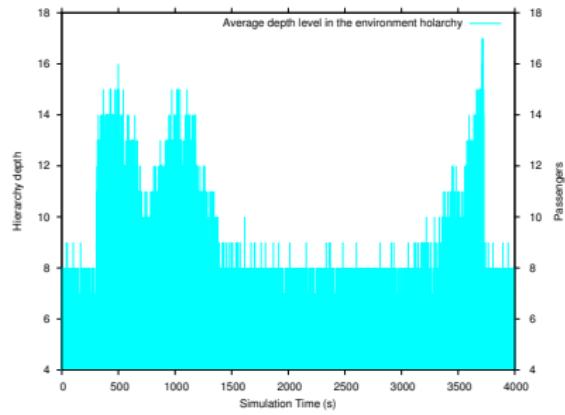
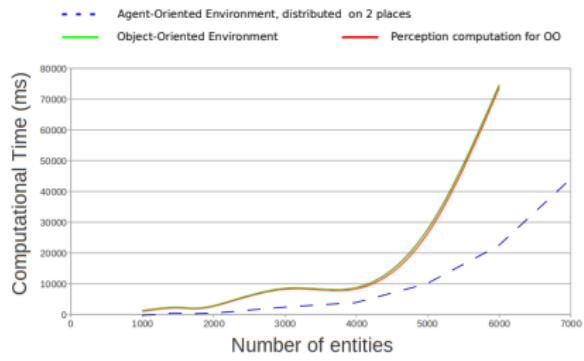
$$E_d = \begin{cases} \frac{\alpha_c |s^\ominus| + \beta_c |s^\oplus|}{|s^\odot|} & \text{si } s^\odot \neq \emptyset \\ \alpha_c |s^\ominus| + \beta_c |s^\oplus| & \text{sinon} \end{cases}$$

- **Goal potential energy:** set of perceived objects  $p$

$$E_o = \begin{cases} \frac{\alpha_{po} |p^\ominus| + \beta_{po} |p^\oplus|}{|p^\odot|} & \text{if } p^\odot \neq \emptyset \\ \alpha_{po} |p^\ominus| + \beta_{po} |p^\oplus| & \text{else} \end{cases}$$

- **Constraint potential energy:** set of perception filters  $f$  (based on semantic)

$$E_c = \begin{cases} \frac{\alpha_{pc}.f^\ominus + \beta_{pc}.f^\oplus}{f^\odot} & \text{si } f^\odot \neq \emptyset \\ \alpha_{pc}.f^\ominus + \beta_{pc}.f^\oplus & \text{sinon} \end{cases}$$



- Complex systems are everywhere.
- Multiagent systems is well-suited for modeling complex systems.
- Organizational modeling approach enables to break down the modeling complexity by focusing on the organizations of the system.
- Holons, and holarchies, enable to model the intrinsically hierarchical nature of a complex system.
- Methodology for designing and implementing a Holonic Multiagent System: ASPECS (<http://www.aspecs.org>).



**Thank you for your attention...**



# **Design and Implementation of an Agent Platform**

**prof.dr.habil. Stéphane GALLAND**

- 1** Presentation of the basics of an agent platform.
  - Standard features.
  - Standard architecture.
- 2** Description the key points for designing an agent platform.
- 3** Overview of the major agent platforms over the World.
- 4** Examples:
  - The TinyMAS platform.
  - The Janus platform.

- 1 Introduction**
- 2 FIPA Standard: Platform Features**
- 3 FIPA Standard: Platform Architecture**
- 4 Designing a Platform**
- 5 A Brief Listing of Existing Platforms**
- 6 Agent Platform From Scratch: the example of the TinyMAS platform**
- 7 Service-based Platform: the example of the Janus platform**

## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

## Two main standards in MAS platforms

- FIPA<sup>a</sup>

Reference Implementation : JADE [Bellifemine, 2001]

- MAF<sup>b</sup>

Reference Implementation :

GrassHopper [Bäumer, 1999, Bäumer, 2000]

<sup>a</sup>FIPA, Foundation for Intelligent Physical Agents : <http://www.fipa.org>

<sup>b</sup>MAF, Mobile Agent Facility : [http://www.omg.org/technology/documents/formal/mobile\\_agent\\_facility.htm](http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm)

## Classification

- **Language-oriented:**
  - Declarative languages: CLAIM, DALI ou ReSpecT.
  - Imperative languages: Jack.
  - Hybrid : Jason ou IMPACT.
- **Mobile or Web Agents:** Jade, AgentBuilder, GrassHopper, Aglet, Able.
- **Simulation:** NetLogo, GAMA, MatSim, Swarm, Cormas, Mobicity.
- **General platforms:** MAST, Geamas, Madkit, Magique, Janus (SARL).

Many platforms have been created, few have survived

Name	Domain	Hierar. <sup>a</sup>	Simu. <sup>b</sup>	C.Phys. <sup>c</sup>	Lang.	Beginners <sup>d</sup>	Free
GAMA	Spatial simulations		✓		GAML, Java	**[*]	✓
Jade	General		✓	✓	Java	*	✓
Jason	General		✓	✓	Agent-Speaks	*	✓
Madkit	General		✓		Java	**	✓
NetLogo	Social/natural sciences		✓		Logo	***	✓
Repast	Social/natural sciences		✓		Java, Python, .Net	**	
SARNL	General	✓	✓ <sup>e</sup>	✓	SARNL, Java, Xtend, Python	**[*]	✓

a Native support of hierarchies of agents.

b Could be used for agent-based simulation.

c Could be used for cyber-physical systems, or ambient systems.

d \*: experienced developers; \*\*: for Computer Science Students; \*\*\*: for others beginners.

e Ready-to-use Library: Jaak Simulation Library, <https://github.com/gallandarakhneorg/jaak>

## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

- A non-profit association of companies, founded in 1996.
- Promotes the success of emerging agent-based applications, services, and equipments to facilitate the end-to-end inter-working:
  - for heterogeneous and interacting agents and agent-based systems.
  - developed by different companies and organizations.
- The goal is pursued by **producing the specifications** of:
  - Agent Management and platform services.
  - Agent Communication Model and Language.
  - set of common Interaction Protocols.
- The FIPA's core message:
  - through a combination of speech acts, predicate logic and public ontologies, standard ways of interpreting communication between agents can be achieved that respect the intended meaning of the communication.
- Website: <http://www.fipa.org>



## Platform Features

- Features are gathering from existing platforms.
- They are grouped into several set of features:

Technological  
Features

Domain  
Features

Development  
Features

Analysis  
Features

Exploration  
Features

## Goal

Provide models and algorithms to create and execute a platform and a multiagent system.

Technological  
Features

Domain  
Features

Development  
Features

Analysis  
Features

Exploration  
Features

## Examples

- **Synchronization methods:** Provides a collection of algorithms and method to synchronize the agents, and the agents and the environment in discrete time or with events.
- **Agent Life Cycle:**
  - Respect the life cycle of the agents.
  - How to launch agents in a thread, applet, or scheduler.
  - A platform must provides at least a multi-thread or network support.

## Examples (cont.)

### ■ Intentional Errors:

- Set of errors fired by the developer of the system.
- Operational errors: errors in the lowest layers of the platform (operating system, network cards, etc.)
- Logical errors: errors in the model of the agent system (domain dependent).

## Goal

- Provide generic models for several application domains.
- Provide extension mechanisms for models.



## Examples

- Controlled or Not-Controlled Environment:
  - The environment is inside the simulator. It is observable, recordable, replicable, and repeatable.
  - The environment is outside the simulator: real world, etc.
- Organization Models:
  - Contract-Net Protocol.
  - University organization.
  - Urban systems.

## Goal

- Provide methods and tools for the development of multiagent systems.
- Agent-Oriented Software Engineering (AOSE).



## Examples

- **Agent Template:** abstract class or template of Agent, with generic features already implemented.
- **Message API:**
  - Creating and building messages.
  - Mechanisms to extend message contents.
  - Parsers for specific message content (KQML...)

## Examples (cont.)

### ■ Organizational abstractions:

- Provides API and implementation of an organizational model such as CRIo or AGR.
- Provides tools to dynamically create and manage organization of agents.

### ■ Multiple Society:

- Provides tools to create a society of agents which is observing an other society of agents.
- Provides tools to permit to a society of agents to control an other society of agents.

## Goal

- Enable observation of the system behavior.
- Analyze and validate the simulation outputs.



## Examples

- Probing Social Events:
  - Probing the interaction events, ie. among the agents.
  - Gathering the exchanged messages.
- Probing Behaviour Events:
  - Probing the events inside the agents.
  - Inspect the agent states.

## Goal

- Enable interactive exploration of emergent behaviors.
- Enable to inspect and change the internal states of the agents and the simulator.



## Examples

- **Event API:** Create new social or behaviour events.
- **Society of agents as Blackbox:** Tools to analyze and change the input and the output of a society of agents seen as a blackbox.
- **Introspection Mechanisms:** Allow the agents to analyze their societies.

## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

## Micro-kernel Architecture

- Features are grouped to create
  - a **micro-kernel**: core features of the platform.
  - **internal services**: application-independent features.
  - **external services**: application-dependent features.
- The micro-kernel provides the API to plug in/out internal and external services.
- All the interactions among the internal and external services must go through the micro-kernel.

Application Agents

## Agent Communication Language (ACL)

### Agent Shell (AS)

Agent Configuration

Task Management  
Conversation Management

### Agent Management System (AMS)

Directory Facilitator (DF)

### Agent Communication Channel (ACC)

Message Transport Service  
Message Transport Internal Protocols

**Agent Management System (AMS)**  
Directory Facilitator (DF)

**Agent Communication Channel (ACC)**  
Message Transport Service  
Message Transport Internal Protocols

## Missions of the ACC

- Provides the low-level features to route messages inside the same micro-kernel or on a network.
- Publishes its network address to be accessed by remote ACC.
- Use the Agent Management System (agent addresses) to route the messages.

## Related to the ACC

- **Message Transport Service (MTS)** is the service associated to the ACC.
- Internal Protocols for Message Transport : Protocols used to route the messages within the same micro-kernel.

## Missions of the AMS

- Manage the low-level resources used by the platform (memory, processor, harddisk, printer, etc.)
- Control the access to and the use of the micro-kernel components.
- Provides informations about the configuration and capabilities of the platform.
- Provides base functions to add agents in the platform.
- Provides a directory of the registered agents (the “white-pages”).

## Missions of the DF

- Gives a directory of all the services provided by the agents (the “yellow-pages”).
- Maintains a recent description for each agent.

**Agent Shell (AS)**  
Agent Configuration

Task Management  
Conversation Management

## Missions of the AS

- Provides canvas, templates and abstractions to implement agents
- Load a profile or a configuration for an agent during its initialization stage
- Create the MTS, and the Task and Conversation managers
- Provides policies to:
  - send messages,
  - get properties of agents,
  - save/reload the platform state,
  - stop agents.

## Missions of the TM

- Divide the behavior of the agent in work units: the tasks
- Parallelize tasks when possible

## Missions of the CM

- Conversation is a set of messages exchanged by tasks to fulfill a goal
- Provides the state of a conversation
- Group the messages for the same conversation before sending them through the MTS
- Ensure that the conversation protocol is not broken
- Provide an API to create new protocols

Application Agents  
**Agent Communication Language (ACL)**



## Missions of the ACL

- Based on speech act theory: messages are actions, or communicative acts
- Consists of a set of message types and the description of their pragmatics
- the effects of the messages on the mental state of the agents

Application Agents

## Agent Communication Language (ACL)

**Agent Shell (AS)**  
Agent Configuration

Task Management  
Conversation Management

**Agent Management System (AMS)**  
Directory Facilitator (DF)

**Agent Communication Channel (ACC)**  
Message Transport Service  
Message Transport Internal Protocols

## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

## 1 Analyze the application domain:

- To define the external services,
- To identify the features mandatory to implement each external service.

## 2 Classify the services into the micro-kernel or internal services.:

For the micro-kernel, services may be:

- fast to be executed,
- Hardware-dependent,
- frequently used.

## 3 Basically, resource management are put in the micro-kernel:

- Agent management and scheduling,
- Task management,
- Memory or hard disk management

## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

- Agent platform
- Agent-agent communications
- Multi-thread and execution policies
- Network support
- Agent observation toolkit
- BDI, Language Acts
- FIPA Std reference implementation
- Environment Model: none



<http://jade.tilab.com>

- Multi-agent platform
- Agent = turtle
- Turtle communication: direct and stigmergy
- Multi-thread and execution policies
- Observation toolkit
- **Environment Model:** Patches and Links.



<http://ccl.northwestern.edu/netlogo>

- Spatial simulation
- Agent-agent communications
- Multi-thread and execution policies
- Network support
- Agent observation toolkit
- Environment Model: Geographical data structures.

<https://github.com/gama-platform>

- Agent and organizational platform
- Agent-agent communications
- Multi-thread and execution policies
- Transparent network support with Hazelcast and ZeroMQ
- Agent observation toolkit
- Maven compliant
- BDI, Language Acts, Android
- **Environment Model:** Jaak extension (open source), JaSim, both based on Perception-Influence-Reaction Models



<http://www.janusproject.io>

- **Agentbuilder:** <http://www.agentbuilder.com>
- **Cougaar:** <http://www.cougaar.org>
- **Goal:** <http://mmi.tudelft.nl/trac/goal>
- **JaCaMo:** <http://jacamo.sourceforge.net> and its components: Jason, CArtAgO, Moise
- **Jack:** <http://www.agent-software.com/products/jack/>
- **Jadex:** <http://jadex-agents.informatik.uni-hamburg.de>
- **JIAC:** <http://www.jiac.de>
- **Madkit:** <http://www.madkit.org>
- **Mason (sugarscape):** <http://cs.gmu.edu/~eclab/projects/mason/>
- **MatSim:** <http://matsim.org>
- **Repast:** <http://repast.sourceforge.net>
- **Soar:** <http://sitemaker.umich.edu/soar/home> and  
<http://www.soartech.com>
- **TinyMAS:** <http://www.arakhne.org/tinymas/>
- **Whitestein LS:** <http://www.whitestein.com>

- 1 Introduction
- 2 FIPA Standard: Platform Features
- 3 FIPA Standard: Platform Architecture
- 4 Designing a Platform
- 5 A Brief Listing of Existing Platforms
- 6 Agent Platform From Scratch: the example of the TinyMAS platform
  - Agent Identification
  - Time Manager
  - Agent Repository
  - Service Repository
  - Message-based Communication
  - Agent Execution

- Designed for teaching basics of the MAS platform design to students.
- It enables to create simple agent-based simulation applications.
- It is not designed for “serious” applications.
- No more under development since all the developers are involved into the Janus project.

Official website:

<http://www.arakhne.org/tinymas/>

Source code:

<https://github.com/gallandarakhneorg/tinymas/>

## What it can do

- Easy to create reactive agents.
- Communication between agents by messages.
- Synchronous execution of the agents.
- Framework for the physic environment.

## What it cannot do

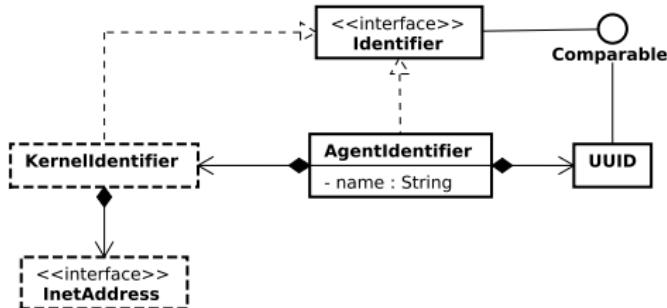
- Hard to create rational agents.
- Asynchronous execution of agents.
- Cannot be easily deployed on a computer network.

## Needs

Agents must be identified in a unique way for receiving the messages.

## How to make it unique?

- Agent identifiers are based on an UUID.
- Each identifier is linked to the identifier of the kernel.
- A name could be associated to the agent identifier, but it may be not unique.

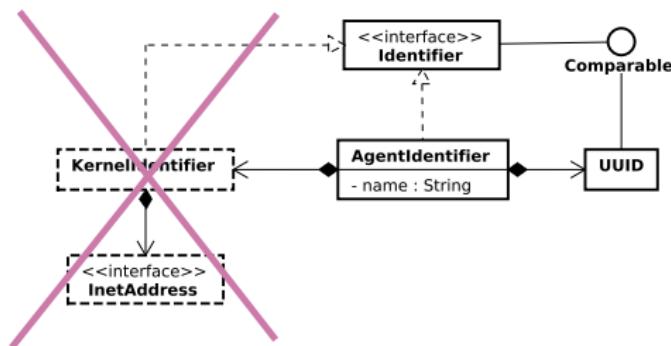


## Problem of the reference to the kernel identifier

If the agent identifiers are binded to the kernel identifiers, the agents are not able to migrate from one kernel to another.

## How to make it possible?

Assuming UUID is unique in all the computers over the World, use only the UUID.

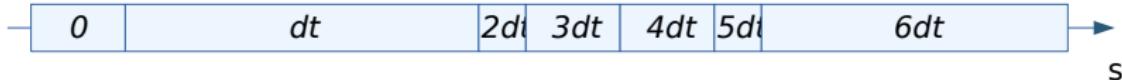


## Definition of Simulation [Banks, 1999]

The simulation is the **imitation in time** of the operations of a real or imaginary process. The simulation involves the generation of an artificial evolution of the system and the observation of these developments to make inferences about the operational characteristics of the real system represented.

How to model the time?  
How to manage the time evolution?

## Agents



## Micro-kernel



## Operating system

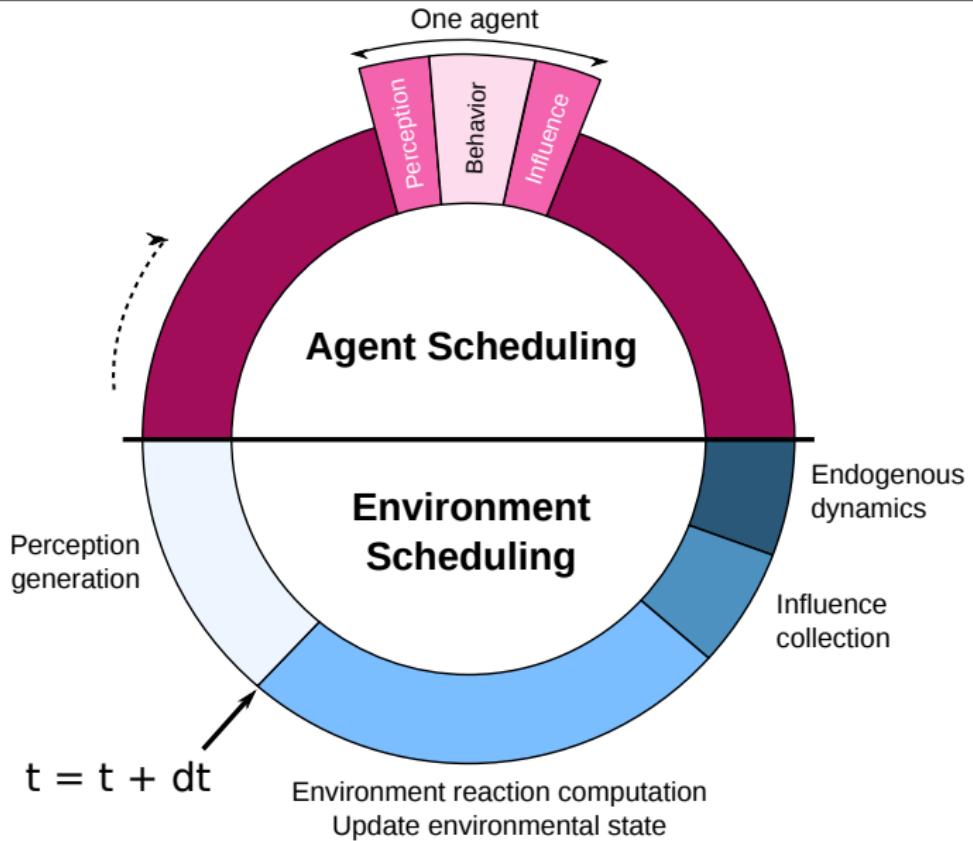


## Processor

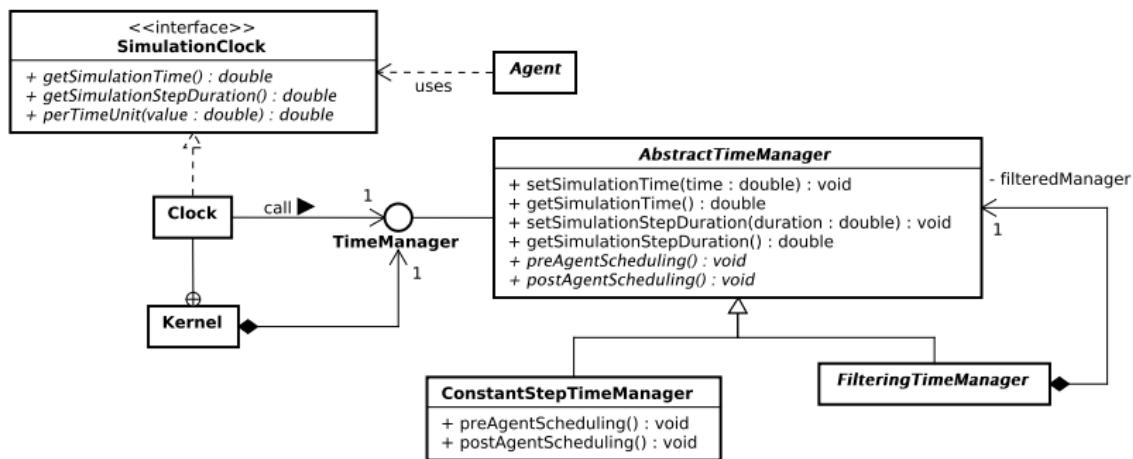


## Human being

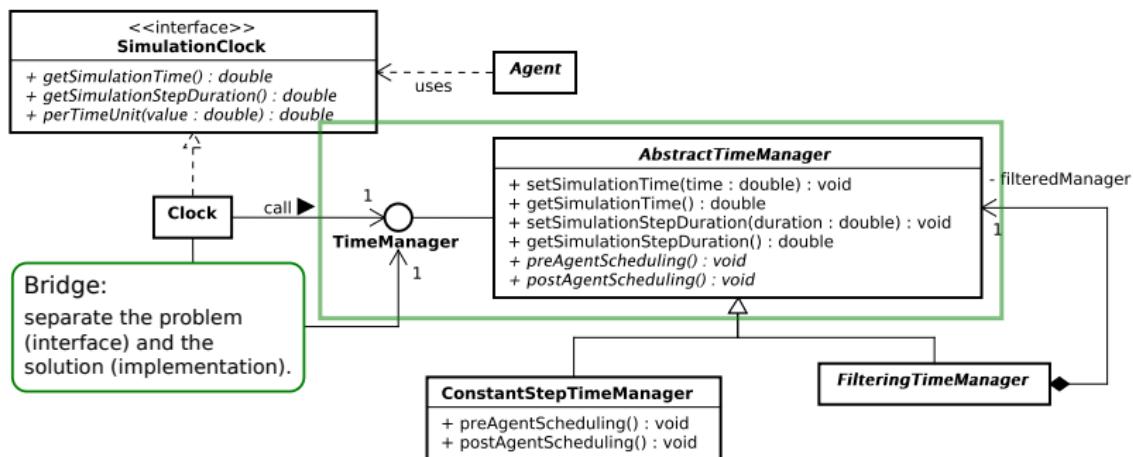




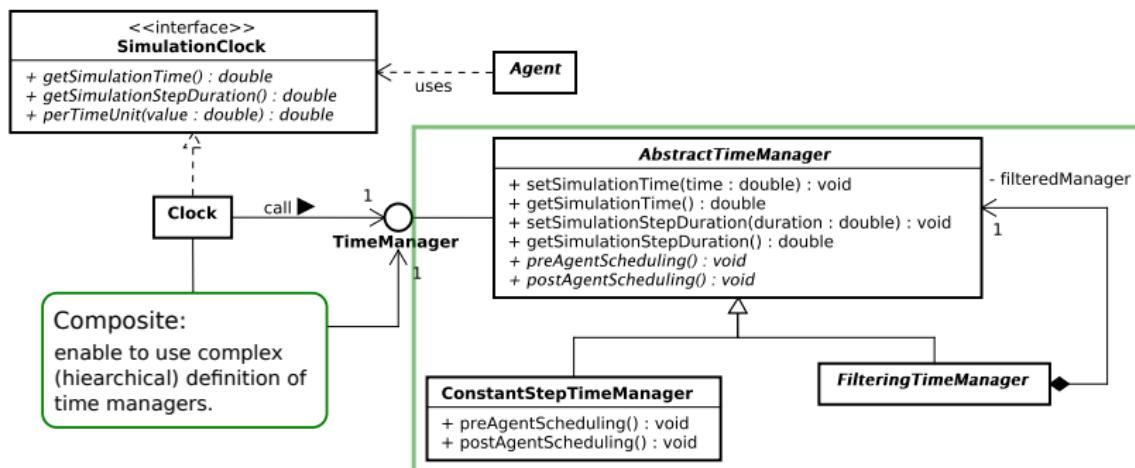
- Abstract implementation that provides basics implementation.
- **preAgentScheduling**: called at the beginning of a simulation step (before any agent execution) in a synchronous execution engine.
- **postAgentScheduling**: called at the end of a simulation step (after execution of all agents).
- **ConstantStepTimeManager**: increments time by the duration of one step, after all agents were executed.



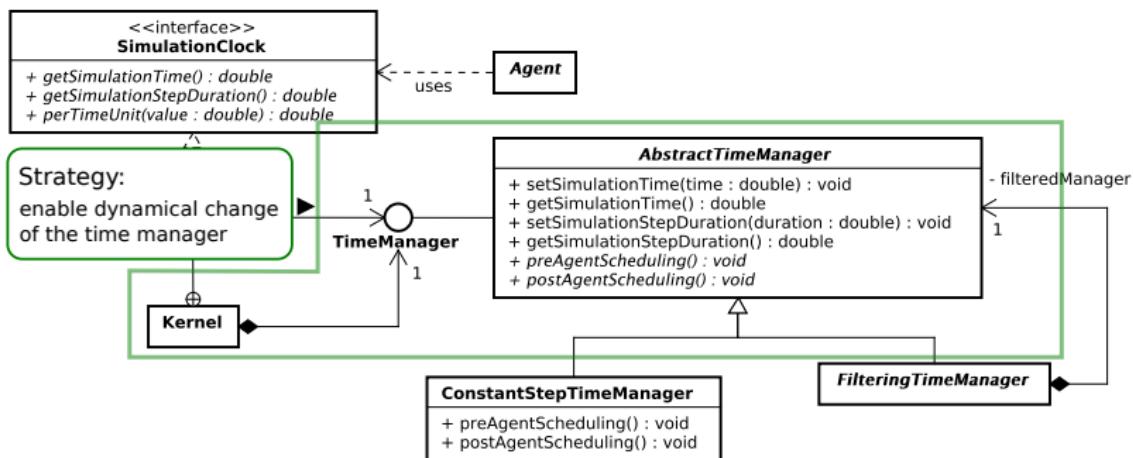
- Abstract implementation that provides basics implementation.
- **preAgentScheduling**: called at the beginning of a simulation step (before any agent execution) in a synchronous execution engine.
- **postAgentScheduling**: called at the end of a simulation step (after execution of all agents).
- **ConstantStepTimeManager**: increments time by the duration of one step, after all agents were executed.



- Abstract implementation that provides basics implementation.
- **preAgentScheduling**: called at the beginning of a simulation step (before any agent execution) in a synchronous execution engine.
- **postAgentScheduling**: called at the end of a simulation step (after execution of all agents).
- **ConstantStepTimeManager**: increments time by the duration of one step, after all agents were executed.

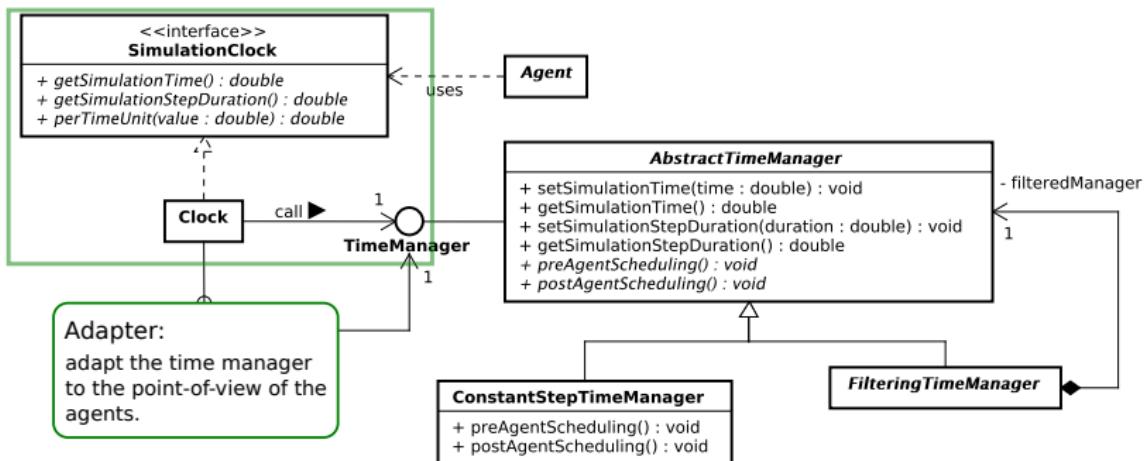


- Abstract implementation that provides basics implementation.
- **preAgentScheduling**: called at the beginning of a simulation step (before any agent execution) in a synchronous execution engine.
- **postAgentScheduling**: called at the end of a simulation step (after execution of all agents).
- **ConstantStepTimeManager**: increments time by the duration of one step, after all agents were executed.



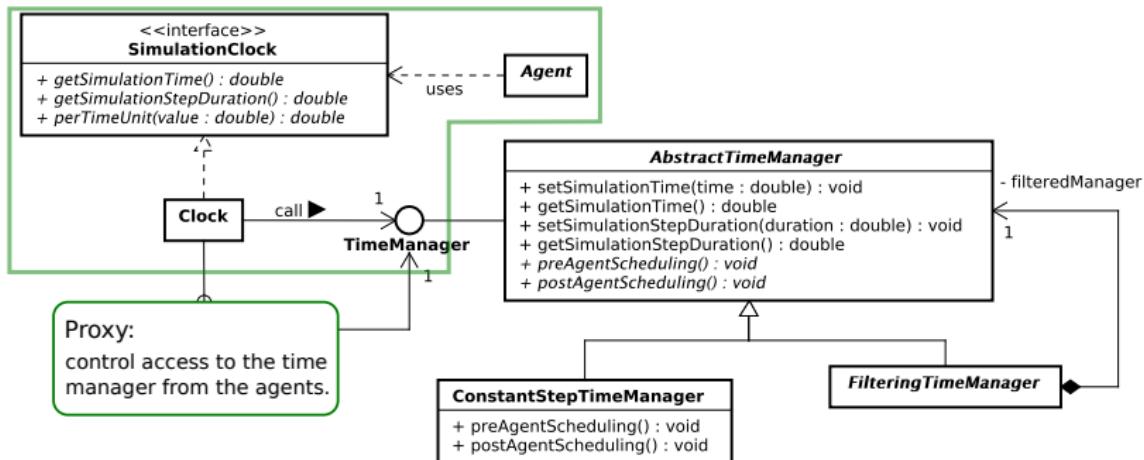
Agents must not access to the time manager directly.  
(for avoiding a call to the setters)

## Application of design patterns



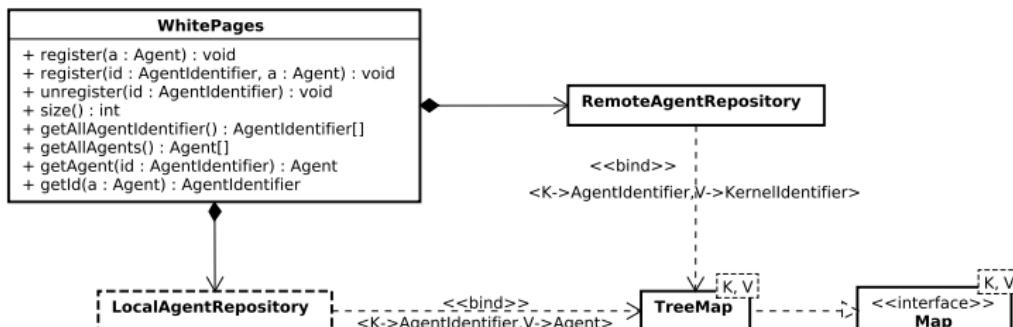
Agents must not access to the time manager directly.  
(for avoiding a call to the setters)

## Application of design patterns



## How to store the identifier-agent mapping?

- Create a map implementation with a specific class interface: the “white pages”.
- Properties:
  - Contains **all the agents** that are living on the current platform kernel.
  - Store information for retrieving the agents located on remote kernels.



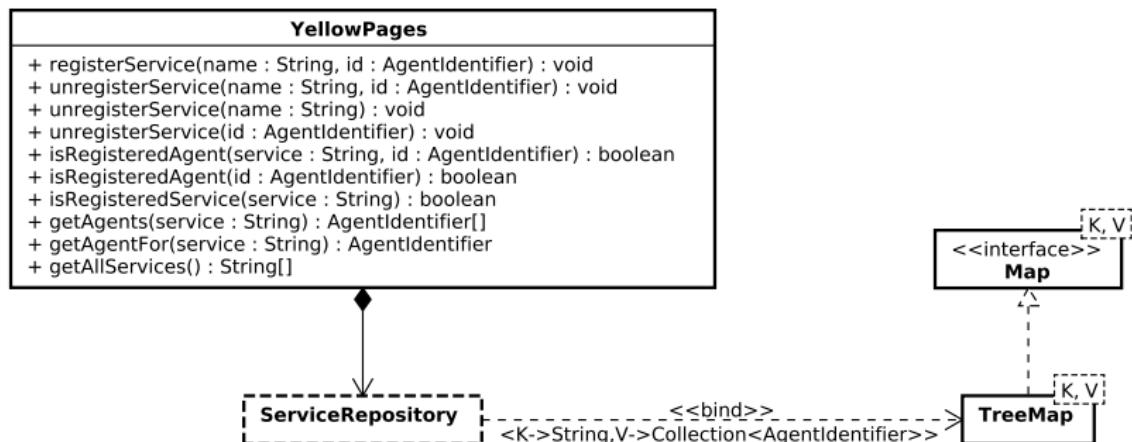
## Definition

A service is something that an agent can do, and another agent could query for using this service.

How to store the mapping between a service and an agent?

## How to store the service-agent mapping?

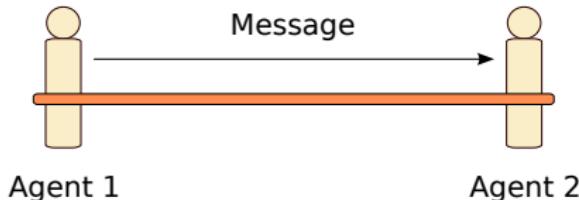
- Create a map implementation with a specific class interface: the “yellow pages”.
- **Values:** the names of the services.
- **Keys:** a collection of the identifiers of the agents that are able to provide the service.



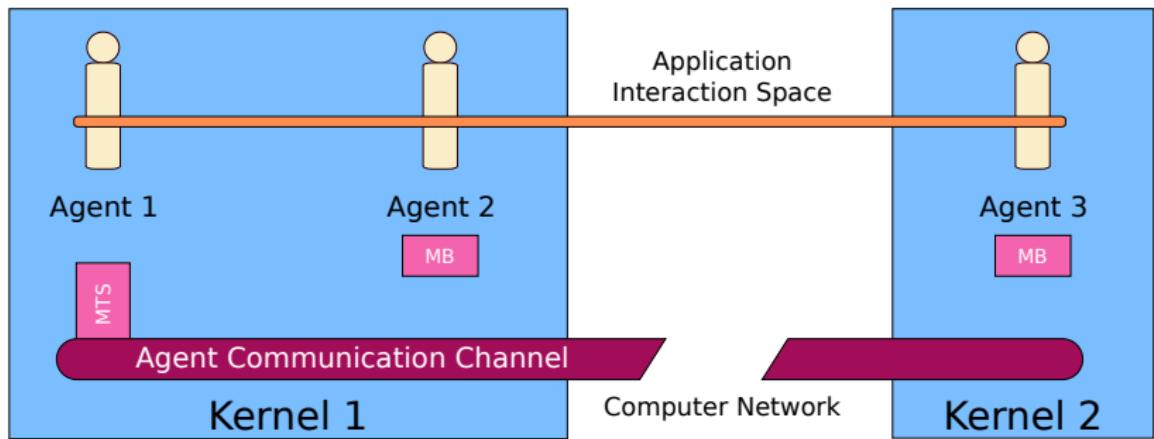
## Direct interaction

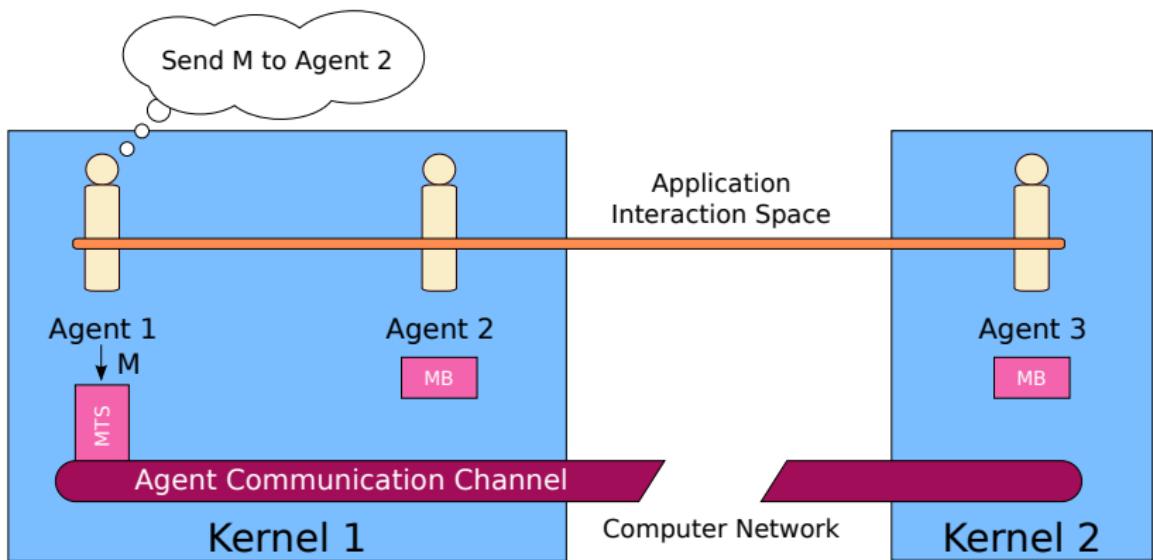
A direct interaction between agents permits to the agents exchanging a piece of information. A direction interaction may take different forms:

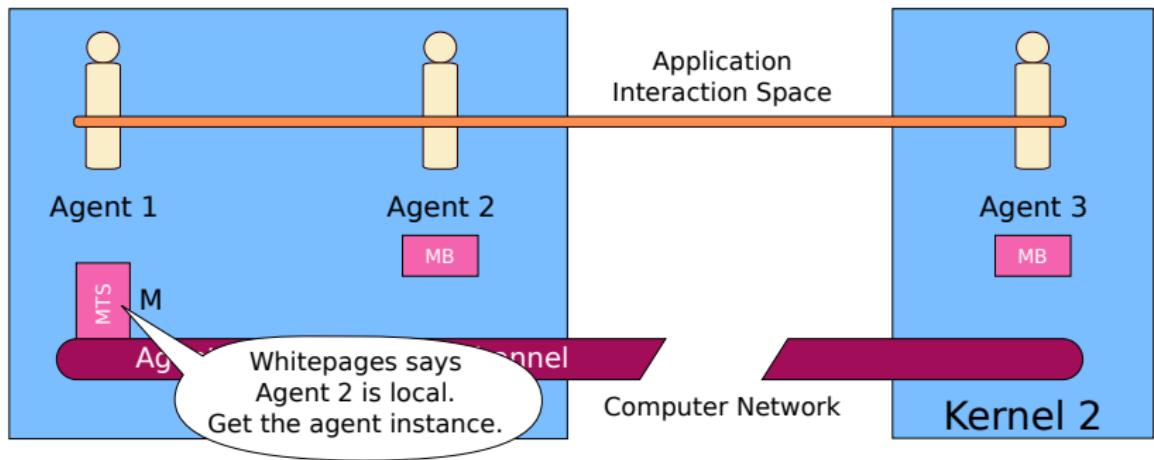
- **Message:** may contain complex information. The sender knows the receiver(s). The receivers know the emitter.
- **Event:** may contain complex information. The sender doesn't know the receiver(s). The receivers know the emitter.
- **Signal:** contains basics information (number, etc.) The sender doesn't know the receiver(s). The receivers know the emitter.

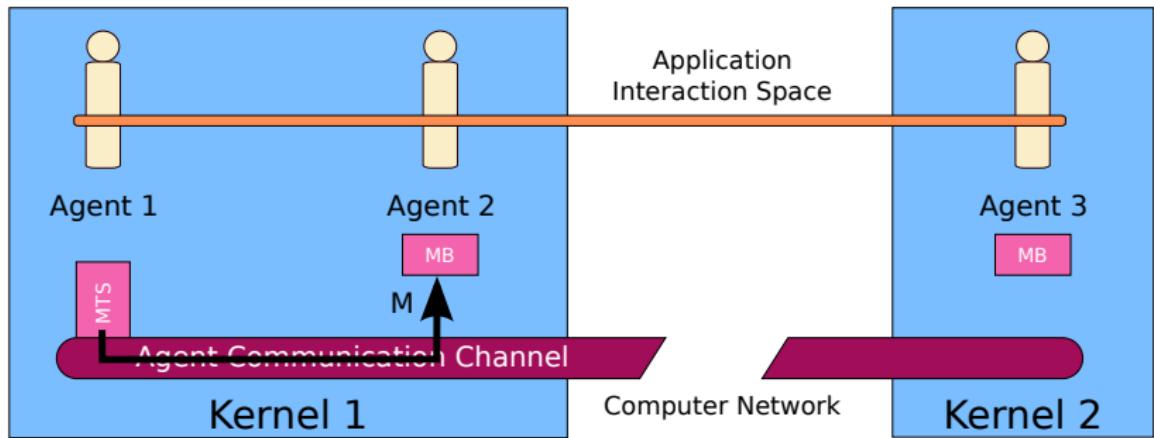


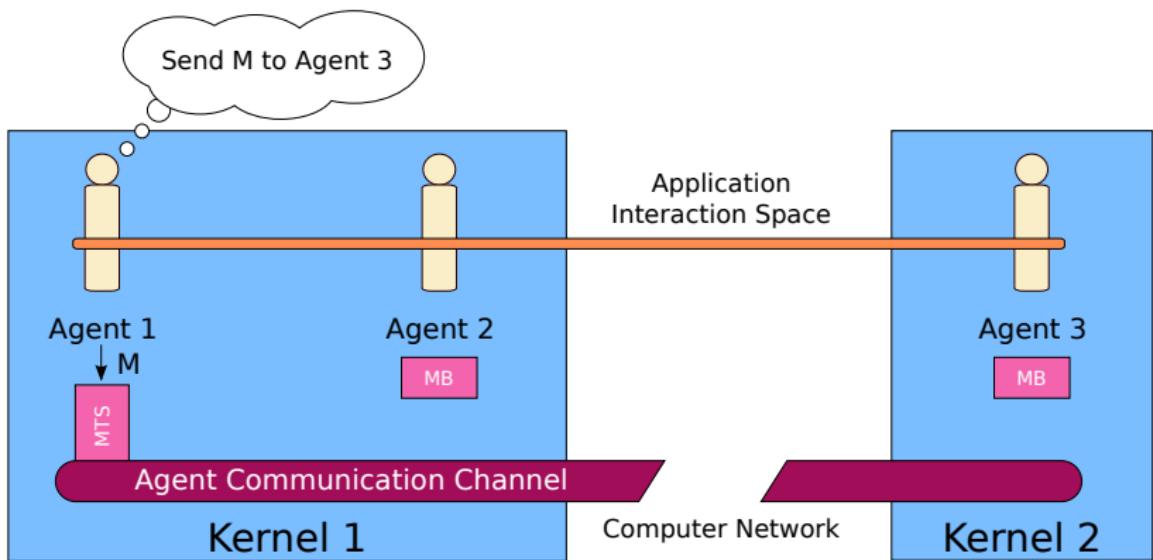


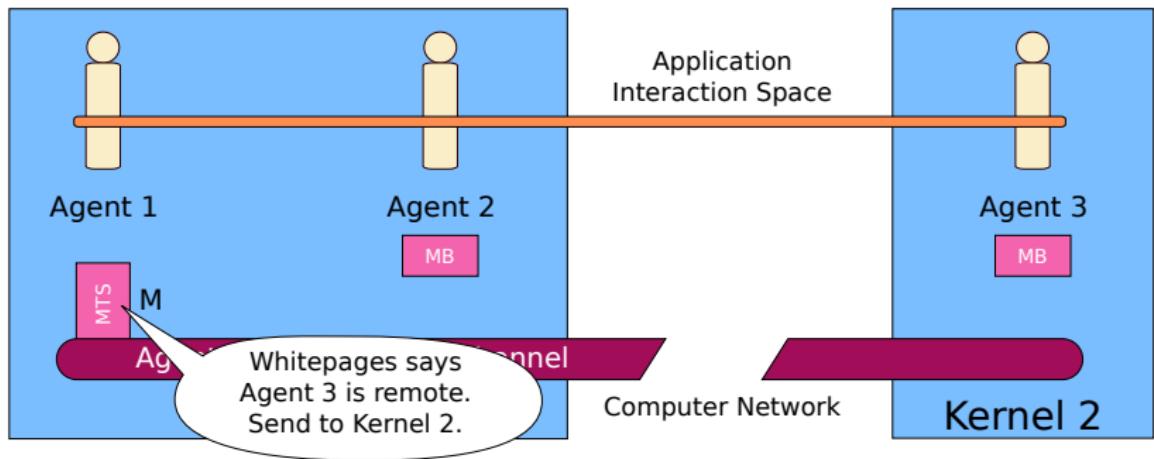


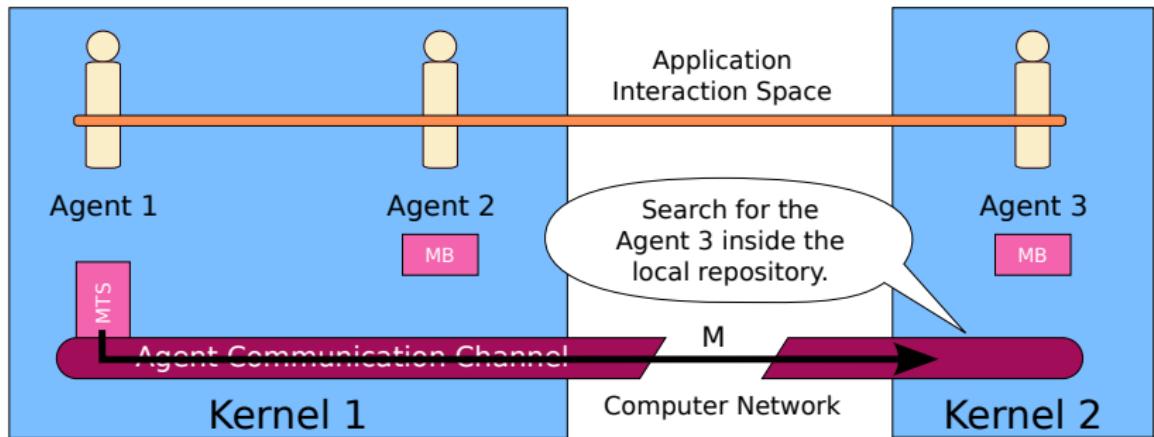


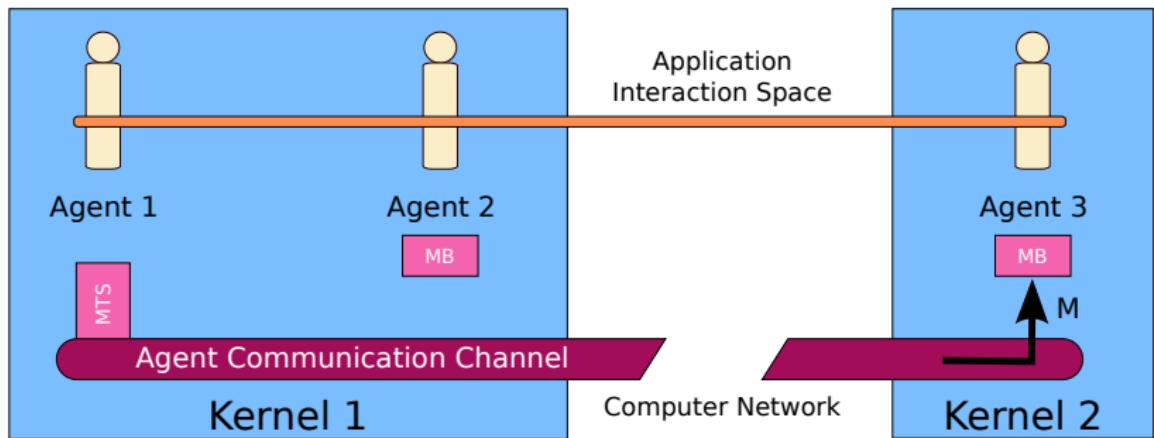


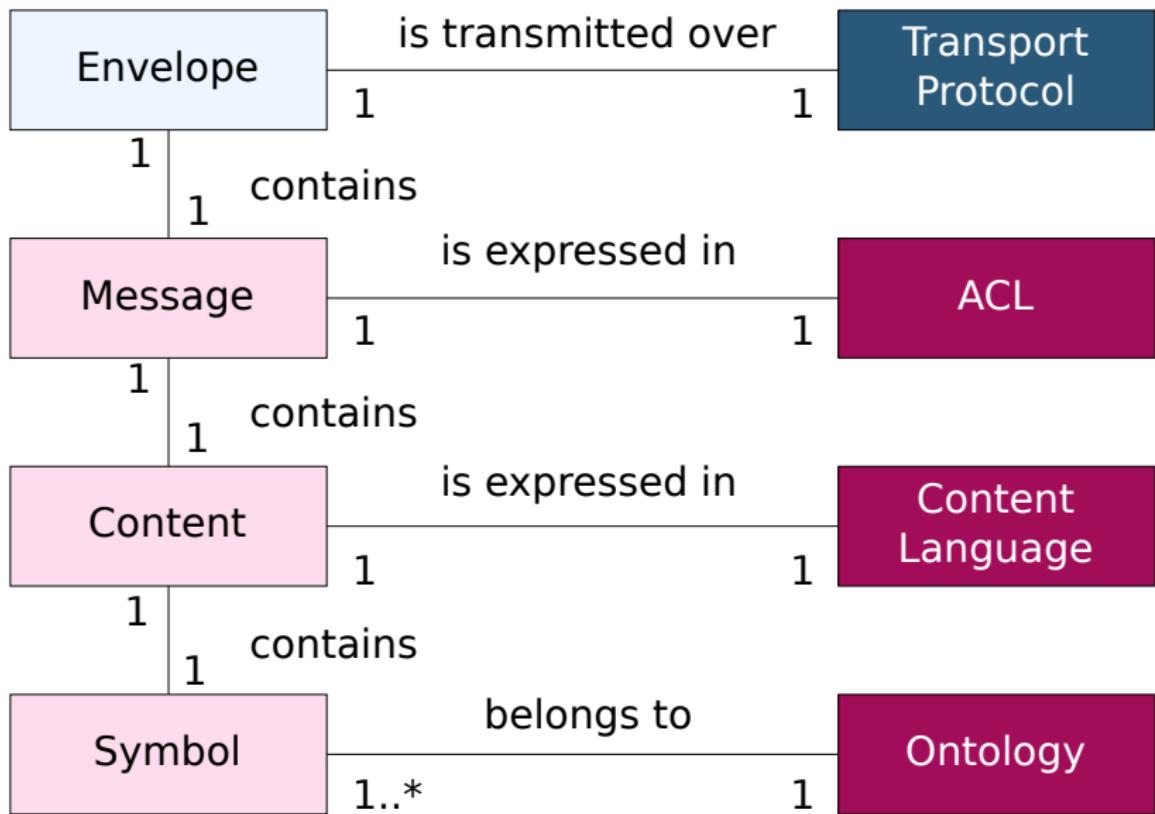












Different modes are usually available for sending a message:

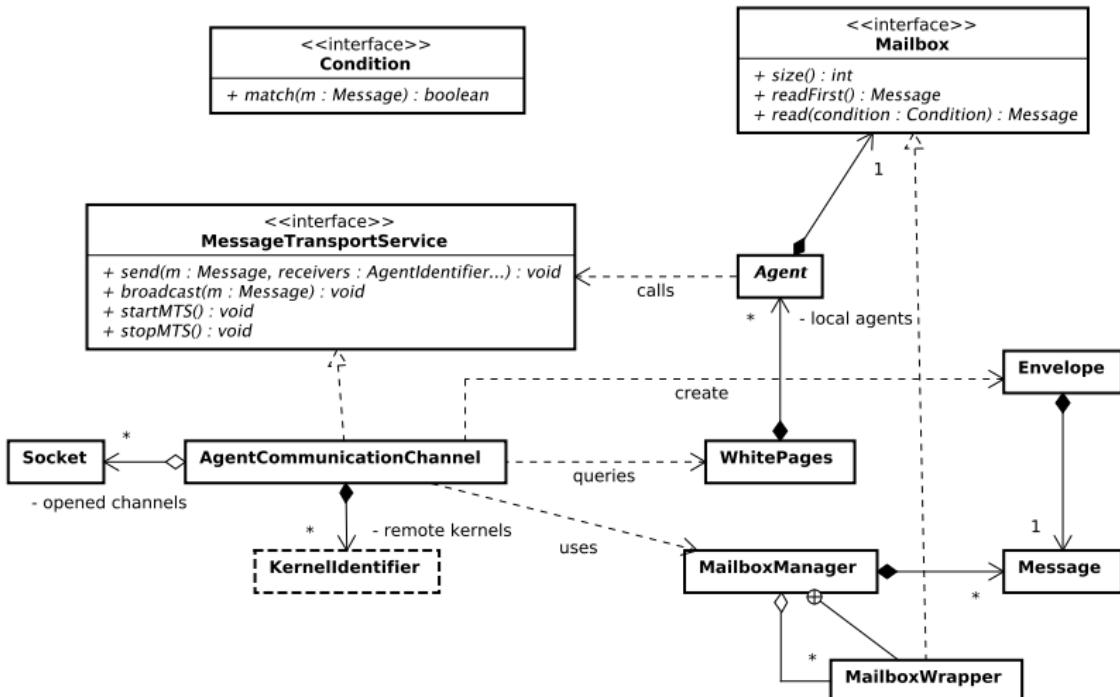
- **one-to-one:** One receiver. The emitter specifies the address of the single receiver.
- **explicit one-to-many:** Multiple receivers. The emitter specifies the addresses of the receivers.
- **implicit one-to-many:** (or broadcast) Multiple receivers. The emitter does not specify the addresses of the receivers.

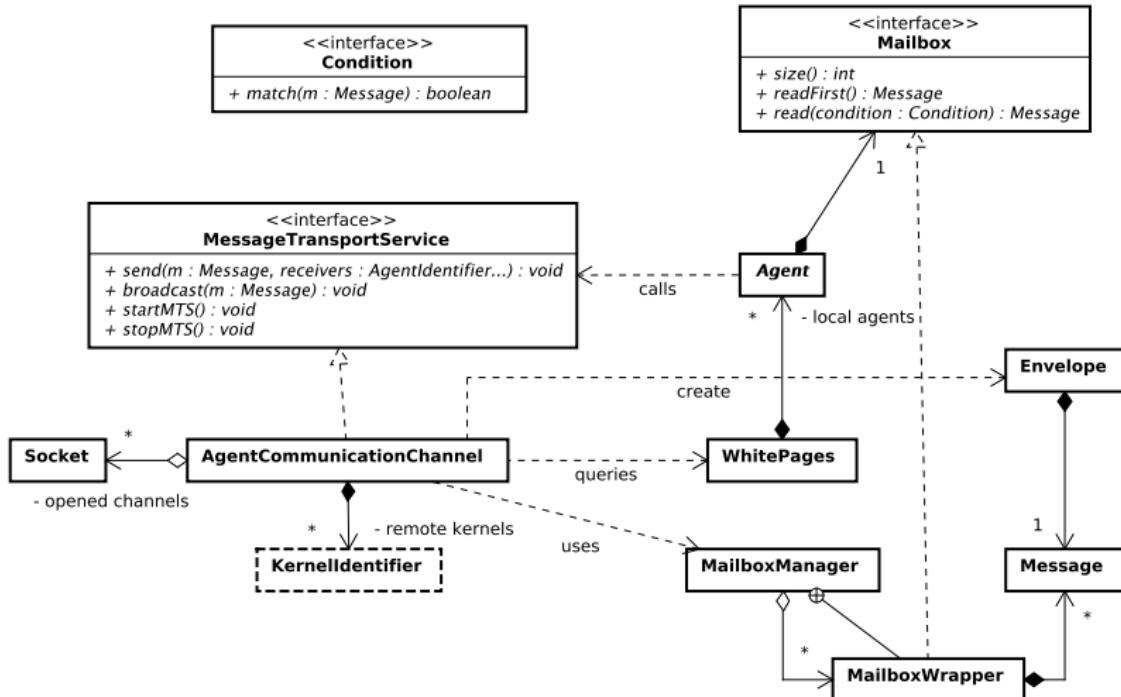
### Mailbox Approach (e.g. TinyMAS)

- The messages are put inside the **mailbox** of the receiving agent.
- The receiving agent explicitly takes a message in its behavior.
- Usually, the messages are sorted in the mailbox (fifo, timestamp, etc.)
- Accesses to the mailbox:
  - read [and consume] the first available message.
  - read [and consume] the first message that is matching a condition.

### Event/Signal Approach (e.g. Janus)

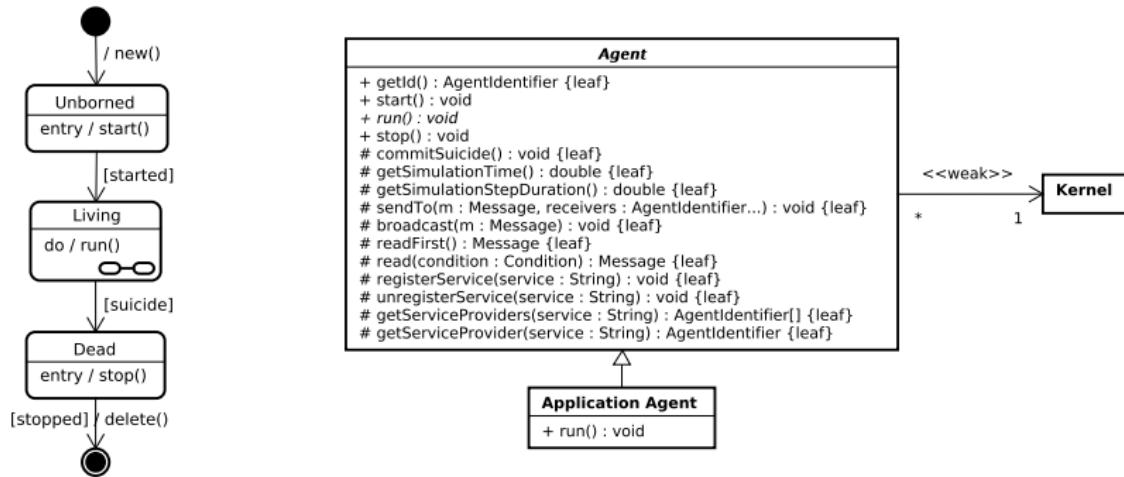
- The events (or signals) are directly fired into the context of the receiving agent.
- The agent cannot specify the event taking in its behavior. It can only specify event handlers.
- Usually, events are fifo.





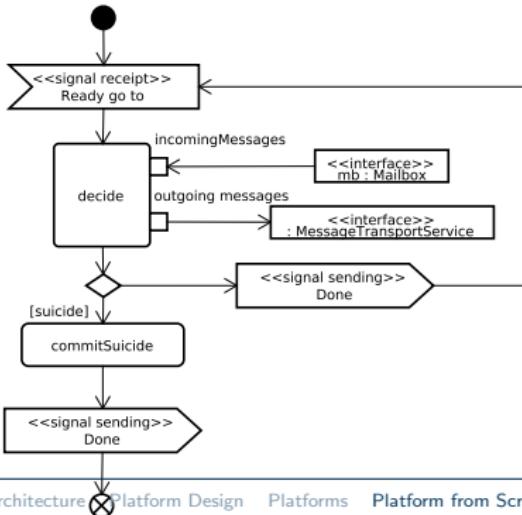
## Key points for the design

- Agent is an abstract class.
- Subclasses (defined in applications) must not access to the kernel's services directly.  
⇒ the Agent class provides protected functions for accessible operations.
- Agent must exhibits the functions related to the agents' lifecycle.



## Agent Execution

- Agents are distributed entities.
- From the agent point-of-view, they are executed in parallel [Resnick, 1995].
- The activity diagram below is executed by every agent, in parallel.



### Problem 1

How to ensure that messages are inside the mailbox at the right time?

(similar problem for the perception of the environment)

### Problem 1

How to ensure that the agent messages that must be sent at time  $t$  are all sent at this time?

(similar problem for the actions in the environment)

### Solution

Synchronization of the agents according on the input information.

[Filloque, 1992]

The treatment of an event must neither be early nor late.

- To ensure this assumption, synchronization mechanisms must be used.
- To be synchronized, you must ensure that the two following constraints are always true:
  - 1 Causality.
  - 2 Liveliness.

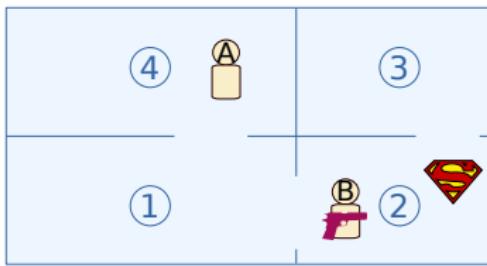
[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.

Agent A



Agent B



[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.



[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.

Agent A



Agent B



[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.

Agent A



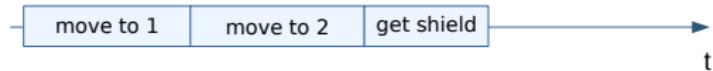
Agent B



[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.

Agent A

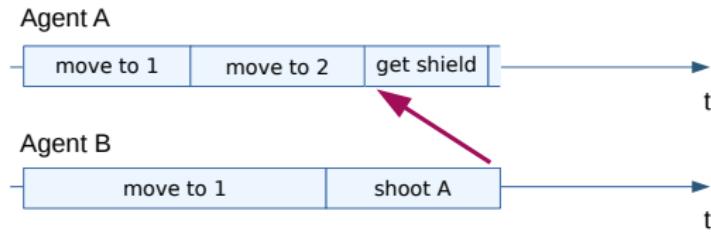


Agent B



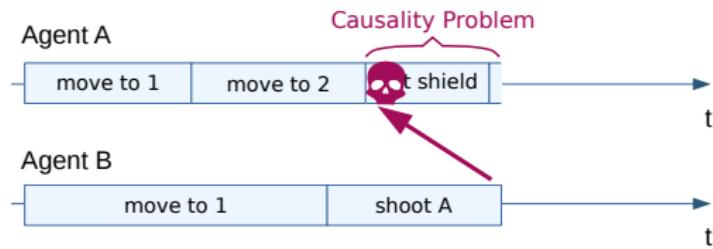
[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.



[Chandy, 1988]

The causality is the principle of no influence of the future on the past. If this constraint is respected locally and that interactions are only done with stamped messages, then the simulation is processing the messages in an order consistent with the partial orders imposed by the constraint of causality of the real system.



[Chandy, 1988]

The **liveliness** is the constraint, which is requiring the time to always evolve. Its respect allows a simulation not being in deadlock.

How to reproduce the parallel execution of the agents on a computer?

### Typical Approaches

- Run agents on a computer network, or a grid computer.
- Associate one thread per agent.
- Run synchronously the agents inside a loop, aka. **scheduling of the execution** of the agents.

## Definition

The execution scheduling imposes a sequential execution of the agents having only a notion of distributed or parallel execution.

- When simulating with a multi-agent system, it is assumed that the actions of the agents are synchronized [Michel, 2007].
- The execution of the agents (order, etc.) has a strong impact on the results provided by the simulation [Lawson, 2000].

## Definition

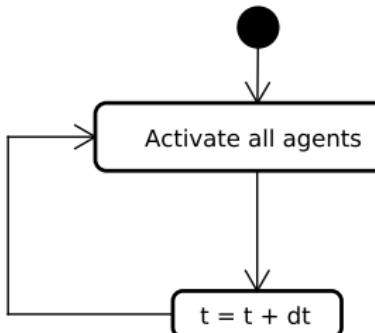
- A loop-based scheduling is a discrete execution policy that executes the agents inside a virtual loop.
- At each loop step, the time evolves with a predefined amount.

## Two approaches in this Lecture

- 1 Basic constant-step scheduling.
- 2 Buffering constant-step scheduling.

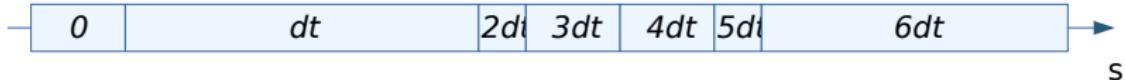
## Principle

- 1 Sequential activation of all the agents:
  - Perception and actions are done when they are requested.
- 2 Increment the time by a predefined amount.



```
t ← 0
loop
    for all  $a \in AGENTS$  do
        LIVE( $a$ )
    end for
     $t \leftarrow t + \Delta t$ 
end loop
```

## Agents



## Agent Framework Kernel



## Operating system



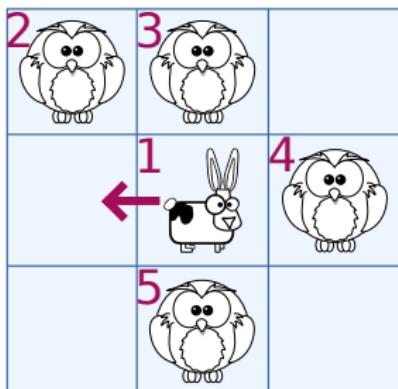
## Processor



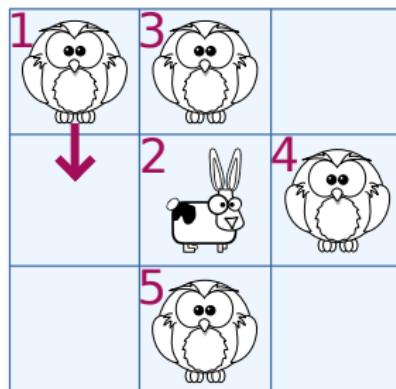
## Human being



- Let a prey try to escape predators.
- Let predators try to catch the prey.
- Let the environment be a 3x3 grid.
- Agents are inserted inside the execution list at the given indexes.
- In the model,  $t$  is the same for all the agents  $\Rightarrow$  they perceive and act at the same time.



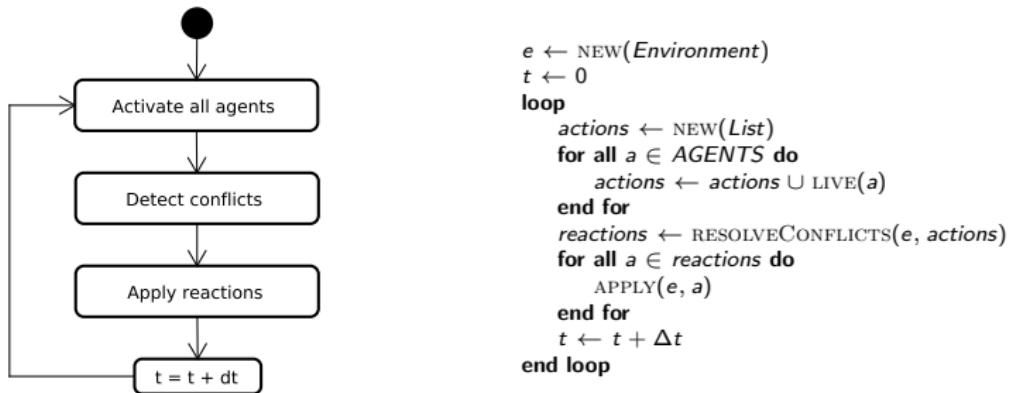
Prey is escaping

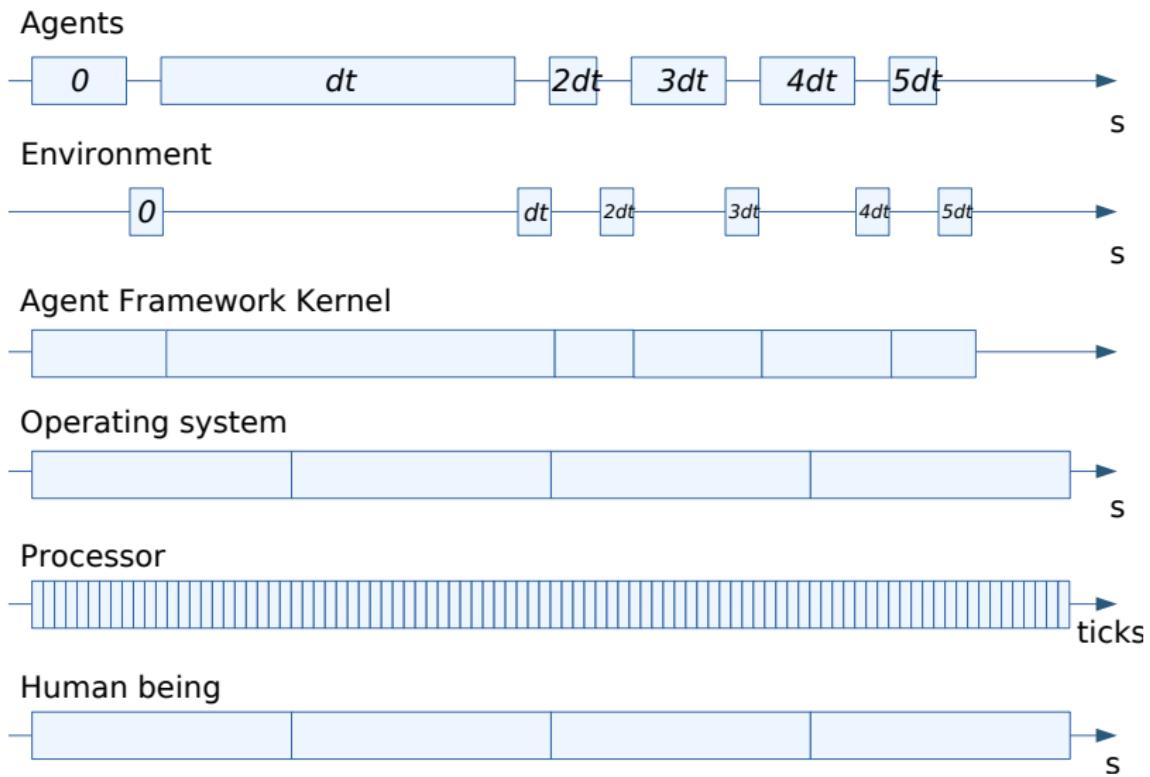


Prey is catched

## Principle

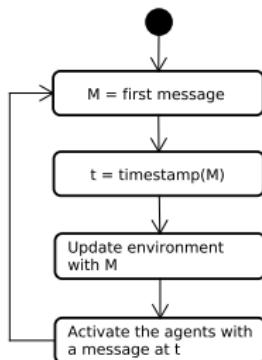
- 1 Sequential activation of all the agents:
  - Perceptions are on the same environment state for all agents.
  - Actions are gathered, but never directly applied.
- 2 Conflicts among agent's actions are detected and solved.
- 3 Corresponding reactions are applied on the environment.
- 4 Increment the time by a predefined amount.





## Basic Principle

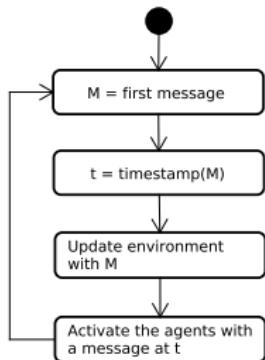
- 1 Determine the messages with the minimal timestamp.
- 2 Test if the causality is broken.
- 3 Set  $t$  to the minimal timestamp.
- 4 Activate the agents.



```
e ← NEW(Environment)
mb ← NEW(MailboxManager)
t ← 0
repeat
    ( $m_t, msgs$ ) ← GETMIN(mb)
    ASSERT( $m_t > t$ )
     $t \leftarrow m_t$ 
    UPDATEENVIRONMENT(e, msgs)
    for all  $m \in msgs$  do
        for all  $a \in AGENTS$  do
            if RECEIVER(m) = a then
                LIVE(a, m)
            end if
        end for
    end for
until  $mb = \emptyset$ 
```

## Basic Principle

- 1 Determine the messages with the minimal timestamp.
- 2 Test if the causality is broken.
- 3 Set  $t$  to the minimal timestamp.
- 4 Activate the agents.



```
e ← NEW(Environment)
mb ← NEW(MailboxManager)
t ← 0
repeat
    ( $m_t$ , msgs) ← GETMIN(mb)
    ASSERT( $m_t > t$ )
     $t \leftarrow m_t$ 
    UPDATEENVIRONMENT(e, msgs)
    for all  $m \in msgs$  do
        for all  $a \in AGENTS$  do
            if RECEIVER(m) = a then
                LIVE(a, m)
            end if
        end for
    end for
until  $mb = \emptyset$ 
```

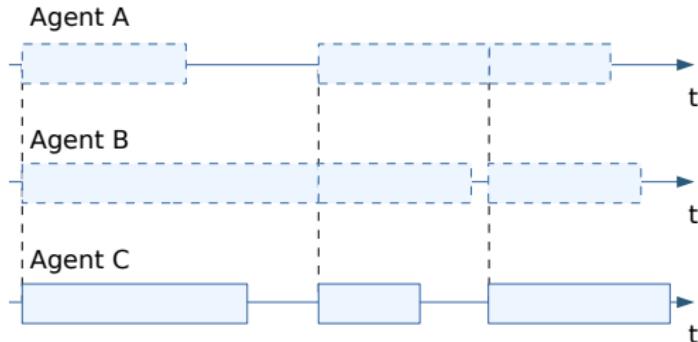
How to ensure this assertion?

- **Pessimistic approach:** time progresses for all the agents at the same time. Fast agents must wait for slow agents.
- **Optimistic approach:** time progresses for each agent as fast as possible, and the system go back in time in case of causality problem.
- **Hybrid approach:** mixing the two previous approaches.

[Chandy, 1979], adapted to agent-based systems

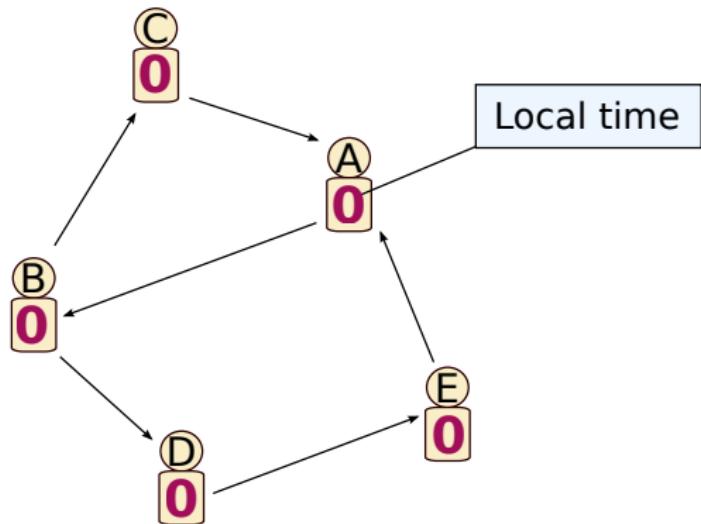
An agent must treat a message only if it is sure that it will never receive any message with a timestamp lower than or equal to its current agent time.

- **Causality:** It is never broken.

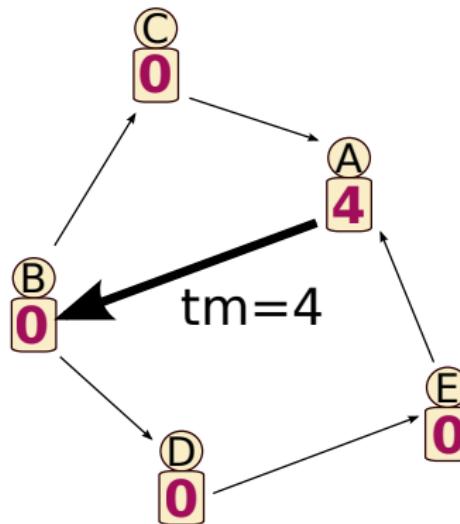


- **Liveliness:** Must be managed by hand.

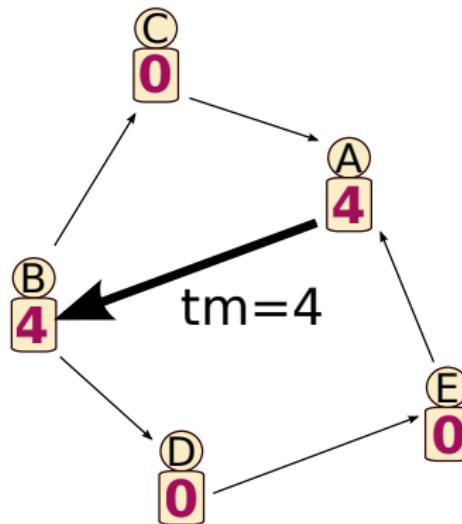
t=0 : A generates the start event.



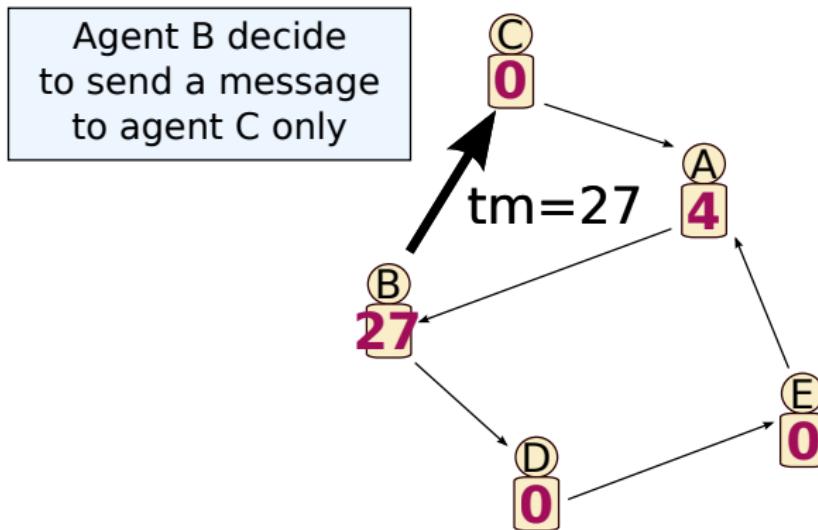
t=0 : A generates the start event.



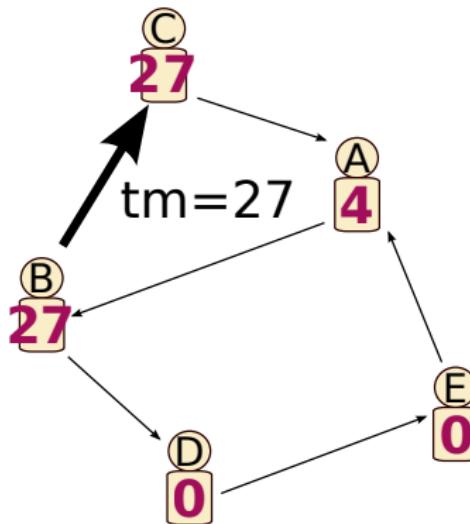
t=0 : A generates the start event.



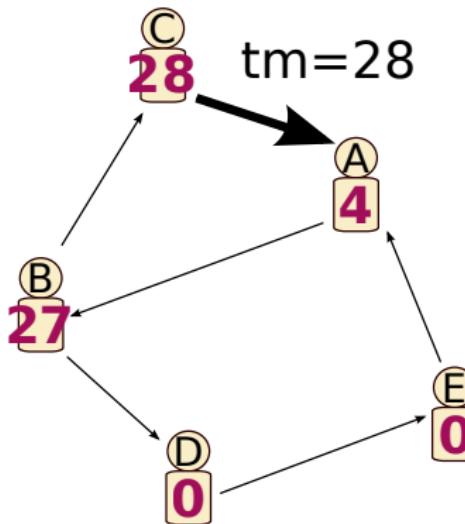
t=0 : A generates the start event.



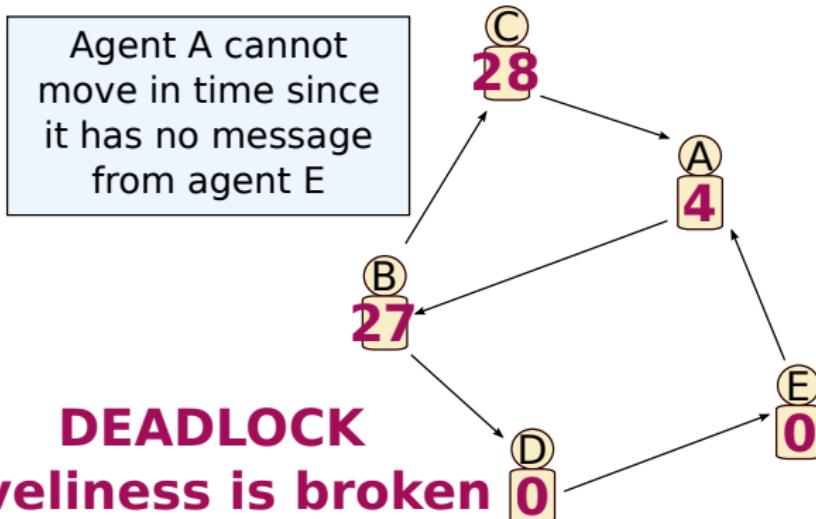
t=0 : A generates the start event.



t=0 : A generates the start event.

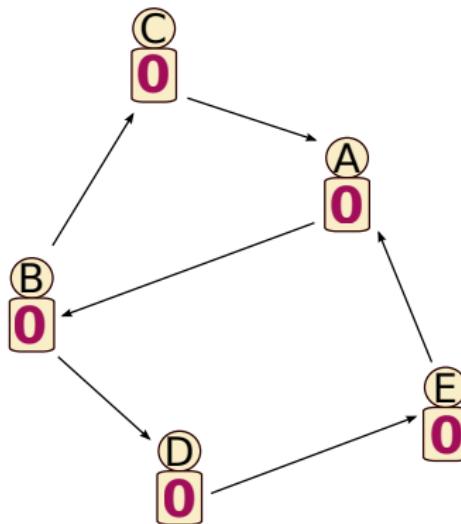


t=0 : A generates the start event.

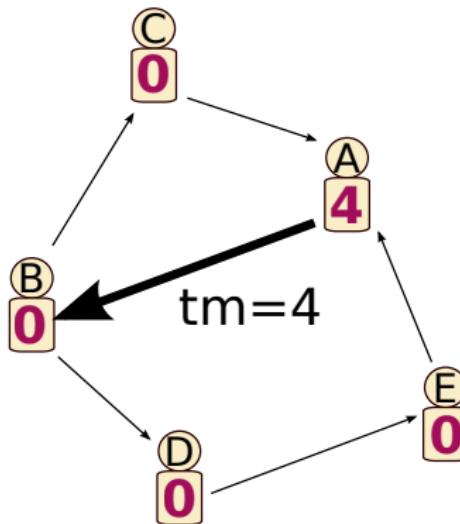


- NULL message is an empty message with a timestamp [Chandy, 1979].
- NULL messages are sent to the connected agents when no other type of message is available.

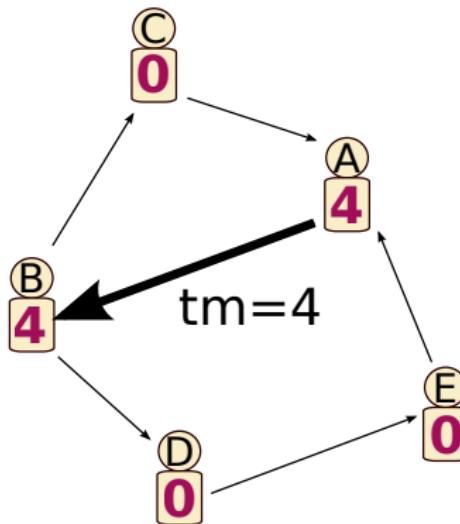
t=0 : A generates the start event.



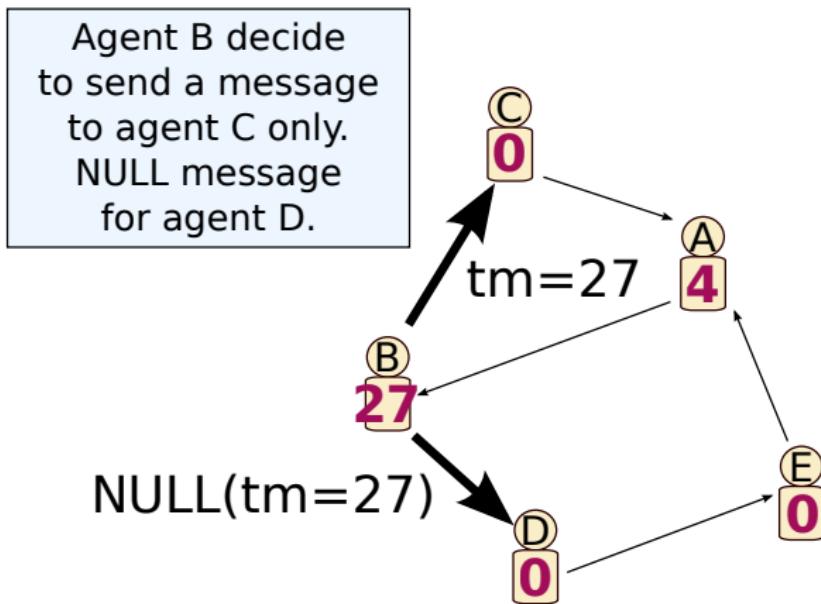
t=0 : A generates the start event.



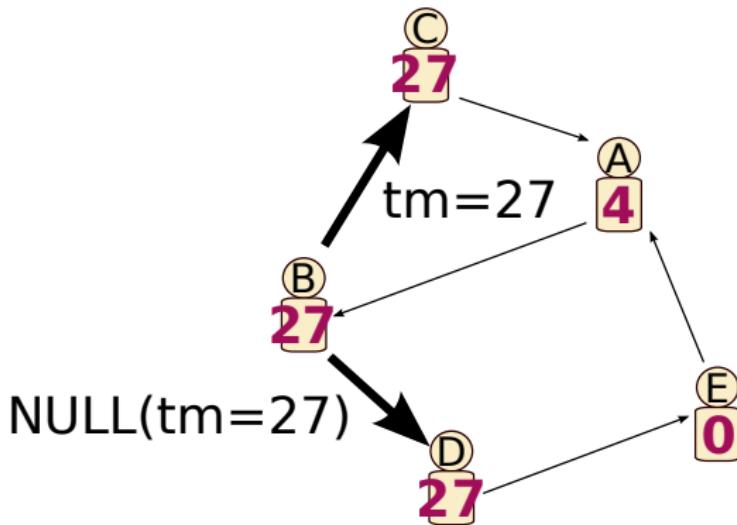
t=0 : A generates the start event.



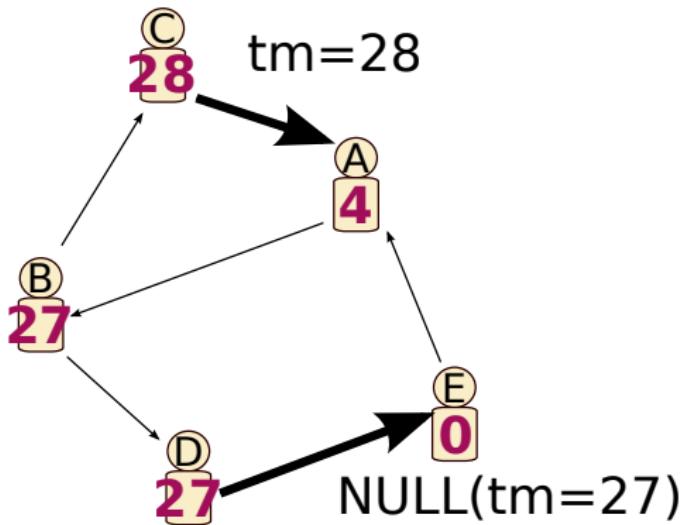
t=0 : A generates the start event.



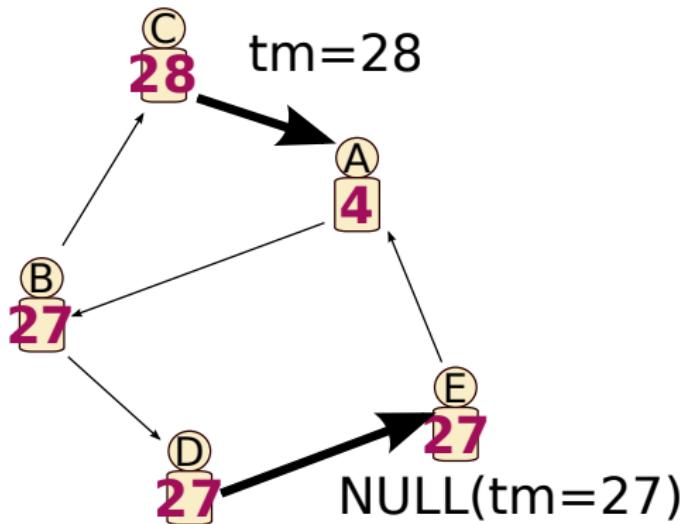
t=0 : A generates the start event.



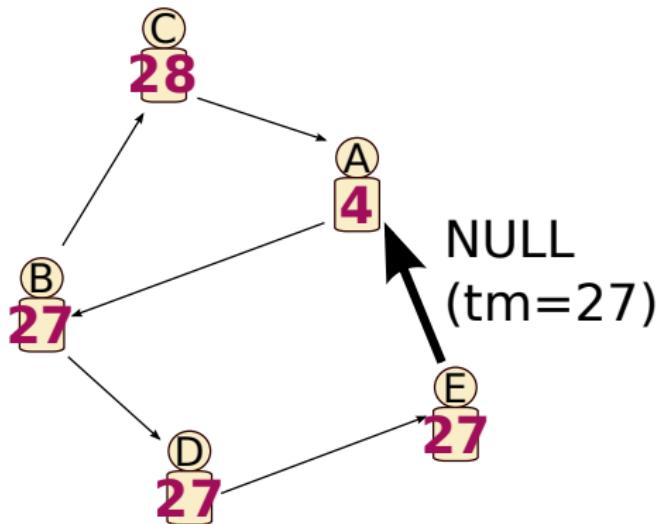
t=0 : A generates the start event.



t=0 : A generates the start event.



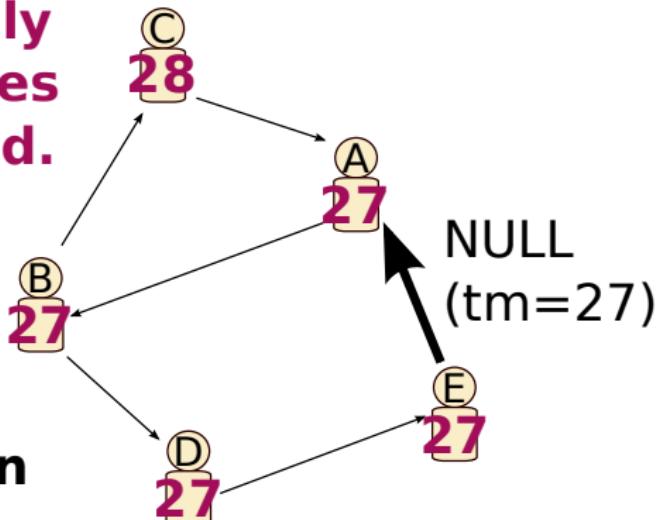
t=0 : A generates the start event.



t=0 : A generates the start event.

**Deadlock is still possible if only NULL messages are exchanged.**

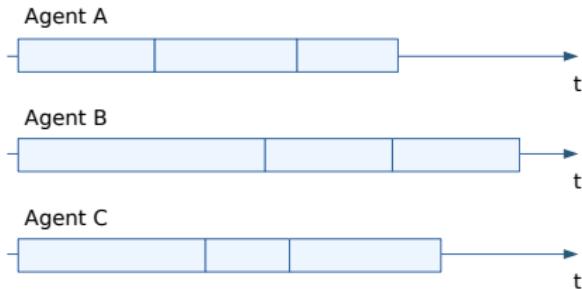
**Solution:**  
**tm of NULL = tm + epsilon**



[Fujimoto, 1990], adapted to agent-based systems

Optimistic approach does not impose any constraint on the simulation model. It is running the agents as fast as possible until a causality problem is detected. Then the simulator goes back in time to the last valid state of the simulator.

- **Liveliness:** It is never broken.



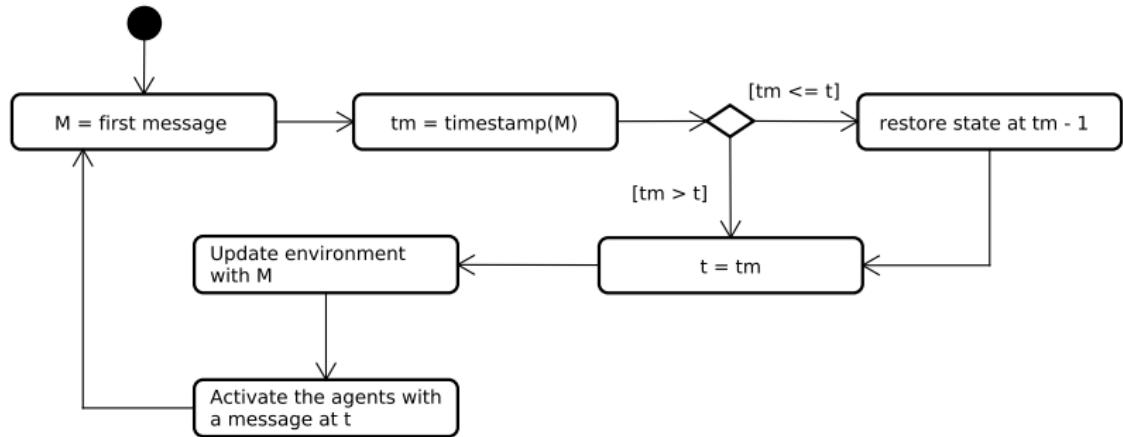
- **Causality:** It is broken when a message received by an agent has a timestamp lower than or equal to the agent's time.

## Solution

- The last correct state is detected from the history of changes.
- The state of the agent is restored to this last correct state.
- All sent messages must be cancelled!
- The state of the environment is restored.

## Assumptions

- The time spent to go back is negligible.
- The simulator has enough memory to store all the states.



[Reynolds, 1988]

Purely pessimistic or optimistic approaches are not suitable in real case studies.

Hybrid approaches use both previous approaches at the same time.

### Example (1)

Safety indicators are computed for each message. Safe messages do not stop the simulator as for the optimistic approach. Unsafe messages are blocking the simulator as for the standard pessimistic approach.

### Example (2)

The simulator may use an optimistic approach until it reaches a rendez-vous time.

## State of an Agent

Each agent is composed of three major elements:

- **Identity or address:** The property of an agent that distinguishes it from other agents.
- **Attributes:** It describes the data stored in the agents, aka. the attributes.
- **Behavior:** It describes the behavior of the agent (its functions activate, live and end, and all the functions called by one of them).

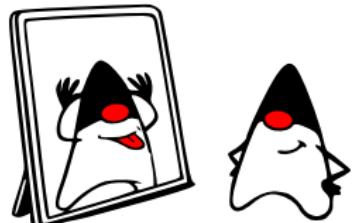
The state of an agent is close to the concept of state of an object in object-oriented computer programming. It is built with the identity and the attributes.

## Probe

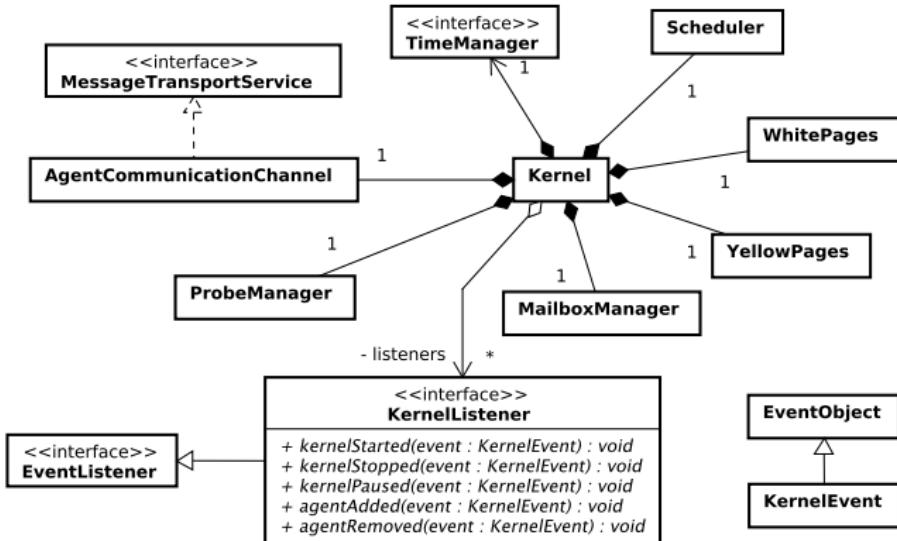
- A probe is a tool that permits to retrieve the state of an agent.
- A probe must be non-invasive for the agent. It means that the agent cannot determine when a probe is getting a value from its state. Moreover, an agent is not able to obtain the list of the probes on it, easily.

The reflection mechanism of the Java runtime environment will enable you to retrieve the attribute values in a non-invasive way.

```
public class ProbeManager{  
  
    public <T> T getProbeValue(Class<T> valueType,  
                             Agent agent, String name){  
  
        Class<?> type = agent.getClass();  
        while (!Agent.class.equals(type)){  
  
            try {  
                return valueType.cast(  
                    type.getDeclaredField(name).get(agent));  
            } catch (Throwable _){}  
  
            type = type.getSuperclass();  
        }  
  
        throw new NoSuchFieldException(name)  
    }  
}
```



- The kernel is the central class of the platform.
- It contains all the internal and external services of the platform.
- It provides to the agents an interface accessing the platform services.



## 1 Introduction

## 2 FIPA Standard: Platform Features

## 3 FIPA Standard: Platform Architecture

## 4 Designing a Platform

## 5 A Brief Listing of Existing Platforms

## 6 Agent Platform From Scratch: the example of the TinyMAS platform

## 7 Service-based Platform: the example of the Janus platform

- Designed for a large set of application domains (academic, research, industrial).
- It supports the execution of agents written with the SARC language.
- Use a fully parallel execution, via extensive use of thread pools.
- Native and transparent support of computer networks.
- Easy to plug/unplug kernel instances from a collection of connected computers.

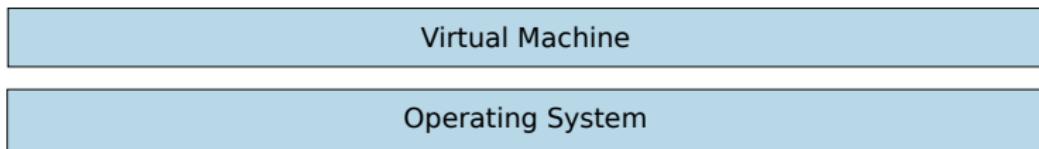


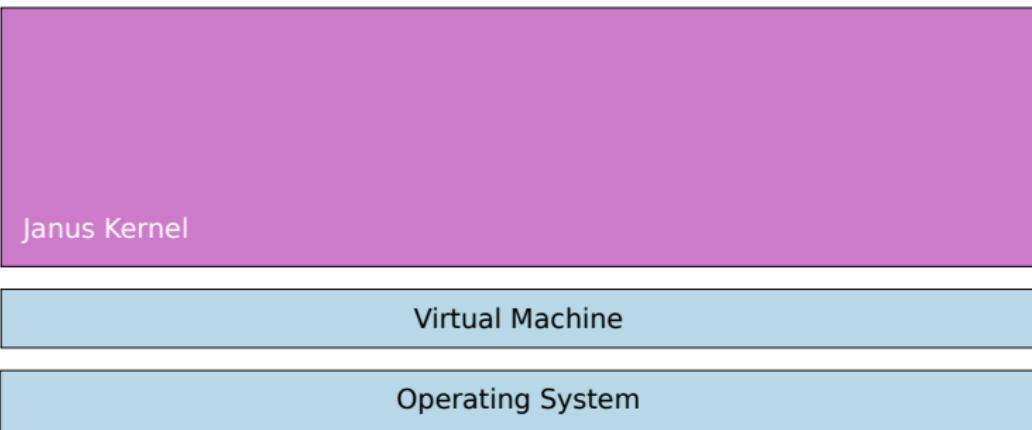
Official website:

<http://www.janusproject.io/>

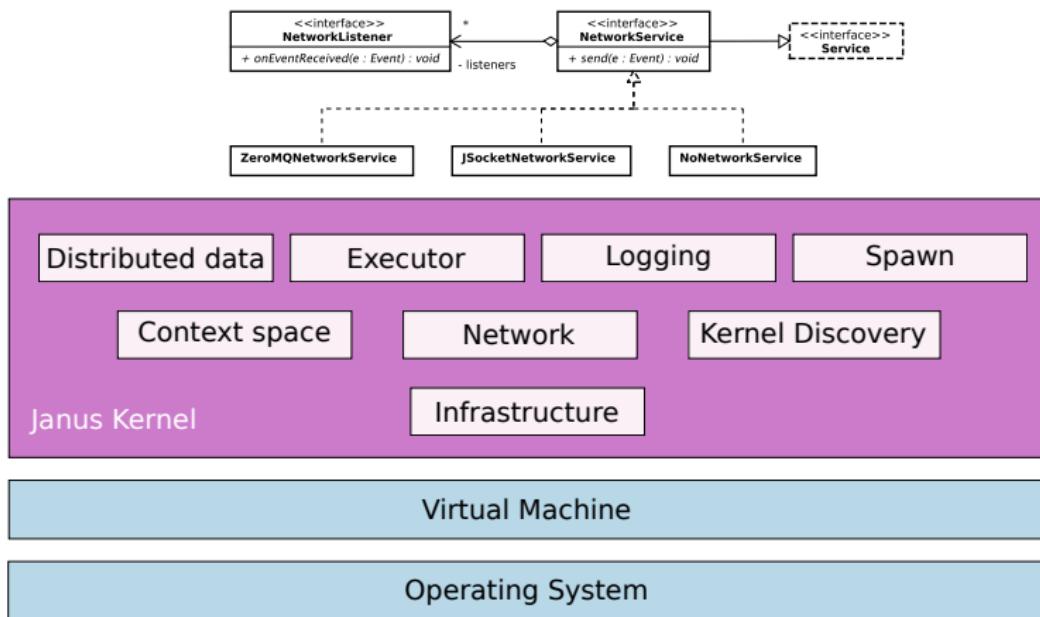
Source code:

<https://github.com/janus-project/janusproject/>

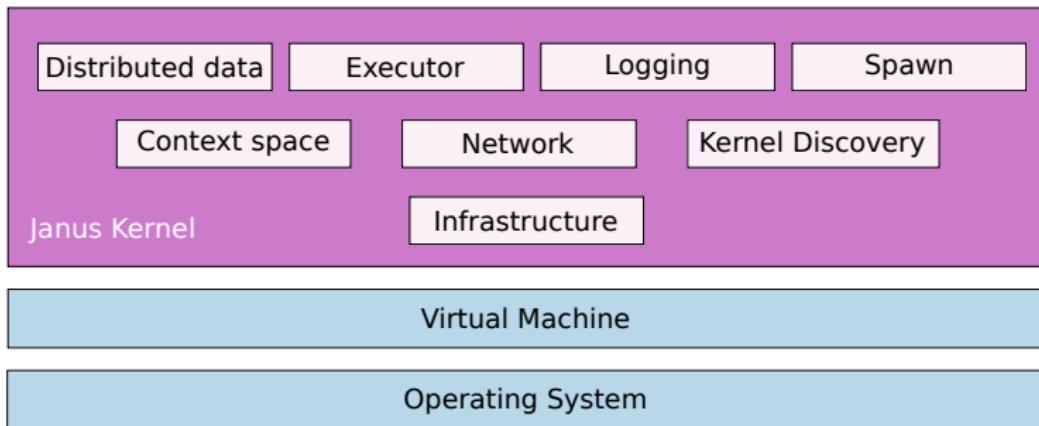
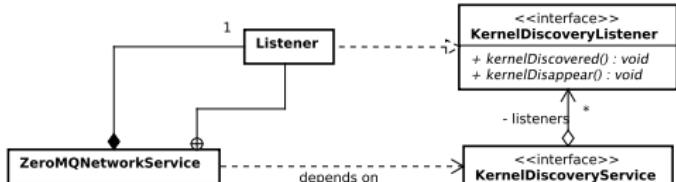




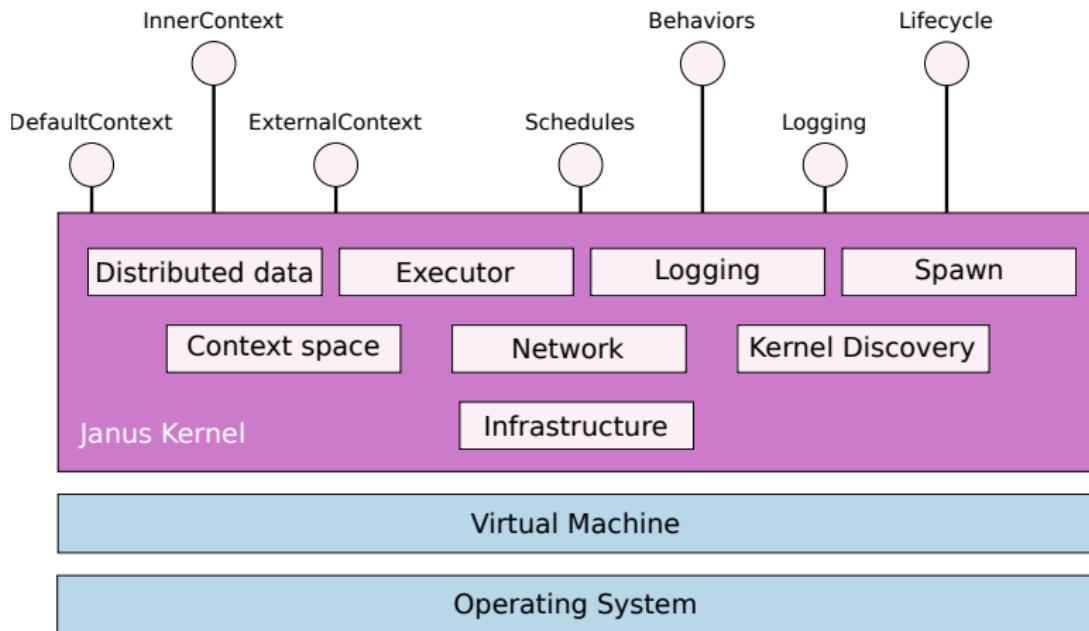
Services implementations are **injected into the Kernel** (with Google Guice) according to the platform configuration.



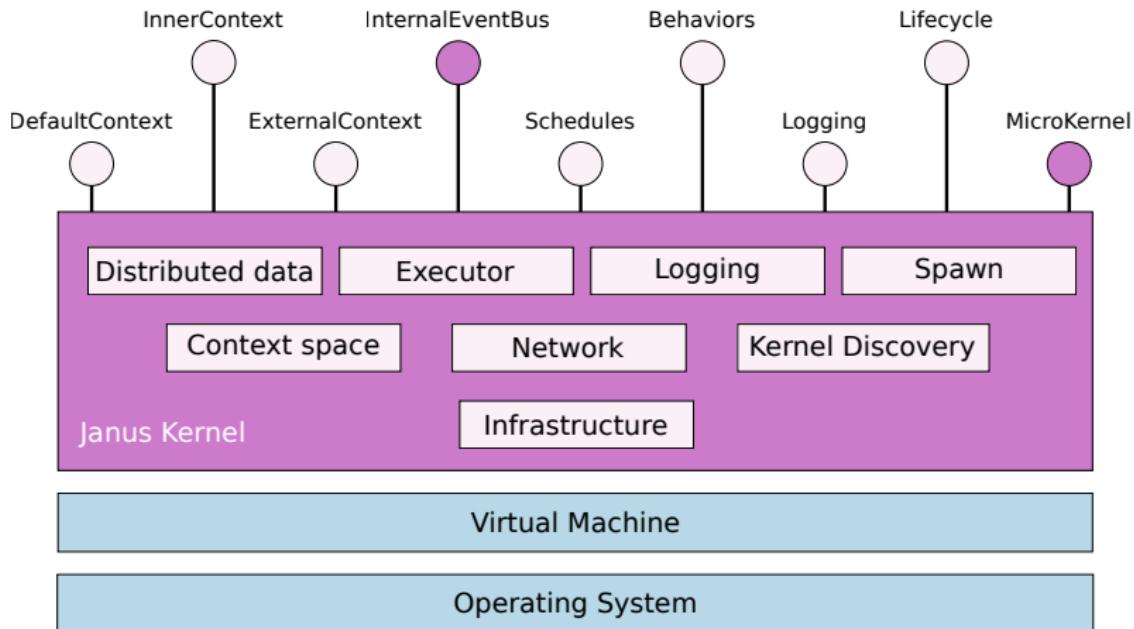
A service implementation could depends on another service. The service manager of the kernel is responsible of launching/stopping the services in the correct order.

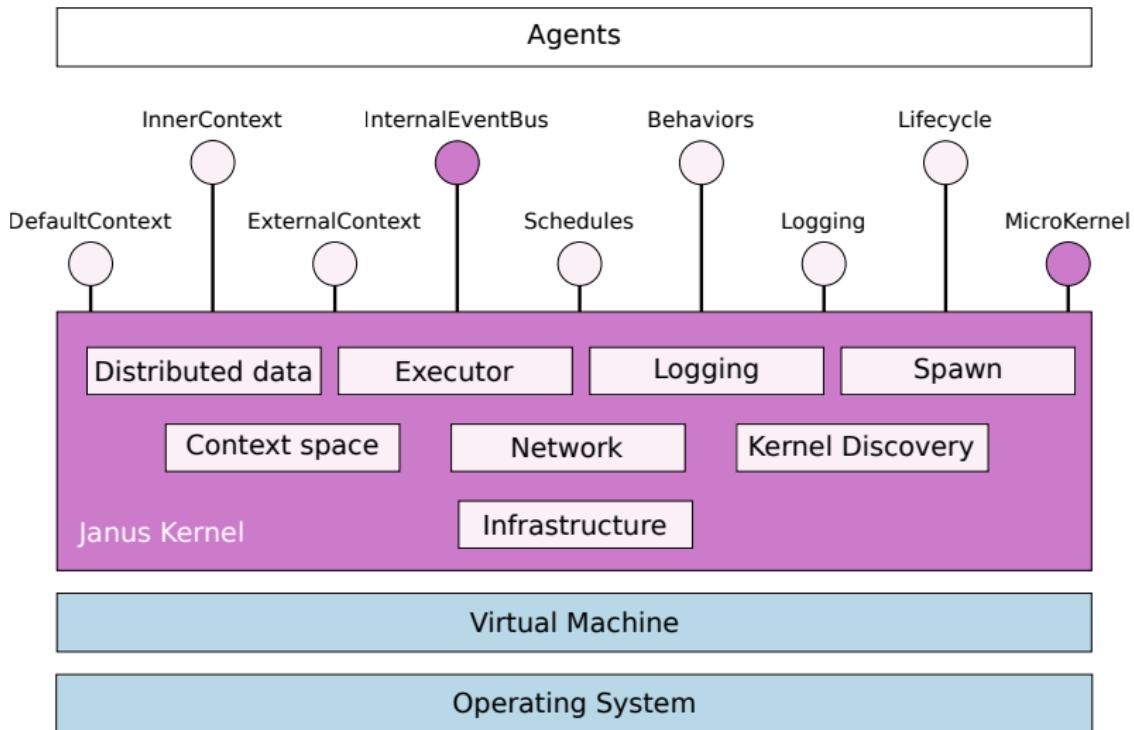


Platform provides implementations of the built-in capacities that are defined into the specifications of the SARL language.



Several capacities are specific to the Janus platform.  
It is not recommended to use them into agents. But they may be useful for creating exploration features.





- The **key features** of an agent platform are related to technological, domain, development, analysis, and exploration needs.
- The platform provides the features for the **targeted users**:
  - **Beginners**: a nice UI for creating the agents, e.g. NetLogo.
  - **Experts**: a powerful programming API, e.g. Janus.
  - **All**: the lower-level software/hardware issues must be hidden to agent developers.
- A “good” platform provides **distributed execution** via threads, computer networking, or both.
- The **typical architecture** is usually service oriented.
- The platform should be coded with **design patterns** for making easier its code upgrades.
- It must be **efficient** for almost all the target users.



**Thank you for your attention...**



# Appendix

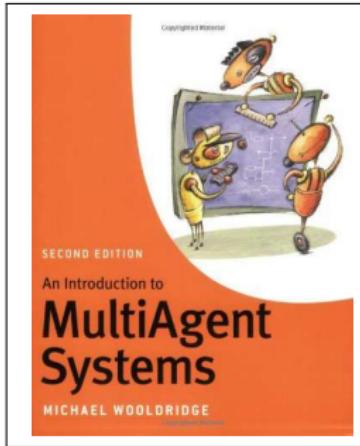


## 1 Books

- Multiagent Systems
- Simulation Theory
- Games and Serious Games
- Transport
- Mathematics

## 2 About the Authors

## 3 Bibliography



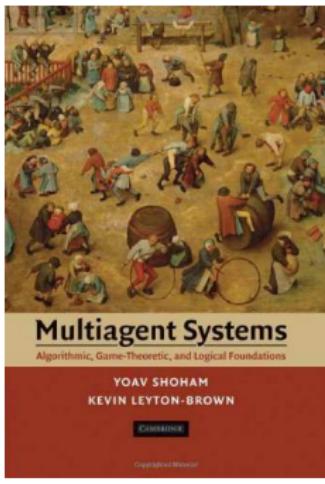
## An Introduction to Multiagent Systems

2nd edition

Michael WOOLDRIDGE

Wiley, 2009

ISBN 0-47-0519460

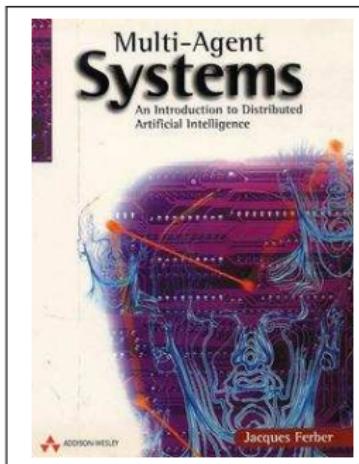


## Multiagent Systems: algorithmic, game-theoretic, and logical foundations

Yoav SHOHAM and  
Kevin LEYTON-BROWN

Cambridge University Press, 2008

ISBN 0-52-1899435

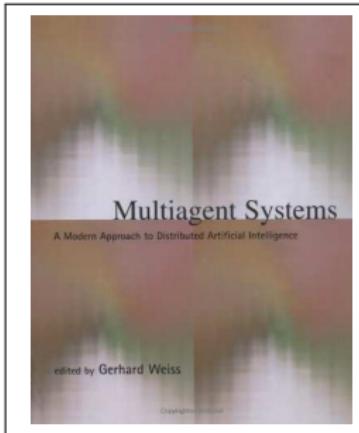


## **Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence**

Jacques FERBER

Addison Wesley, 1999

ISBN 0-20-1360489

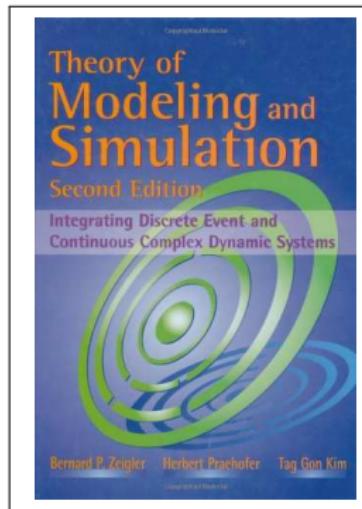


## **Multiagent Systems: a modern approach to distributed Artificial Intelligence**

Gerhard WEISS

MIT Press, 2000

ISBN 0-26-2731312



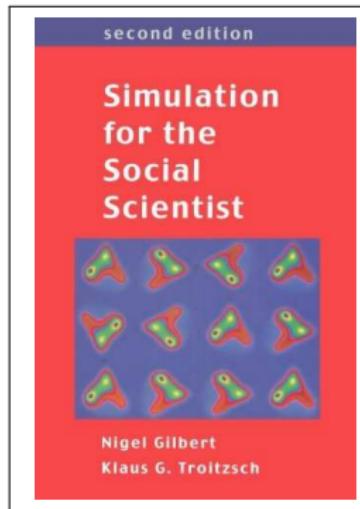
## Theory of Modeling and Simulation

2nd edition

Bernard ZEIGLER, Herbert Praehofer, and  
Tag Gon Kim

Academic Press, 2000

ISBN 0-12-7784551



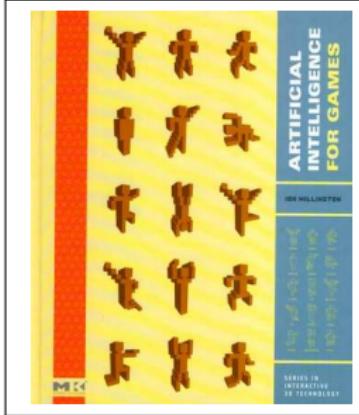
## Simulation for the Social Scientist

2nd edition

Nigel GILBERT and Klaus TROITZSCH

Open University Press, 2005

ISBN 0-33-5216005

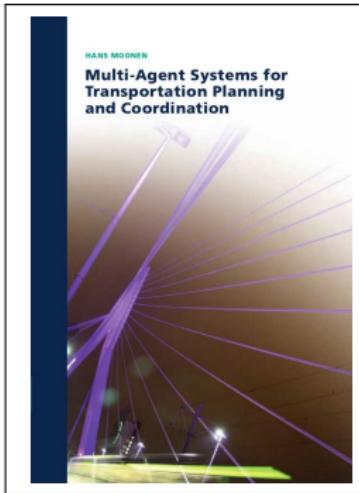


## Artificial Intelligence for Games

Ian MILLINGTON

Morgan Kaufmann Publishers & Elsevier  
Science, 2006

ISBN 0-12-4977820

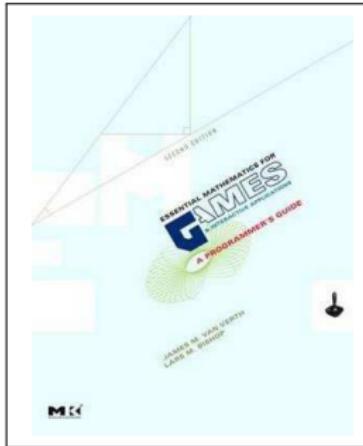


## **Multi-Agent Systems for Transportation Planning and Coordination**

Hans MOONEN

Erasmus Research Institute of  
Management, 2009

ISBN 978-90-5892-216-8



## Essential Mathematics for Games & Interactive Applications: a programmer's guide

2nd edition

James VAN VERTH and Lars BISHOP

Morgan Kaufmann Publishers, 2008

ISBN 0-12-3742971



## **Calculabilité, Complexité et Approximation**

Jean-François REY

Vuibert France, 2004

ISBN 2-71-1748081

1 Books

2 About the Authors

3 Bibliography

*Professor*

Université de Bourgogne Franche-Comté

Université de Technologie de Belfort-Montbéliard, France



**Topics: Multiagent systems, Agent-based simulation, Agent-oriented software engineering, Mobility and traffic modeling**

Web page: [http://www.multiagent.fr/People:Galland\\_stephane](http://www.multiagent.fr/People:Galland_stephane)  
Email: [stephane.galland@utbm.fr](mailto:stephane.galland@utbm.fr)

Open-source contributions:

- <http://www.sarl.io>
- <http://www.janusproject.io>
- <http://www.aspecs.org>
- <http://www.arakhne.org>
- <https://github.com/gallandarakhneorg/>

*Associate Professor*

Université de Bourgogne Franche-Comté

Université de Technologie de Belfort-Montbéliard, France



**Topics: Multiagent systems, Agent-based simulation, Agent-oriented software engineering**

Web page: [http://www.multiagent.fr/People:Gaud\\_nicolas](http://www.multiagent.fr/People:Gaud_nicolas)  
Email: nicolas.gaud@utbm.fr

Open-source contributions:

- <http://www.sarl.io>
- <http://www.janusproject.io>
- <http://www.aspecs.org>
- <http://www.arakhne.org>



(2008).

Cyber-physical systems.

Adam, E. (2000).

*Modèle d'organisation multi-agent pour l'aide au travail coopératif dans les processus d'entreprise: application aux systèmes administratifs complexes.*

PhD thesis, Univ. de valenciennes et du hainaut-cambresis.

Agre, P. and Horwitz, I. (1997).

Lifeworld analysis.

*Journal of Artificial Intelligence Research*, 6(1):111–145.

Amant, R. (2005).

Tool use for autonomous agents.

In Veloso, M. and Kambhampati, S., editors, *AAAI/IAAI'05 Conference*, pages 184–189, Pittsburgh, PA, USA. AAAI Press/The MIT Press.

Argente, E., Julian, V., and Botti, V. (2006).

Multi-agent system development based on organizations.

In *First International Workshop on Coordination and Organisation (CoOrg)*, volume 150 of *Electronic Notes in Theoretical Computer Science*, pages 55–71. Elsevier.

Banks, J. (1999).

Introduction to simulation.

In Farrington, P., Nembhard, H., Sturrok, D., and Evans, G., editors, *Winter Simulation Conference*, pages 7–13.

Barbat, B., Candea, C., and Zamfirescu, C. (2001).

Holons and Agents in Robotic Teams. A Synergistic approach.

In *proceedings of ENAIS'2001*, pages 654–660.

Baum, L. and Petrie, T. (1966).

Statistical inference for probabilistic functions of finite state markov chains.

*Annals of Mathematical Statistics*, 37(6):1554–1563.

Bäumer, C., Breugst, M., Choy, S., and Magedanz, T. (2000).

Grasshopper: a universal agent platform based on omg masif and fipa standards.

Technical report, IKV++ GmbH.

Bäumer, C. and Magedanz, T. (1999).

Grasshopper - a mobile agent platform for active telecommunication networks.

In *Intelligent Agents for Telecommunication Applications: Third International Workshop (IATA)*, volume 1699 of *LNCS*, pages 19–32. Springer Berlin / Heidelberg, Stockholm, Sweden.

Bellifemine, F., Poggi, A., and Rimassa, G. (2001).

JADE: a FIPA2000 compliant agent development environment.

In *Agents*, pages 216–217.

Braun, A., Bodmann, B. E. J., and Musse, S. R. (2005).

Simulating virtual crowds in emergency situations.

In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST '05*, pages 244–252, New York, NY, USA. ACM.

Brooks, R. (1990).

Elephants don't play chess.

*Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*.

Brussel, H. V., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P. (1998).

Reference architecture for Holonic Manufacturing Systems: PROSA.

*Computers in Industry*, 37:255–274.

Buisson, J., Galland, S., Gaud, N., Yasar, Gonçalves, M., and Koukam, A. (2013).

Real-time collision avoidance for pedestrian and bicyclist simulation: a smooth and predictive approach.

In *the 2nd International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS)*, Halifax, Nova Scotia, Canada. Procedia Computer Science, Elsevier.

Buisson, J., Gaud, N., Galland, S., Gonçalves, M., and Koukam, A. (2014).

Toward an environment for the simulation of heterogeneous entities in virtual cities.

In *International Workshop on Environments for Multiagent Systems (E4MAS14)*, Paris, France. IFAAMAS, Springer.

Bürckert, H., Fischer, K., and G.Vierke (1998).

Transportation Scheduling with Holonic MAS - The TeleTruck Approach.

In *Conf. on Practical Applications of Intelligent Agents and Multiagents*, pages 577–590.

Burghout, W. (2004).

*Hybrid microscopic-mesoscopic traffic simulation*.

PhD thesis, Royal Institute of Technology, Stockholm, Sweden.

Candea, C., Staicu, M., and Barbat, B. (2000).

Holon - Like Approach for Robotic Soccer.

In *Proceedings of the RoboCup European Workshop 2000*, Amsterdam, Holland.

Chandy, K. and Misra, J. (1979).

Distributed simulation: a case study in design and verification of distributed programs.

*IEEE Transation on Software Engeneering*, 5(5):440–452.

Chandy, K. and Misra, J. (1988).

*Parallel Program Design: A Foundation*.

Addison-Wesley.

**Correa e Silva Fernandes, K. C. (2001).**

*Systèmes Multi-Agents Hybrides: Une Approche pour la Conception de Systèmes Complexes.*  
PhD thesis, Université Joseph Fourier- Grenoble 1.

**Cossentino, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010).**

ASPECS: an agent-oriented software process for engineering complex systems - how to design agent societies under a holonic perspective.

*Autonomous Agents and Multi-Agent Systems*, 2(2):260–304.

**Davidsson, P. (2000).**

Multi agent based simulation: Beyond social simulation.

*Multi Agent Based Simulation, LNCS series*, 1979.

**Dechter, R. and Pearl, J. (1985).**

Generalized best-first search strategies and the optimality of a\*.

*J. ACM*, 32(3):505–536.

**Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009).**

Engineering route planning algorithms.

In Lerner, J., Wagner, D., and Zweig, K., editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer Berlin Heidelberg.

**Demange, J. (2012).**

*A model of environment for the multilevel simulation: application to the simulation of crowds.*

PhD thesis, Université de Technologie de Belfort-Montbéliard, Belfort, France.

In French.

**Doniec, A., Mandiau, R., Piechowiak, S., and Espié', S. (2008).**

A behavioral multi-agent model for road traffic simulation.

*Engineering Applications of Artificial Intelligence*, 21(8):1443 – 1454.

**Edwards, M. (2003).**

A brief history of holons.

**Farenc, N., Boulic, R., and Thalmann, D. (1999).**

An informed environment dedicated to the simulation of virtual humans in urban context.

In *Proceedings of EUROGRAPHICS 99*, pages 309–318.

**Ferber, J. (1999).**

*Multiagent Systems: An Introduction to Distributed Artificial Intelligence.*

Addison-Wesley Professional.

**Ferber, J. and Gutknecht, O. (1998).**

A meta-model for the analysis and design of organizations in multi-agent systems.

In Demazeau, Y., Durfee, E., and Jennings, N., editors, *Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135, Paris, France.

**Ferber, J., Gutknecht, O., and Michel, F. (2004).**

From agents to organizations: an organizational view of multi-agent systems.

In *Agent-Oriented Software Engineering IV 4th International Workshop*, volume 2935 of *LNCS*, pages 214–230, Melbourne, Australia. Springer Verlag.

**Ferber, J. and Müller, J. (1996).**

Influences and reactions : a model of situated multiagent systems.

In *Second International Conference on Multi-Agent Systems (ICMAS 96)*, pages 72–79.

**Filloque, J.-M. (1992).**

*Synchronisation Répartie sur une machine parallèle à couche logique reconfigurable.*

PhD thesis, Institut de Formation Supérieure en informatique et communication, Université de Rennes 1, Rennes, France.

Fischer, K., Schillo, M., and Siekmann, J. (2003).

Holonic multiagent systems: A foundation for the organisation of multiagent systems.

In *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744 of *LNCS*, pages 71–80. Springer Berlin, Heidelberg.

Fujimoto, R. (1990).

Optimistic approaches to parallel discrete event simulation.

*Transactions of the Society for Computer Simulation*, 7(2):152–191.

Galland, S. (2013a).

*Méthodologie et outils pour la simulation multiagent dans des univers virtuels*.

PhD thesis, Université de Franche-Comté, Université de Technologie de Belfort-Montbéliard.

Galland, S., Balbo, F., Gaud, N., Rodriguez, S., Picard, G., and Boissier, O. (2015).

Contextualize agent interactions by combining social and physical dimensions in the environment.

In Demazeau, Y., Decker, K., De la prieta, F., and Bajo perez, J., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection. Lecture Notes in Computer Science 9086.*, pages 107–119. Springer International Publishing.

Galland, S. and Gaud, N. (2014a).

Holonic model of a virtual 3D indoor environment for crowd simulation.

In *International Workshop on Environments for Multiagent Systems (E4MAS14)*, Paris, France. IFAAMAS, Springer.

Galland, S., Gaud, N., Demange, J., and Koukam, A. (2009).

Environment model for multiagent-based simulation of 3D urban systems.

In *the 7th European Workshop on Multiagent Systems (EUMAS09)*, Ayia Napa, Cyprus.

Paper 36.

Galland, S., Gaud, N., Demange, J., and Koukam, A. (2014b).

Multilevel model of the 3D virtual environment for crowd simulation in buildings.

In *1st International Workshop on Agent-based Modeling and Simulation of Cities (AgentCities14)*, pages 822–827, Hasselt, Belgium. Elsevier.

Procedia Computer Science, vol. 32.

Galland, S., Gaud, N., Yasar, A.-U.-H., Knapen, L., Janssens, D., and Lamotte, O. (2013b).

Simulation model of carpooling with the janus multiagent platform.

In *the 2nd International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS)*, Halifax, Nova Scotia, Canada. Procedia Computer Science, Elsevier.

Galland, S., Lamotte, O., and Gaud, N. (2011).

METROB: Evaluation and simulation of public transport system.

In *the Institution of Engineering and Technology Conference on Smart and Sustainable City 2011 (ICSSC11)*, Shanghai, China. Institute Engineering and Technology and Shanghai University, Shanghai University Press.

Gaud, N. (2007).

*Holonic Multi-Agent Systems: From the analysis to the implementation. Metamodel, Methodology and Multilevel simulation.*

PhD thesis, University of Technology of Belfort-Montbéliard, Belfort, France.

Gechter, F., Conter, J.-M., Galland, S., Lamotte, O., and Koukam, A. (2012).

Virtual intelligent vehicle urban simulator: Application to vehicle platoon evaluation.

*Simulation Modelling Practice and Theory (SIMPAT)*, 24:103–114.

Gerber, C., Siekmann, J. H., and Vierke, G. (1999).

Holonic multi-agent systems.

Technical Report DFKI-RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz - GmbH, Postfach 20 80, 67608 Kaiserslautern, FRG.

Giret, A. (2005).

*ANEMONA: A Multi-Agent Method for Holonic Manufacturing Systems.*  
PhD thesis, Universidad Politécnica de Valencia, Valencia, Spain.

Grassé, P.-P. (1959).

La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp.

La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs.

*Insectes Sociaux*, 6(1):41–80.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000).

MOISE: An organizational model for multi-agent systems.

In et Sichman J., M. M., editor, *Advances in Artificial Intelligence, IBERAMIA-SBIA*, pages 156–165, Brazil.

Harel, D. (1987).

Statecharts: a visual formalism for complex systems.

*Science of Computer Programming*, pages 231–274.

Helbing, D. and Molnar, P. (1997).

Self-organization phenomena in pedestrian crowds.

*Self-organization of complex structures: from individual to collective dynamics*, pages 569–577.

Henderson-Sellers, B. and Giorgini, P., editors (2005).

*Agent-Oriented Methodologies*.

Idea Group publishing.

Hilaire, V., Koukam, A., Gruer, P., and Müller, J.-P. (2000).

Formal specification and prototyping of multi-agent systems.

In Omicini, A., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents' World*, number 1972 in Lecture Notes in Artificial Intelligence. Springer Verlag.

Holland, J. H. (1995a).

*Hidden order: how adaptation builds complexity.*

Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Holland, J. H. (1995b).

*Hidden order: how adaptation builds complexity.*

Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Hoogendoorn, S. P. and Bovy, P. H. (2001).

State-of-the-art of vehicular traffic flow modelling.

*Special Issue on Road Traffic Modelling and Control of the Journal of Systems and Control Engineering*, 215(4):283–303.

Hoogendoorn, S. P. and Bovy, P. H. (2004).

Pedestrian route-choice and activity scheduling theory and models.

*Transportation Research Part B*, 38:169–190.

Hübner, J., Sichman, J., and Boissier, O. (2002).

A model for the structural, functional, and deontic specification of organizations in multiagent systems.

In Bittencourt, G. and Ramalho, G., editors, *Advances in Artificial Intelligence: 16th Brazilian Symposium on Artificial Intelligence (SBIA)*, volume 2507 of *LNAI*, pages 118–128. Springer.

Ishii, H. and Ullmer, B. (1997).

Tangible bits: Towards seamless interfaces between people, bits and atoms.

In *Conference on Human Factors in Computing Systems (CHI'97)*, pages 234–241. ACM Press.

Jennings, N. (2000).

On agent-based software engineering.

*Artificial Intelligence*, 177(2):277–296.

Jorissen, P., Wijnants, M., and Lamotte, W. (2005).

Dynamic interactions in physically realistic collaborative virtual environments.

*IEEE Transactions on Visualization and Computer Graphics*, 11(6):649–660.

Kallmann, M. and Thalmann, D. (1998).

Modeling objects for interaction tasks.

In *9th Eurographics Workshop on Animation and Simulation (EGCAS)*, pages 73–86, Lisbon, Portugal.

Karamouzas, I., Geraerts, R., and Overmars, M. (2009).

Indicative routes for path planning and crowd simulation.

In *Proceedings of the 4th International Conference on Foundations of Digital Games, FDG '09*, pages 113–120, New York, NY, USA. ACM.

Koenig, S. and Likhachev, M. (2005).

Fast replanning for navigation in unknown terrain.

*Robotics, IEEE Transactions on*, 21(3):354–363.

Koestler, A. (1967).

*The Ghost in the Machine*.

Hutchinson.

Kriz, D. (1995).

Holonic Manufacturing Systems: Case Study of an IMS Consortium.

<http://hms.ifw.uni-hannover.de/public/Feasibil/study.htm>.

Lamarche, F. and Donikian, S. (2004).

Crowds of Virtual Humans : a New Approach for Real Time Navigation in Complex and Structured Environments.

*Computer Graphics Forum (Proc. of Eurographics 2004)*, 23(3):509–518.

**Lawson, B. and Park, S. (2000).**

Asynchronous time evolution in an artificial society mode.  
*Journal of Artificial Societies and Social Simulation*, 3(1).

**Le Moigne, J.-L. (1999).**

*La modélisation des systèmes complexes.*

DUNOD, 4ième édition.

**Maturana, F., Shen, W., and Norrie, D. (1999).**

MetaMorph: An Adaptive Agent-based Architecture for Intelligent Manufacturing.

**Michel, F. (2004).**

*Formalism, tools and methodological elements for the modeling and simulation of multi-agents systems.*

PhD thesis, LIRMM, Montpellier, France.

**Michel, F. (2007).**

The IRM4S model: the influence/reaction principle for multiagent based simulation.

In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA. ACM.

**Michon, J. (1985).**

A critical view of driver behaviour models: What do we know, what should we do?

*Human Behavior and Traffic Safety*, pages 487–525.

**Norman, D. (1991).**

Cognitive artifacts.

In Carroll, J., editor, *Designing interaction: Psychology at the human-computer interface*, pages 17–38.

Cambridge University Press, New York.

**Odell, J., Nodine, M., and Levy, R. (2005).**

A metamodel for agents, roles, and groups.

In Odell, J., Giorgini, P., and Müller, J., editors, *Agent-Oriented Software Engineering (AOSE) IV*, Lecture Notes on Computer Science. Springer.

**Odell, J. J., Parunak, H. V. D., Fleischer, M., and Brueckner, S. (2003).**

Modeling agents and their environment.

In *Proceedings of the 3rd International Conference on Agent-oriented Software Engineering III*, Lecture Notes In Computer Science, pages 16–31, Berlin, Heidelberg. Springer-Verlag.

**Paris, S., Donikian, S., and Bonnalet, N. (2007).**

Vers une architecture pour la simulation microscopique de foule.

*Revue Electronique Francophone d'Informatique Graphique*, 1(1):33–43.

**Parunak, H. (2003).**

Making swarming happen.

In *Conf. on Swarming and Network Enabled Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR)*, McLean, Virginia, USA.

**Poslad, S. (2009).**

*Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction.*

Wiley.

**Rao, M. P. G. (1995).**

BDI-agents: From theory to practice.

In *the First International Conference on Multiagent Systems (ICMAS'95)*.

**Resnick, M. (1995).**

New paradigms for computing, new paradigms for thinking.

*Computers and Exploratory Learning*.

Reynolds, C. (1999).

Steering behaviors for autonomous characters.

In *Proceedings of the Game Developers Conference*, page 763–782.

Reynolds, P. (1988).

A spectrum of options for parallel simulation.

In *Winter Simulation Conference*, pages 325–332, San Diego, USA.

Ricci, A., Viroli, M., and Omicini, A. (2005).

Environment-based coordination through coordination artifacts.

In Weyns, D., Van Dyke Parunak, H., and Michel, F., editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 190–214. Springer Berlin Heidelberg.

Rodriguez, S. (2005).

*From analysis to design of Holonic Multi-Agent Systems: A Framework, Methodological Guidelines and Applications.*

PhD thesis, Université de Technologie de Belfort-Montbéliard.

Rodriguez, S., Gaud, N., and Galland, S. (2014).

SARL: a general-purpose agent-oriented programming language.

Warsaw, Poland. IEEE Computer Society Press.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999).

*The Unified Modeling Language Reference Manual.*

Addison-Wesley.

Saunier, J., Carrascosa, C., Galland, S., and Kanmeugne, P. s. (2015).

Agent bodies: An interface between agent and environment.

*E4MAS 2014 - 10 years later, LNCS*, 9068(1):1–16.

**Shannon, R. E. (1977).**

Simulation modeling and methodology.

*SIGSIM Simul. Dig.*, 8(3):33–38.

**Shao, W. and et al. (2005).**

Environmental modeling for autonomous virtual pedestrians.

**Simon, H. A. (1996).**

*The Science of Artificial.*

MIT Press, Cambridge, Massachusetts, 3rd edition.

**Tecchia, F., Loscos, C., Conroy, R., and Chrysanthou, Y. (2001).**

Agent behaviour simulator (abs): a platform for urban behaviour development.

In *In GTEC'2001*, pages 17–21.

**Thalmann, D., Grillon, H., Maim, J., and Yersin, B. (2009).**

Challenges in crowd simulation.

In *CyberWorlds, 2009. CW '09. International Conference on*, pages 1–12.

**Thalmann, D. and Musse, S. R. (2007).**

*Crowd simulation.*

Springer.

**Thomas, G. and Donikian, S. (2000).**

Modelling virtual cities dedicated to behavioural animation.

*Computer Graphics Forum*, 19(3):71–80.

**Treiber, M., Hennecke, A., and Helbing, D. (2000).**

Congested traffic states in empirical observations and microscopic simulations.

*Phys. Rev. E*, 62:1805–1824.

Treuille, A., Cooper, S., and Popović, Z. (2006).

Continuum crowds.

*ACM Trans. Graph.*, 25(3):1160–1168.

Ulieru, M. and Geras, A. (2002).

Emergent holarchies for e-health applications: a case in glaucoma diagnosis.

In *IEEE IECON'02*, volume 4, pages 2957– 2961.

Viroli, M., Holvoet, T., Ricci, A., Schelfhout, K., and Zambonelli, F. (2007).

Infrastructures for the environment of multiagent systems.

*Autonomous Agents and Multi-Agent Systems*, 14(1):49–60.

Weiser, M. (1991).

The computer for the twenty-first century.

*Scientific American*, 265(3):94–104.

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2005).

Environment for multiagent systems state-of-the-art and research challenges.

In *Environments for Multi-Agent Systems (E4MAS)*, volume 3374, pages 1–47. Springer Berlin / Heidelberg.

Wilber, K. (1995).

*Sex, Ecology, Spirituality*.

Shambhala.

Wilensky, U. and Rand, B. (2015).

*Introduction to Agent-Based Modeling: Modeling Natural, Social and Engineered Complex Systems with NetLogo*.

The MIT Press.

Willemsen, P., Kearney, J., and Wang, H. (2003).

Ribbon networks for modeling navigable paths of autonomous agents in virtual urban environments.  
In *Virtual Reality, 2003. Proceedings. IEEE*, pages 79–86.

Wooldridge, M. and Ciancarini, P. (2001).

Agent-oriented software engineering: The state of the art.

In *Agent-Oriented Software Engineering: First International Workshop (AOSE 2000)*, volume 1957 of *Lecture Notes in Computer Science*, pages 1–28. Springer-Verlag.

Wyns, J. (1999).

*Reference architecture for Holonic Manufacturing Systems - the key to support evolution and reconfiguration.*  
PhD thesis, Katholieke Universiteit Leuven.

Zambonelli, F., Jennings, N., and Wooldridge, M. (2003).

Developing multiagent systems: the GAIA methodology.

*ACM Trans. on Software Engineering and Methodology*, 12(3).

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000).

*Theory of Modeling and Simulation.*

Academic Press, 2nd edition edition.



# Bibliography (#17)

xxxii



# Bibliography (#18)

xxxiii