



MultiAgent Systems

Faculté des Sciences Économiques et de Gestion - Sfax - Tunisie - 2016

Stéphane GALLAND



- A Chapter 1: Introduction**
- B Chapter 2: Multiagent-based Simulation**
- C Chapter 3: Agent-Oriented Software Engineering**
- D Chapter 4: Action Selection**
- E Chapter 5: Rational agents**
- F Chapter 6: Planification**



Chapter 1: Introduction

Stéphane GALLAND



- 1 Trends in computer science
- 2 Agent
- 3 Introduction to Multiagent systems
- 4 Hierarchical Multiagent Systems
- 5 A first summary
- 6 The SARC Agent Programming Language
- 7 Examples



Section 1

Trends in computer science

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.

Programming has progressed through:

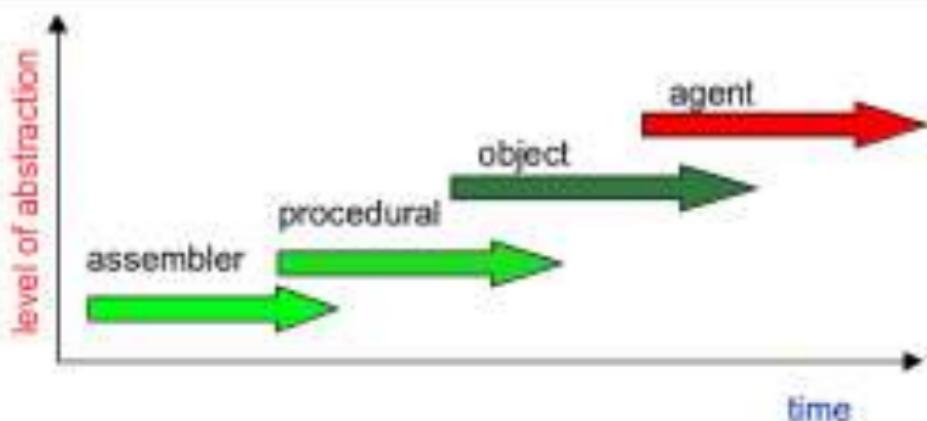
- machine code;
- assembly language;
- machine-independent programming languages;
- sub-routines;
- procedures & functions;
- abstract data types;
- objects (and actors);

to

- agents.

Agent: a new paradigm ?

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



Section 2

Agent



Multiagent systems: a new view ? Which characteristics ?

- Multiagent-based approach (metaphor or paradigm) is represents a new way of analyzing, designing and implementing software systems, especially complex systems
- It strongly improves/impacts the way in which people conceptualizes and implements a large number of systems.
- Strong interdisciplinary inspiration: social and biological sciences, Economics and Game theory, control theory.
- Large panel of application

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

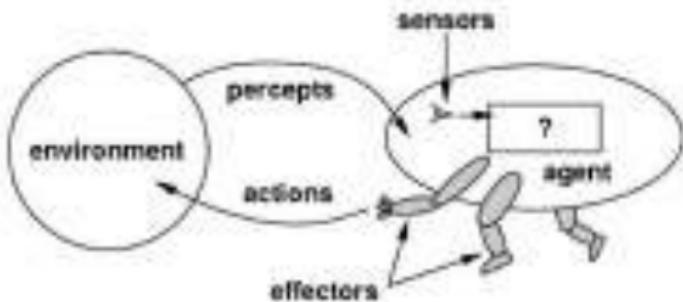
Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative; They pursue their own personal or collective goals.

- 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.



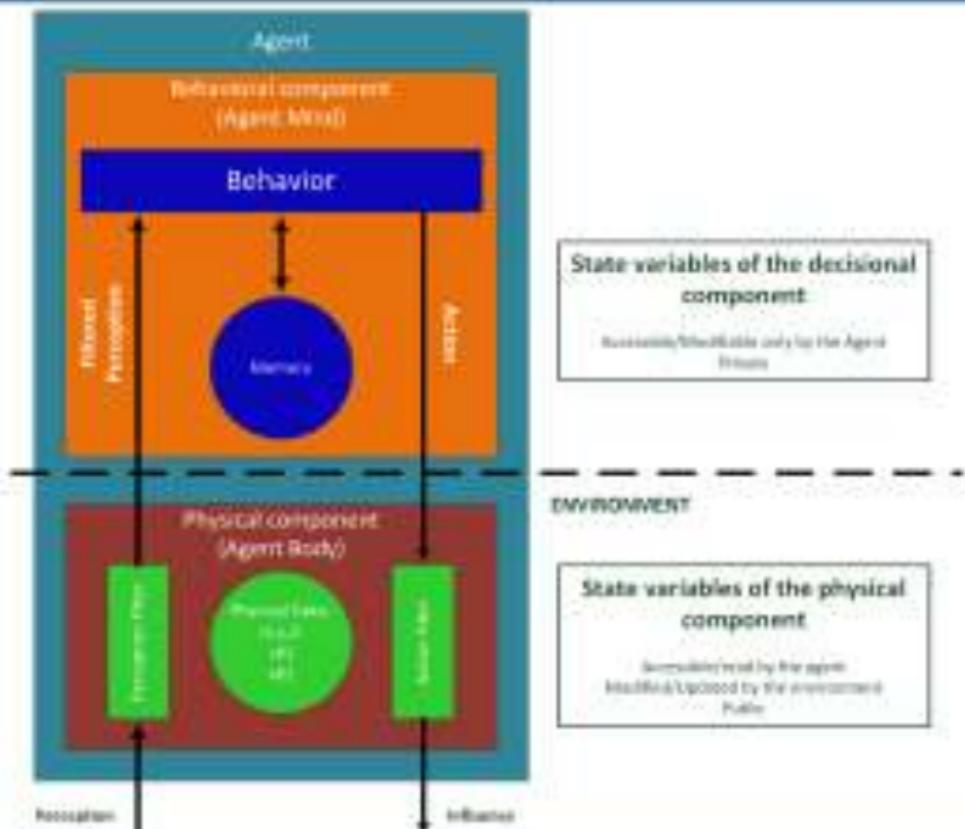
Agent

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



Discussions

- What contains/includes the environment ?
- Are all agent situated ?
- What/Where is located the body of the agent ?





- The agent behavior is strongly affected by the environment in which it is embedded!
- **⚠ Attention:** Difference between body and mind of the agent

- **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.

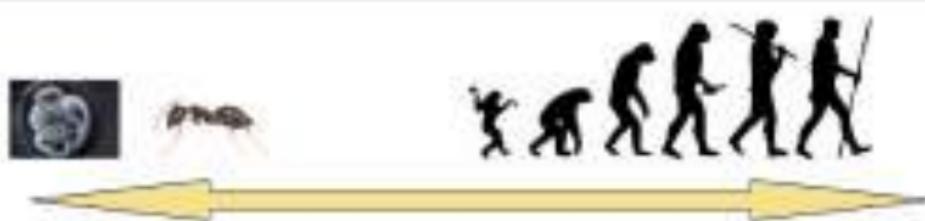
Agent [Ferber, 1999]

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.

Two main kinds of Agents

- **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.



Reactive Agent

Hybrid Agent

Deliberative/Cognitive Agent



Section 3

Introduction to Multiagent systems

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.

⇒ Direct (Agent-Agent) and Indirect (Environment, Stigmergy) Interactions



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



Closed Systems

The set of entities composing the system is fixed and (usually) known *a priori*.

Open Systems

The set of entities is dynamic, new entities join/leave the system during its execution. (e.g. internet)

Homogeneous Systems

All entities have the same architecture, are built from the same model.

Heterogeneous Systems

Different types, Different architectures, built from different models.



Characteristics/Features of Complex Systems

- Emergence
- Local Interactions
- Relationships are non-linear
- Feedback Loops (Butterfly effect)
- The components can themselves be considered as Complex Systems
- A great number of entities interact
- These interactions generate "emergent" behaviors
- Different levels of abstraction can be distinguished

Agent vs Object [Wooldridge, 2001]

- 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.
- MAS are inherently "multi-thread". Each agent has its own thread of execution.

Key Differences

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 user.
- 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).



Section 4

Hierarchical Multiagent Systems



Hierarchical Multiagent Systems

Agents are not necessarily atomic.

The idea of "Agent composed of agent" is recurrent in the scientific community:

- Collective Agents [Ferber, 1999]
- Meta-Agents [Holland, 1995]
- Recursive agents [Correa e Silva Fernandes, 2001]
- Agentified-Groups [Odell, 2005]

Two different kinds of hierarchy

Hierarchical MAS are increasingly used.

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

Interests

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

Characteristics

- Complexity often takes the form of a hierarchy [Simon, 1996].
- In most systems, it is somewhat arbitrary as to where the partitioning is left off and what subsystems are taken as elementary [Simon, 1996].
- Components themselves complex: atomic at a given level of observation, an entire system at a finer level.
- Distinction between interactions intra-and inter-subsystems.
- Emergence, Feed-back loop, non linear interactions
non-linéaires...



MAS 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.

→ MAS are considered well suited for modeling and simulating complex systems.

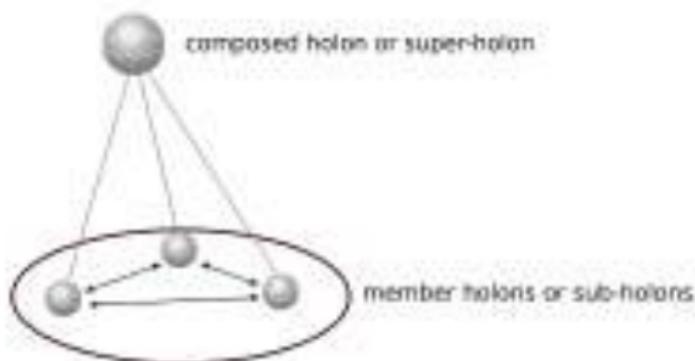
Pb with most multi-agent models

- Most approaches consider as atomic agents (non composed, one single level).
- Difficulties in integrating different levels of abstraction within a model.

→ By considering agents as composed, Hierarchical MAS (including Holonic) represent a possible solution.

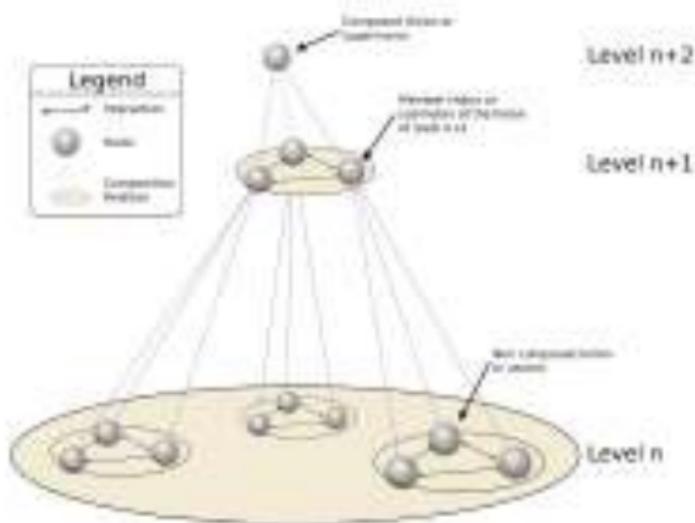
Holon [Koestler, 1967]

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

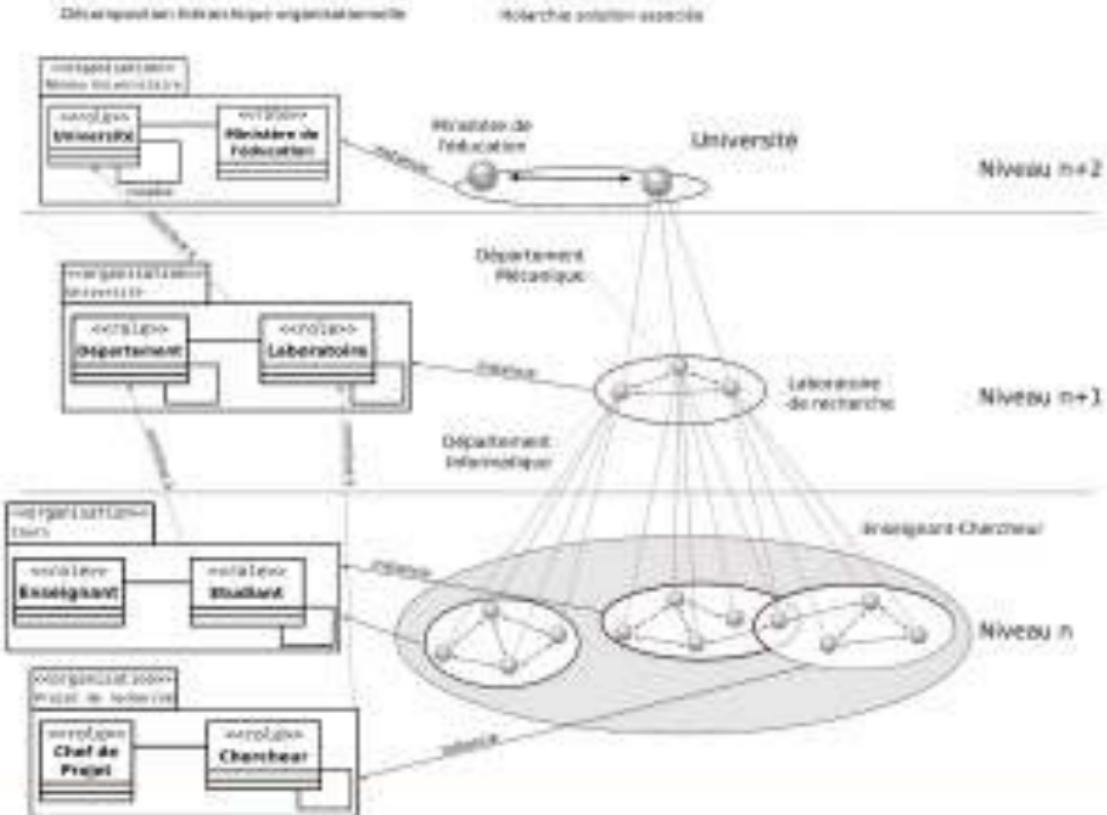


Hierarchy [Koestler, 1967]

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



Hierarchy: An example



Brief History

- 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, 1999b]

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



The concept of holon allows us to:

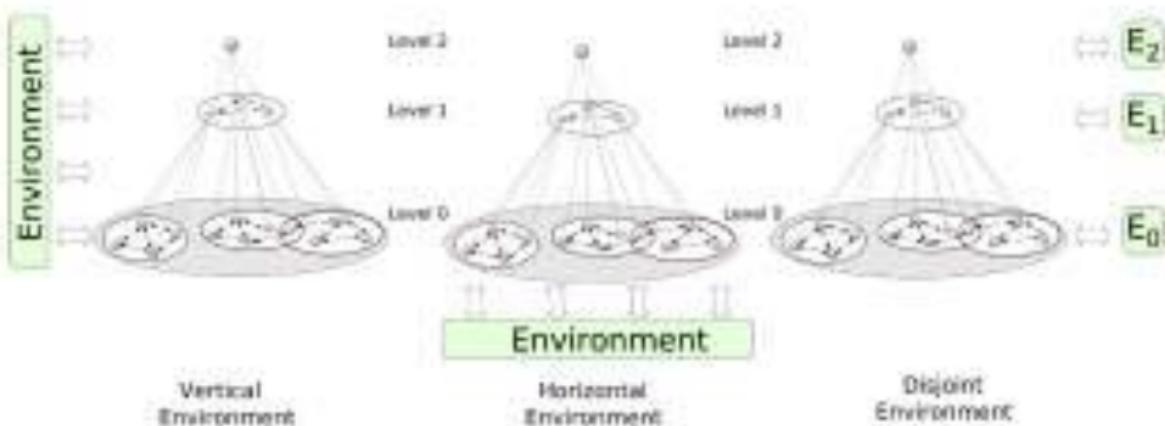
- to view a collection of interacting entities as a single higher-level holon.
- to decompose a entity seen as atomic at a certain level as a set of interacting entities at a lower level.

Main advantages

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

Challenges

- What does it mean that an entity to be an Agent and a group of Agent 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?
- etc.





Section 5

A first summary



Agent Design

- How to design agents capable of performing the tasks delegated to them ? Dynamic Agent capacities/services ?
- Normally, an agent has a predetermined architecture with a predetermined set of abilities.

Society Design

- How to create agents capable of interacting to perform collective/individual tasks? Cooperation, Coordination, Negotiation, etc.
- How to guarantee such a system will converge ? System Stability ? Performances ?
- And if the other agents do not share the same goals / tasks ? Concurrency, Competition, Utility ?
- Should we Trust these other agents ? Malware, open systems (new agents incoming)

Two levels : Micro: Agent (Individual), Macro: MAS (Society)



The MAS Vision

- MAS such as a Software Engineering paradigm (AOSE)
 - Improved our understanding of the characteristics of software complexity.
 - It is now recognized that **interaction** is probably the most important characteristic of complex software.
- MAS as a tool for understanding human societies
- MAS as a tool for analyzing/modeling/simulating Complex Systems



Section 6

The SARL Agent Programming Language

SARL Design Principles

- All agents are holonic (recursive agents).
- There is not only one way of interacting but infinite.
- Event-driven interactions as the default interaction mode.
- Clear separation between Language and Platform related aspects.
- Everything is distributed and it should be transparent.
- Massively parallel.
- Platform- and architecture-independent.
- Coding should be fun (Ruby/Scala-like) ☺.



Expectations for SAML

- Stop implementing Agents with Object-Oriented concepts.
- Playground to find minimal Agent-Oriented Programming concepts.
- Agents should be simple to extend.
- Provide the community a common discussion forum. (a testbed)

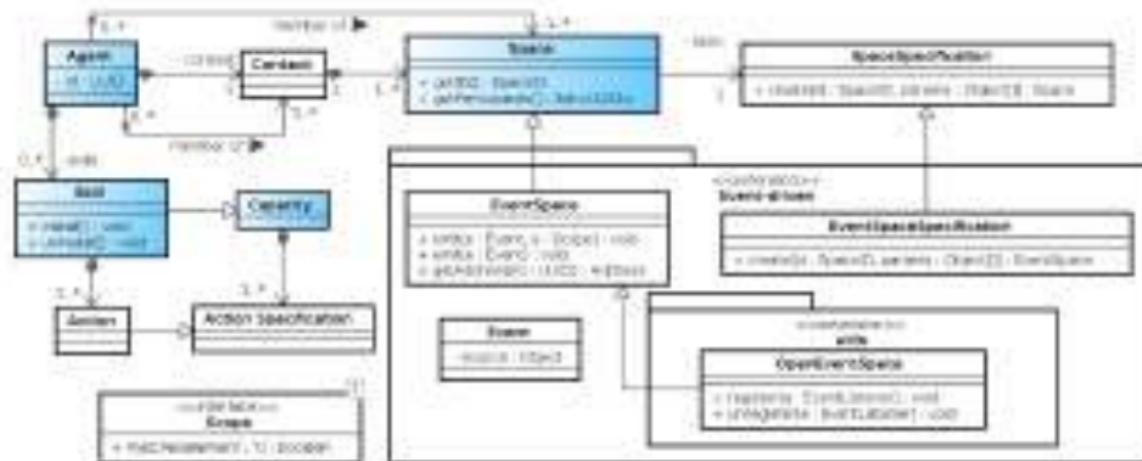
Subsection 1

Programming MAS in SARL



a MAS in SARL

A collection of Agents interacting together in a collection of shared distributed Spaces.



4 main concepts:

- Agent
- Capacity
- Skill
- Space

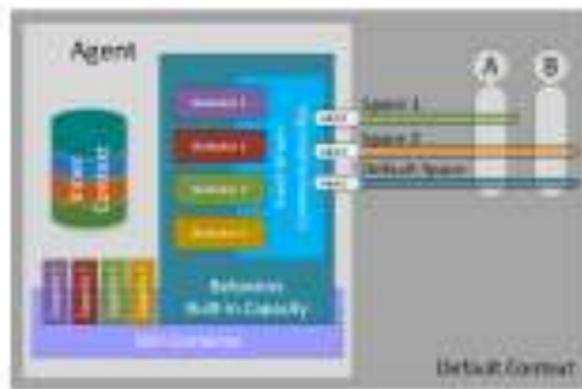
3 main dimensions:

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



Agent

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



```
agent HelloAgent {  
    uses Lifecycle, Schedules  
    on Initialize {  
        println("Hello World!")  
        in(2000) | killme |  
    }  
    or Destroy {  
        println("Goodbye World!")  
    }  
}
```



Capacity

Specification of a collection of actions.

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.

constraint
Capacity C1

+ action1
+ action2

SARL

constraint System
+ action3
+ action4

Skill

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



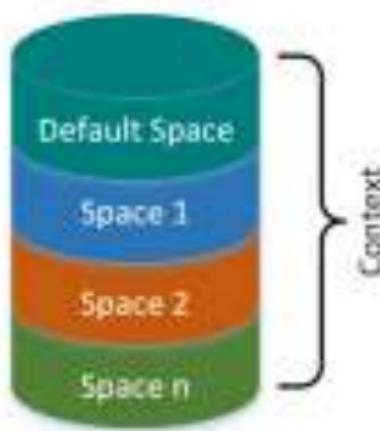
```
capacity Logging {  
    def debug(s : String)  
    def info(s : String)  
}  
  
skill! BasicConsoleLogging  
implements Logging {  
    def debug(s : String) {  
        println("DEBUG:" + s)  
    }  
  
    def info(s : String) {  
        println("INFO:" + s)  
    }  
}
```

```
agent HelloAgent {  
    uses Lifecycle, Schedules, Logging  
  
    on Initialize {  
        setSkill! Logging,  
            new BasicConsoleLogging()  
        info("Hello World!")  
        in(2000) | killMe |  
    }  
  
    on Destroy {  
        info("Goodbye World!")  
    }  
}
```



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.



Space

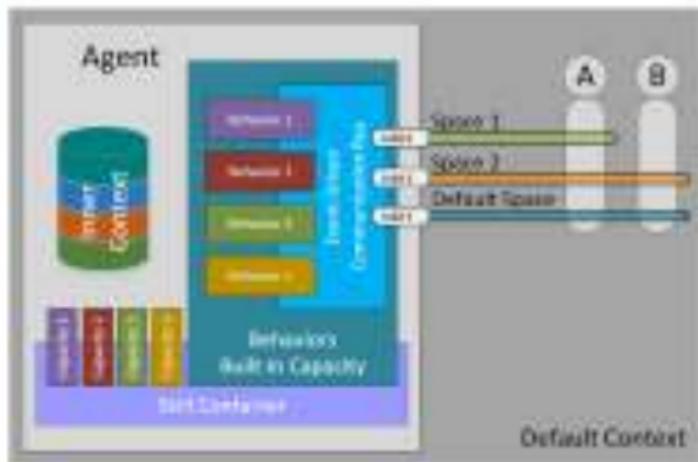
Support of interaction between agents respecting the rules defined in various Space Specifications.

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.

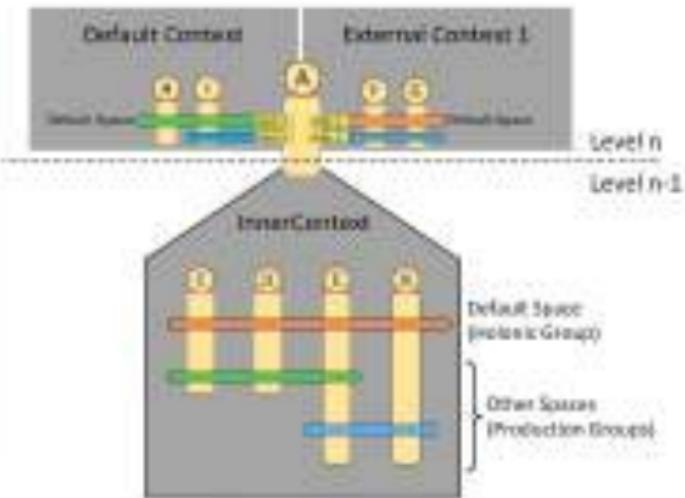
Default Space: an Event Space

- 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.



Contexts and Holonic properties

- All agents have at least one External Context (the default one).
- All agents participate in the Default Space of all Contexts they belong to.
- The Janus Context is omnipresent.



Subsection 2

Built-in Capacities

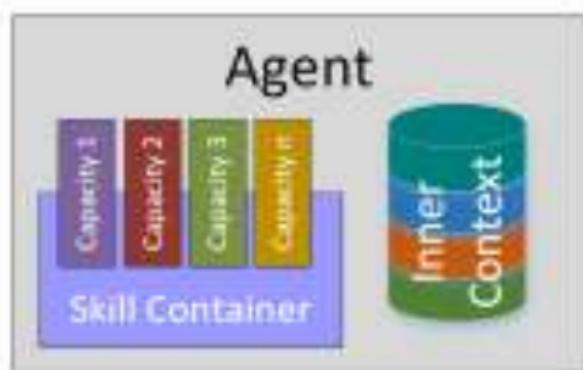
• [Index](#)

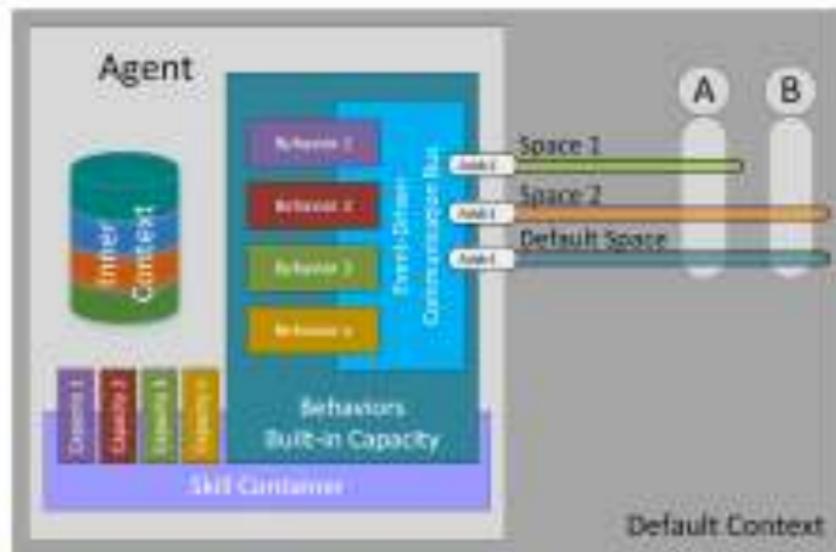


A SART Agent has inherently a set of **Built-in Capacities**

Current Built-in Capacities

- ExternalContextAccess
- InnerContextAccess
- Behaviors
- Lifecycle
- Schedules
- DefaultContextInteractions





Behavior

Defines the actions to be performed on a given perception (Events) in a Space.

Section 7

Examples

Subsection 1

A Ping - Pong in SARL

Specification – Principle

- The Ping agent is sending a Ping message to all agents.
- The Pong agent is receiving the Ping message, and replies with a Pong message to the sender of the Ping message.
- The Ping agent is receiving a Pong message and replies to the sender of the Pong with a new Ping message.

These messages contains an integer number that indicates the number of the event.

All the messages are exchanged into a new space of the default context.

```
package io.sarl.demos.pingpong.subspace
import io.sarl.core.Behaviors
import io.sarl.core.DefaultContextInteractions
import io.sarl.core.ExternalContextActions
import io.sarl.core.Initialize
import io.sarl.core.Lifecycle
import io.sarl.core.Schedules
import io.sarl.demos.pingpong.PingPongConstants
import io.sarl.util.SpecEventSpace

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

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



```
agent SubspacePingAgent[  
    uses Lifecycle, Schedules, DefaultContextInteractions, ExternalContextAccess,  
    Behaviors  
    var count : Integer  
    var space : OpenEventSpace  
    /* Print "Hello World" when spawned and wait 2 seconds are still itself */  
on Initialize {  
    println("SHARED_SPACE "+PingPongConstants.SHARED_SPACE)  
    space = defaultContext.getOrCreateSpace(OpenEventSpaceSpecification,  
        PingPongConstants.SHARED_SPACE)  
    space.register(asEventListener{})  
    println("Starting waiting for partner")  
    count = 0  
    every{1000} {  
        if (space.participants.size()>1) {  
            println("Send Ping "+count)  
            var evt = new Ping(count)  
            evt.source = space.getAddress(getID())  
            space.emit(evt)  
            count = count + 1  
        } else {  
            println("I am alone")  
        }  
    }  
}  
on Pong {  
    println("Recv Pong "+occurrence.value)  
}
```

```
agent SubspacePongAgent {  
    uses Lifesycles, DefaultContextInteractions, Behaviors  
  
    var space : OpenEventSpace  
  
    on Initialize {  
        println("SHARED_SPACE=" + PingPongConstants.SHARED_SPACE)  
        space = defaultContext.getOrCreateSpace(OpenEventSpaceSpecification,  
            PingPongConstants.SHARED_SPACE)  
        space.register(asEventListener())  
        println("Waiting for pings")  
    }  
  
    on Ping {  
        println("Recv Ping: " + occurrence.value)  
        println("Send Pong: " + occurrence.value)  
        var evt = new Pong(occurrence.value)  
        evt.source = space.getAddress(getID());  
        space.emit(evt)  
    }  
}
```

Subsection 2

Vacuum Agent



Vacuum World

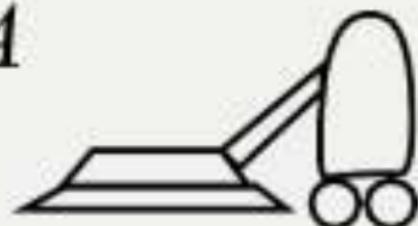
Jeremy Morris (Ohio State University)

<http://www.csse.ohio-state.edu/~bmorris/csc583/Teaching/CSC583/Week01-IntelligentAgents.pdf>

- Two locations: A and B
- Some locations have dirt
- Vacuum agent in one place at a time



A



B



Vacuum Agent

- Sensors: Location, Dirt?
- Percepts: InA/InB, Dirty/NotDirty
- Effectors: Wheels, Vacuum
- Actions: MoveL, MoveR, Suck, NoOp

Vacuum Agent Design

A good agent should be autonomous

- Actions depend on percepts
- Actions depend on conclusions drawn from percepts
- Actions should never be predetermined

Vacuum Agent Design: the most basic version

A simple thread/runnable with the following *run()* method:

```
var done = false
var p  Percept
var a  Action
while (!done) {
    p = getPercept()
    a = chooseAction(p)
    doAction(a)
    done = checkIfDone()
}
die()
```

Designing *chooseAction* method

Using a lookup table

Percept	Action
[A, clean]	MoveRight
[A, dirty]	Suck
[B, clean]	MoveLeft
[B, dirty]	Suck

Does my agent perform well?

- Need a performance measure
 - Maximize amount of dirt picked up?
 - Maximize ratio of dirt to energy expended?
 - Maximize ratio of dirt to combination of energy and time?
- Selecting a performance measure is not always easy

Subsection 3

Foraging bots



http://www.janus-project.org/Forager_Bots



4 rules:

- "Explore" rule: if I'm carrying anything and I do not perceive mineral then I explore randomly
- "Find" rule: if I'm carrying anything and I perceive mineral then I take some mineral as much as I can.
- "Bring back" rule: if I'm carrying mineral and I'm not at home then I return home
- "Drop" rule: if I'm carrying mineral and I'm at home then I drop my mineral

Coordinated/Collaborative behaviors using (Pheromone)

- "Explore" rule: if I'm carrying anything and I do not perceive mineral and I do not perceive any pheromone then I explore randomly
- "Bring back" rule: if I'm carrying mineral and I'm not at home then I return home and I drop a pheromone
- "Follow pheromone" rule: if I'm carrying anything and I do not perceive mineral and I perceive a pheromone then I head to the pheromone.

- 1 Present the basics of the simulation.
- 2 Detail the principles of the multiagent-based simulation.
- 3 Introduction to organizational modelling.
- 4 Illustration on:
 - Carpooling simulation,
 - Highway simulation,
 - Pedestrian simulation,
 - Intelligent Vehicle.



SARL Agent Programming Language
www.sarl.io

INTUITIVE SYNTAX. Develop faster.

EXTENSIBLE AGENT FEATURES. Keep it DRY.

TRUE HOLONIC AGENTS. The whole is greater than the sum of its parts.

FULLY DISTRIBUTED MAS. Janus Platform for SARL

SIMULATION ENVIRONMENT. Move "turtles" with the Jaak extension.

- 1 Simulation Fundamentals
- 2 Multiagent Based Simulation (MABS)
- 3 Examples of Multiagent-based Simulations with SARL



Section 1

Simulation Fundamentals

Module 1

(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.

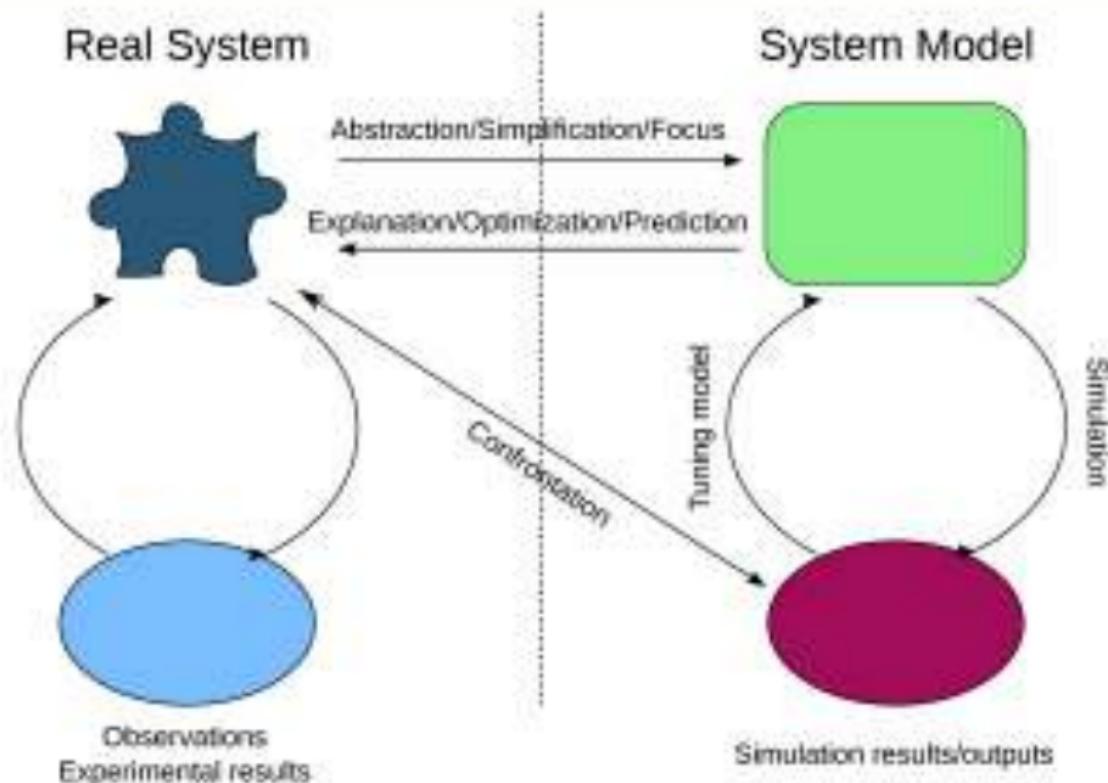
[Fishwick, 1997]

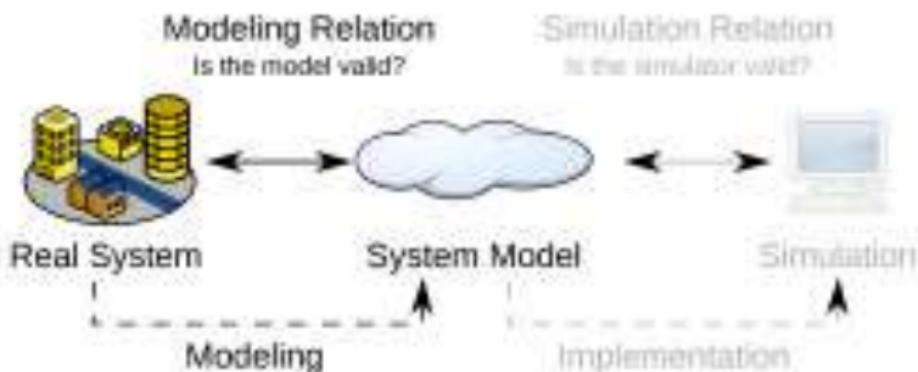
Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output.

Three fundamental tasks in all kinds of simulation

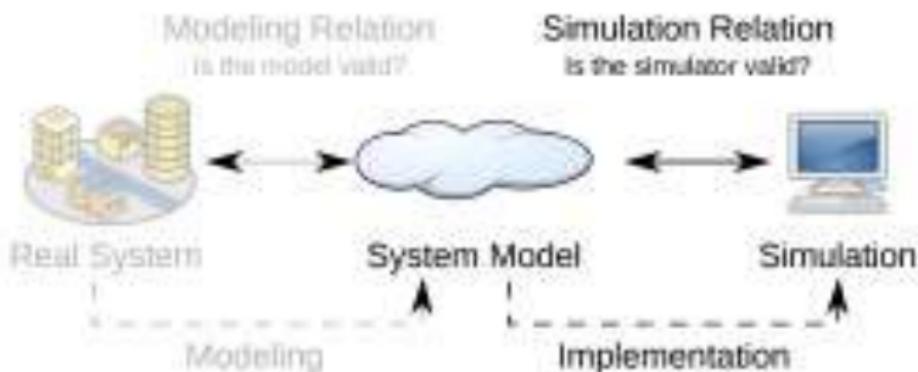
- Model design.
- Model running.
- Result analysis.

- 1 Requirement analysis: defining the objectives addressed by the model,
 - 2 Model conceptualization and specification,
 - 3 Implementation,
 - 4 Calibration and bug fixing,
 - 5 Experiments,
 - 6 Analysis of results.
- Parallel activity: model validation (against the real system).
- Model verification (end of each activity).





- 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.

A dynamic system

- See the systemic theory and the general system theory.
- Two fundamental aspects in a dynamic system:
 - ① External behavior of the system (at its bounds): the observable reactions of the system from outside it.
 - ② Internal structure and behavior of the system: its internal state and its inner dynamics (state-transition function).



- Primarily defined according to the way it evolves over time.
- One of the most important characteristics of a model:

How the passage of time is represented?

- 3 main approaches:

- Continuous model:** state variables evolve continuously.
Differential equation system specification.

- Continuous model: state variables evolve continuously following a constant speed \dot{x} (the derivative)

$$x(t + \Delta t) = x(t) + \dot{x}(t)\Delta t$$

With $x(t)$ the state of the system at t , and \dot{x} the transition function.

- Event-based model: does not follow similar dynamics of specific discrete transitions by events

- Primarily defined according to the way it evolves over time.
- One of the most important characteristics of a model:

How the passage of time is represented?

- 3 main approaches:
 - Continuous model:** state variables evolve continuously.
Differential equation system specification.
 - Discrete model:** time axis is discretized following a constant period Δt (the time step)

$$\sigma(t + \Delta t) = \Phi(\sigma(t))$$

With $\sigma(t)$ the state of the system at t , and Φ the transition function.

• Evolution of model from one time-step to the next
↳ discrete time-stepped by Δt

- Primarily defined according to the way it evolves over time.
- One of the most important characteristics of a model:

How the passage of time is represented?

- 3 main approaches:
 - Continuous model:** state variables evolve continuously.
Differential equation system specification.
 - Discrete model:** time axis is discretized following a constant period Δt (the time step)

$$\sigma(t + \Delta t) = \Phi(\sigma(t))$$

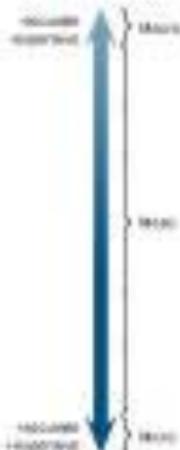
With $\sigma(t)$ the state of the system at t , and Φ the transition function.

- Event-based model:** state variables evolve discretely at specific instants represented by events.

Topology according to the **granularity of the simulation**, the level of details that is possible 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.



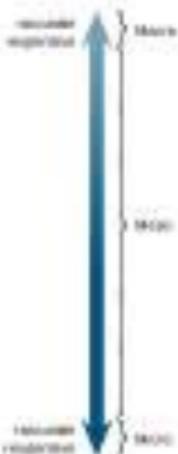
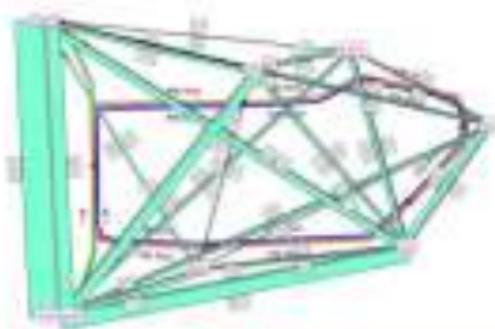
Mesoscopic Simulation

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



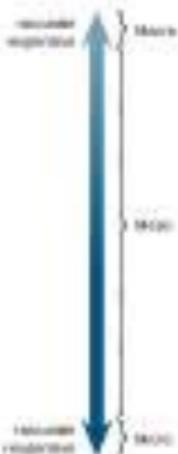
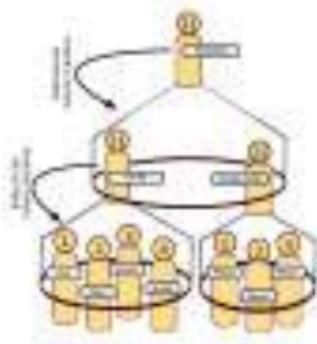
Macroscopic Simulation

- 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.



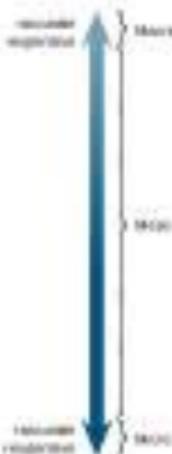
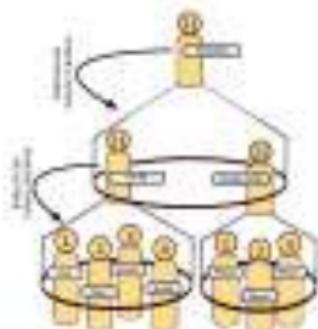
Multilevel Simulation

- 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 traditionally considered as a special form of microscopic simulation, but not restricted to.

- Usually are: equation-based model, differential equations
⇒ well understood, established mathematical framework.
- Formulas are concise and form a complete model.
- Proven success in several domains.
- Usually, the volume and the accuracy of data required for the initialization are much smaller than for other approaches.
- Usually, lighter in terms of computations.
- Easier to simulate large scale systems.





- Difficult to switch from micro to macro level (transition).
- Difficult (or impossible) to represent certain behaviors, e.g., predation, mating rituals, acquisition of food...
- Does not represent behaviors but the results/outputs of the behaviors (aggregated data: number of descendants, food intake quantity...)
- Not appropriate for certain kinds of systems:
 - Systems that draw their dynamics from flexible local interactions;
 - Social systems, social hierarchies;
 - Emergent phenomena and self-organizing systems: biological systems, traffic systems;
 - Multi-level systems;
 - Intelligent human behavior.



Section 2

Multiagent Based Simulation (MABS)



- 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, resources) 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.



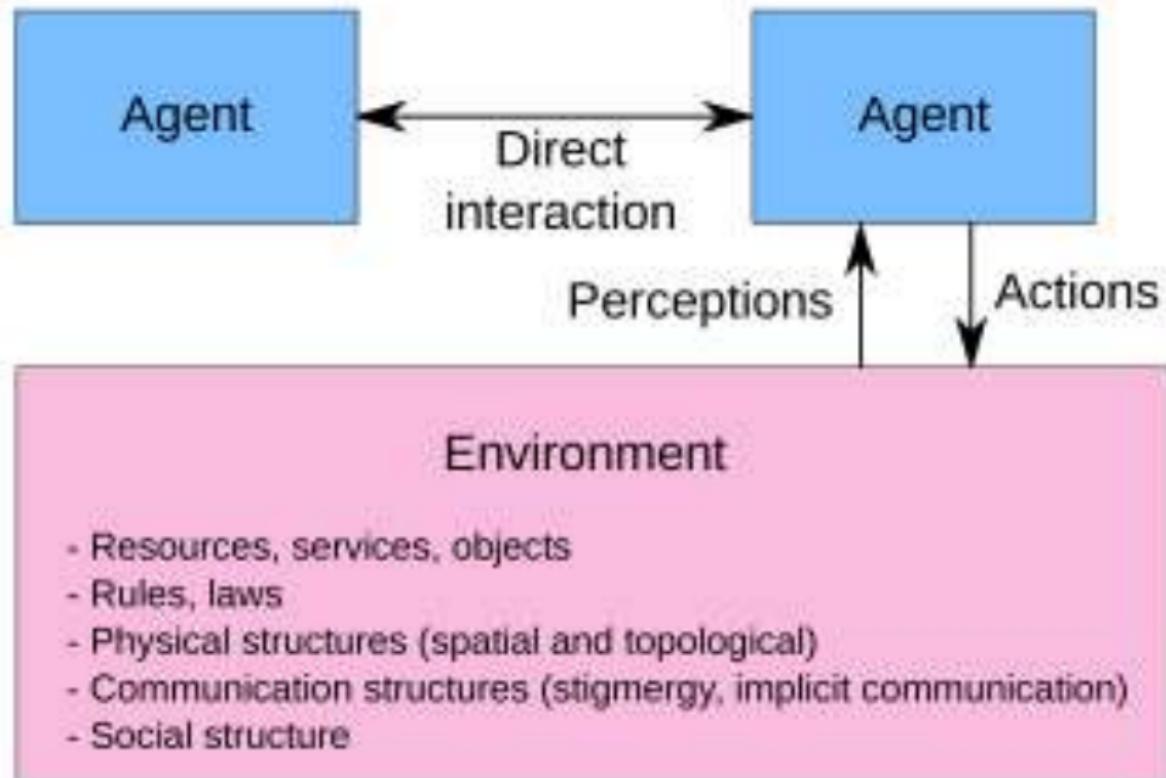
- Offer a significant level of accuracy at the expense of a larger computational cost.
- Require many and accurate data for their initialization.
- It is difficult to apply to large scale systems.
- Actual simulation models are costly in time and effort.

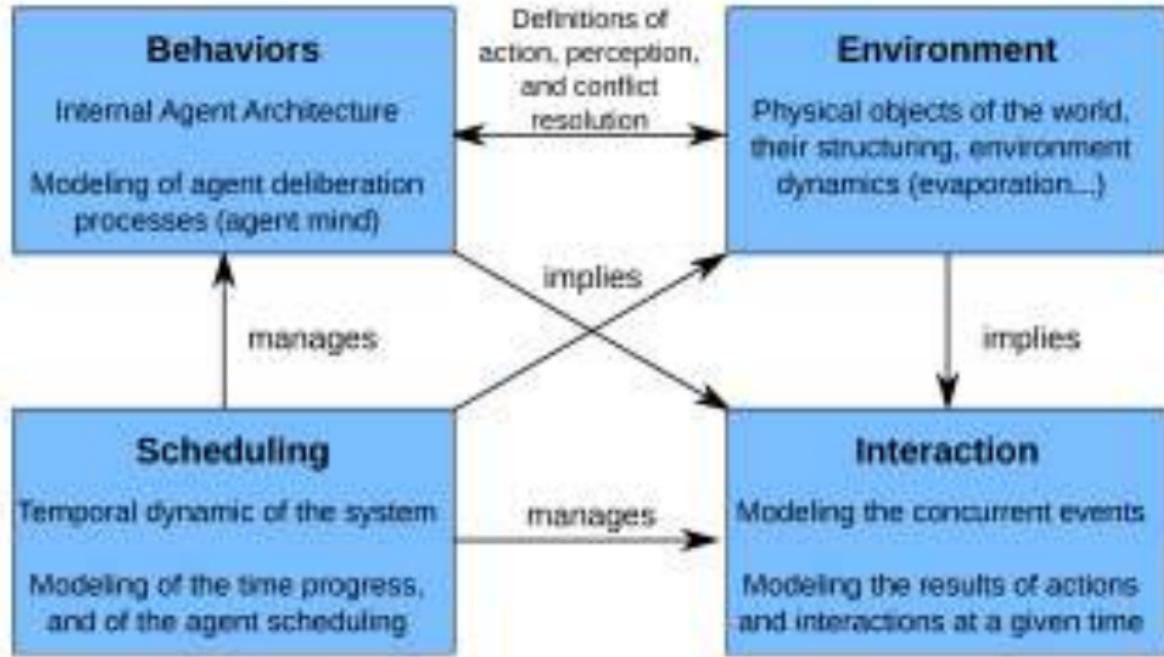




Subsection 1

Overview of a MABS Architecture

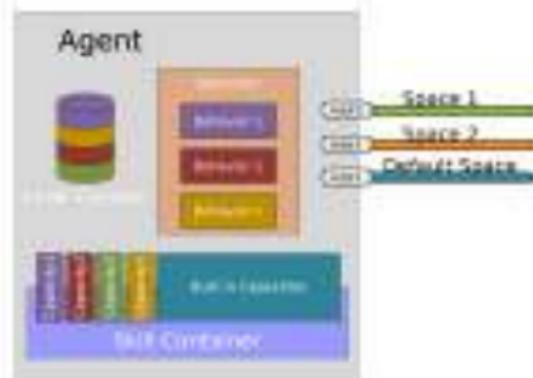






Agent

- 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 {
```

Define the code of all
the agents of type
HelloAgent

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

```
    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 {

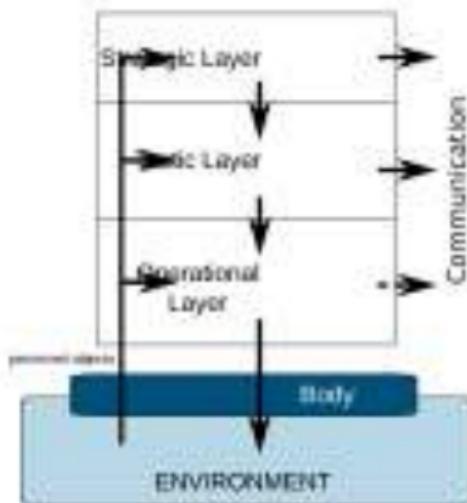
    var myvariable : int
    val myconstant = "abc"

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

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

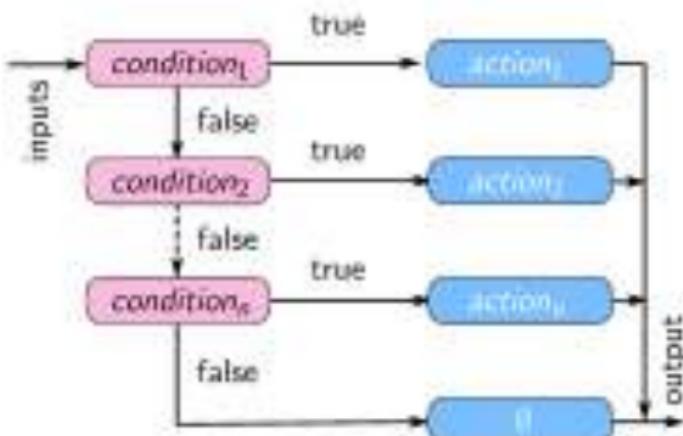
Events predefined in the SARC language:
- When initializing the agent
- When destroying the agent

- **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.
- Mostly contributes to the lower level in the three-layer architecture.

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



See Chapter 4: Action Selection, for details.

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

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



separation

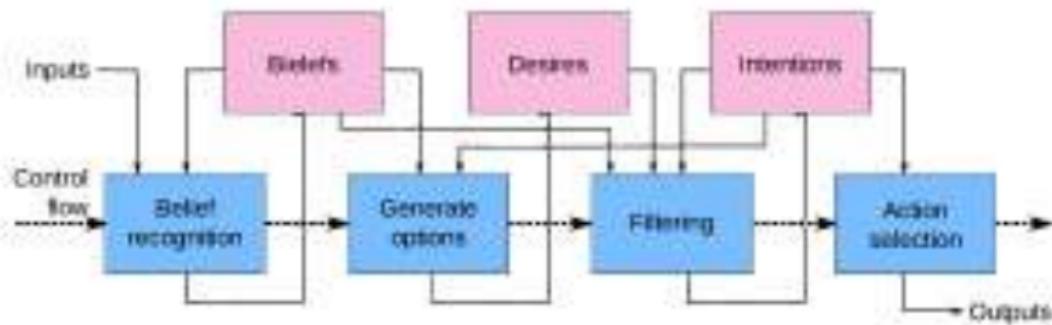


alignment



cohesion

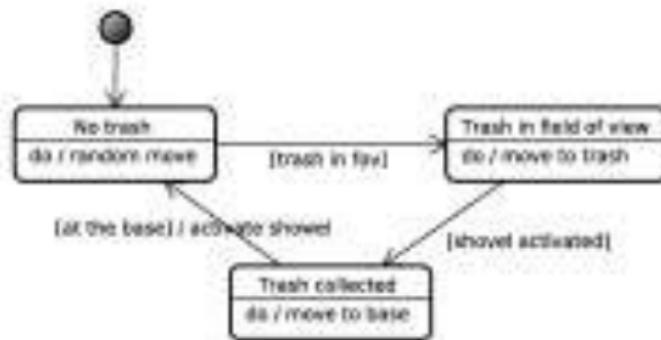
- **Beliefs:** the informational state of the agent, in other words its beliefs about the world (including itself and other agents).
- **Goals:** a desire that has been adopted for active pursuit by the agent.
- **Intentions:** the deliberative state of the agent – what the agent has chosen to do.
-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



See Chapter 4: Action Selection, for details.

- 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()
    }
}
}
```



Subsection 2

Interaction

Interactions between variables



Understanding together

- Agents may use the same concepts for understanding together.
- **Ontology:** specification of the knowledge in a specific domain.
- **Agent Communication Language:** specification of the communication elements with language acts.

- Specification of interaction protocols
- Blackboard functionality
- Actions, and negotiations

?

See 77, for details.



Understanding together

- Agents may use the same concepts for understanding together.
- **Ontology:** specification of the unknownledge in a specific domain.
- **Agent Communication Language:** specification of the communication elements with language acts.

Managing interaction

- Specification of interaction protocols,
- Blackboard Architecture,
- Auction, and negotiation,
- ...

See 77. for details.



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.
- Agents can interact with it via Capacities and Spaces.

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: Environment is a shared structure for agents, where each of them perceives and acts.

⇒ M1 is a mission of environment, which is related to the management of agents' perceptions and joint actions and to the preservation of the environmental integrity influence reaction model.

[Ferber, 1998; Michel, 2004; Gattorci, 2009]

- M2 - Managing perception and cognition: Agents can manage the access to environmental informations and guarantee the genuineness and localness of perceptions.

⇒ M2 is a mission of environment. This mission is to guarantee that agents can have genuine perceptions of their environment.

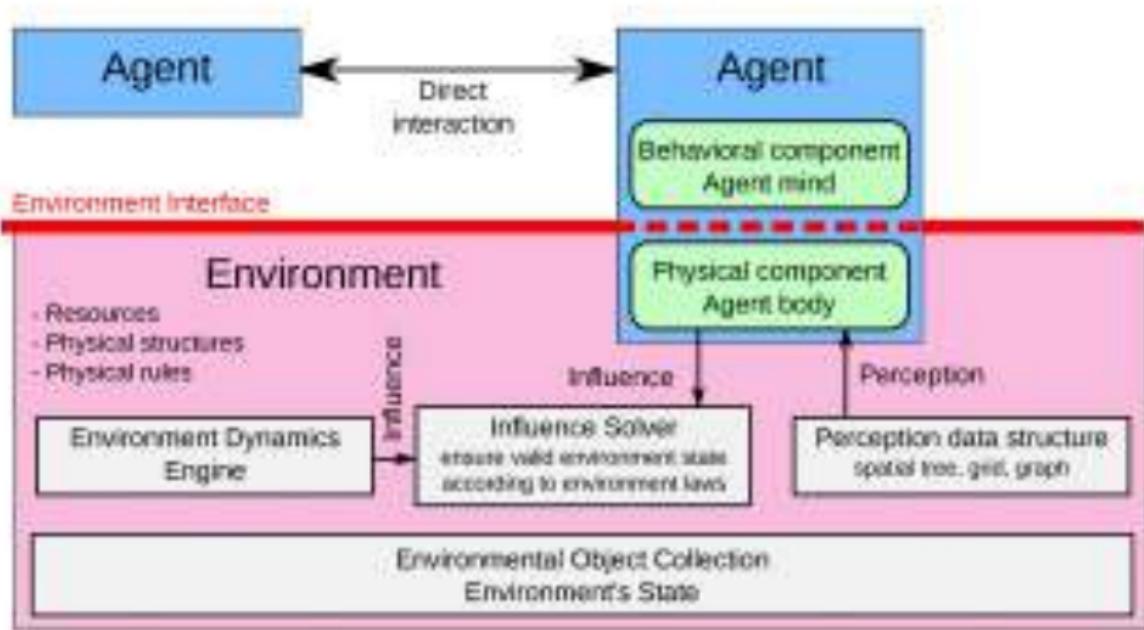


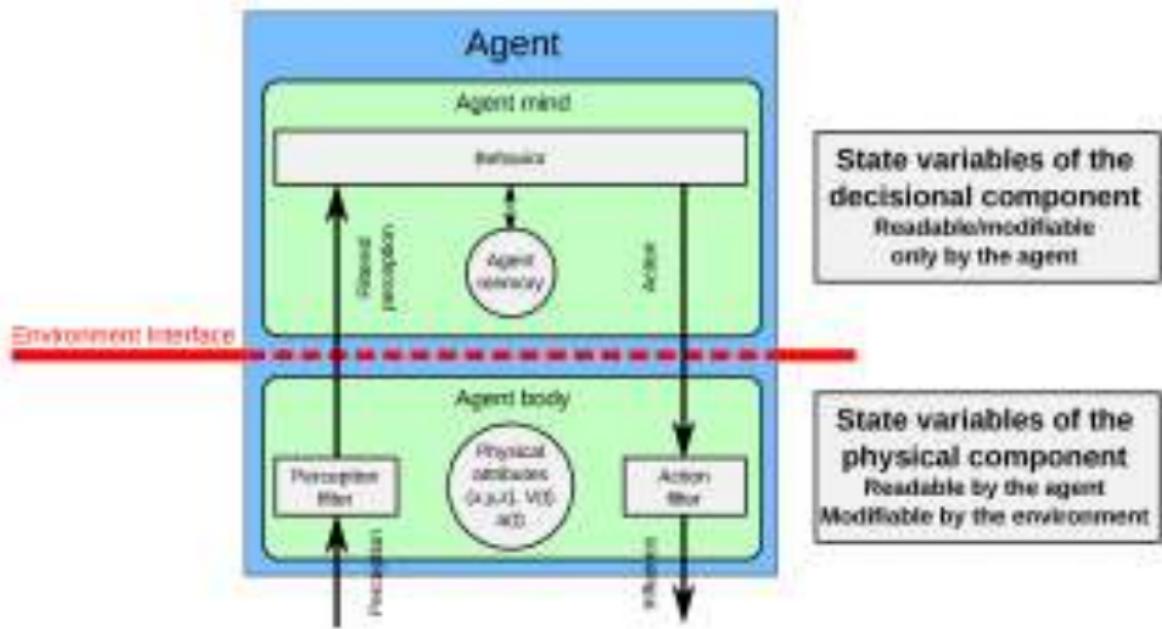
- M1 - Sharing informations: Environment is a shared structure for agents, where each of them perceives and acts.
- M2 - Managing agents actions and interactions: It is related to the management of agents' simultaneous and joint actions and to the preservation of the environmental integrity:
influence-reaction model
[Ferber, 1996, Michel, 2004, Galland, 2009].
- M3 - Managing perception and orientation: Agents can manage the access to environmental informations and guarantee the guaranteed and localness of perceptions.
- M4 - Managing entanglement dynamics: The mission is to act on supply. It can have other missions, independently of the ones of the agents.

- M1 - Sharing informations: 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.

m3.1 Managing perception and observation: The environment is a shared structure. It can have different perceptions, independently of the ones of the agents.

- M1 - Sharing informations: 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.





- It is required to define a data structure, which allows to efficiently access to the smallest set of objects concerned by a process.
- Typical environmental data structures:
 - perception structure dedicated to the static objects,
 - perception structure dedicated to the mobile objects,
 - ground model for "keep on floor," "traversability, ..."
- Some classical structures are:
 - a) Graph (*1D*). Perception and actions restricted to the graph edges.

Graph (1D)

Perception and actions

restricted to the graph edges

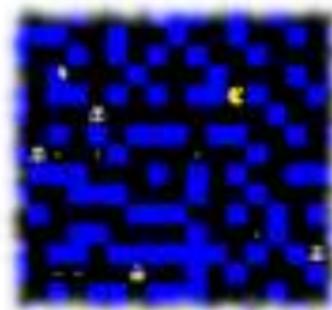


- It is required to define a data structure, which allows to efficiently access to the smallest set of objects concerned by a process.
- Typical environmental data structures:
 - perception structure dedicated to the static objects,
 - perception structure dedicated to the mobile objects,
 - ground model for "keep on floor," "traversability, ..."
- Some classical structures are:
 - a) Graph (1D).
 - b) Grid (2D, 3D). Perception and actions restricted to the cells.

Graph (1D)

b) Grid (2D, 3D)

Perception and actions restricted to the cells



- It is required to define a data structure, which allows to efficiently access to the smallest set of objects concerned by a process.
- Typical environmental data structures:
 - perception structure dedicated to the static objects,
 - perception structure dedicated to the mobile objects,
 - ground model for "keep on floor," "traversability..."
- Some classical structures are:
 - a) Graph (1D),
 - b) Grid (2D, 3D),
 - c) Spatial tree (2D, 3D). Perception and actions on/in the plane/space. Tree is used for partitioning the plane/space, and providing faster algorithms.



- It is required to define a data structure, which allows to efficiently access to the smallest set of objects concerned by a process.
- Typical environmental data structures:
 - perception structure dedicated to the static objects,
 - perception structure dedicated to the mobile objects,
 - ground model for "keep on floor," "traversability, ..."
- Some classical structures are:
 - a) Graph (1D).
 - b) Grid (2D, 3D).
 - c) Spatial tree (2D, 3D).
 - d) Hybrid, e.g., grid of trees,





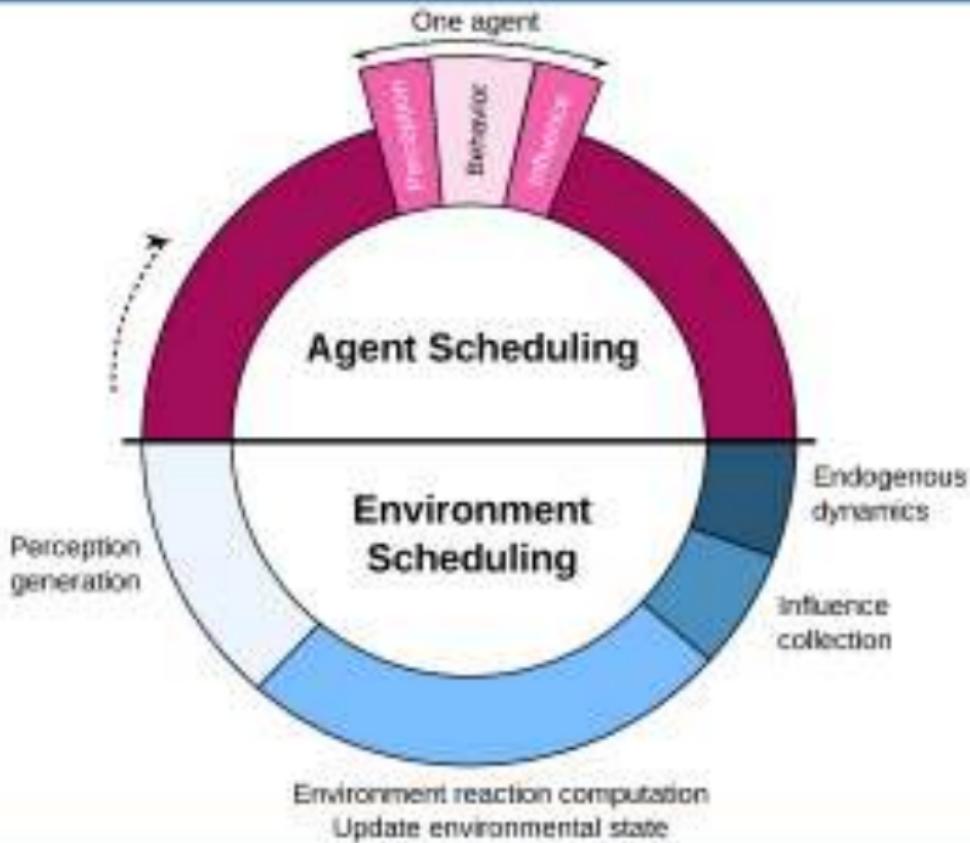
Subsection 3

Agent Scheduling

Definition

The agent scheduling may impose 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, 2001].
- The execution of the agents (order, etc.) has a strong impact on the results provided by the simulation [Lawson, 2000].
- Three major types of scheduling approaches:
 - Synchronous Scheduling: sequential execution.
 - Asynchronous Scheduling: fully distributed execution.
 - Hybrid Scheduling



```
set t to 0 ;
while true do
    set p to {} ;
    forall the a in agents do
        | set p at index a to computePerceptionOf( $\sigma_e, a$ ) ;
    end
    set i to 0 ;
    forall the a in varAgents do
        | set i at index a to runAgent(a.get p at index a) ;
    end
    set i to i ∪ runEnvironmentDynamics;
    set a to computeReactions( $\sigma_e, i$ ) ;
    set  $\sigma_e$  to applyActions( $\pi_e, a$ ) ;
    set t to timeEvolution(t,  $\sigma_e$ ) ;
end
```

This algorithm works only if the agents are not run asynchronously AND the influence-reaction model is used.

Otherwise, synchronization algorithms must be used among the agents, and among the agents and the environment.

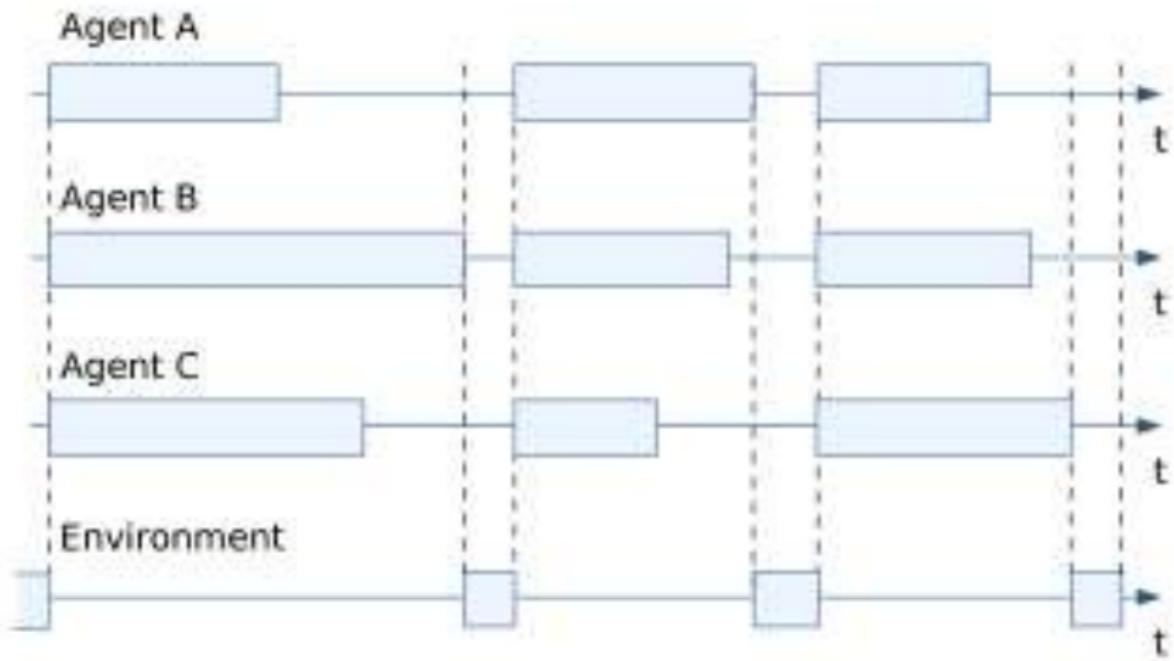
See also ??



Principle

- Each agent runs inside a specific thread.
- Time is evolving for agent a when:
 - Pessimistic approach: iff $t_a \leq t_o \forall o \in \text{other agents}$.
 - Optimistic approach: always, but go back to past when an inconsistent state is detected.

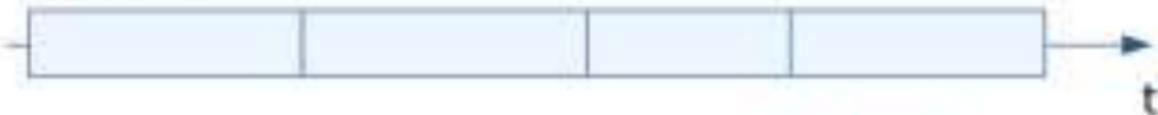
Most of the time, it is done at the platform level.
See ??.



Each agent runs:

```
set timeKnowledge to Map{Identifier => double} ;
set the value for the environment identifier in timeKnowledge to 0 ;
forall the a in known agents do
    set the value for a in timeKnowledge to 0 ;
end
set t to 0 ;
while true do
    if (t < v)|\v in values of timeKnowledge then
        set t to min(values of timeKnowledge) ;
        set p to getEnvironmentPerception(t) ∪ getMessagesAt(t) ;
        set (influences, msgs) to decide(t, p) ;
        emitInfluences(influences, t + δ) ;
        sendMessages(msgs, t + δ) and update the variable timeKnowledge
        of the receivers;
    end
end
```

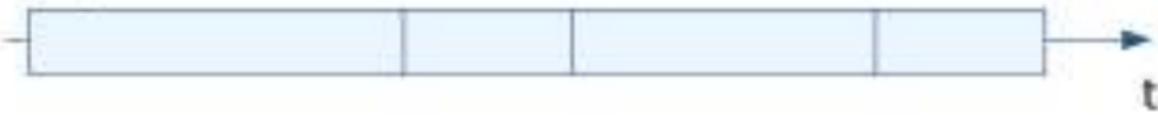
Agent A



Agent B

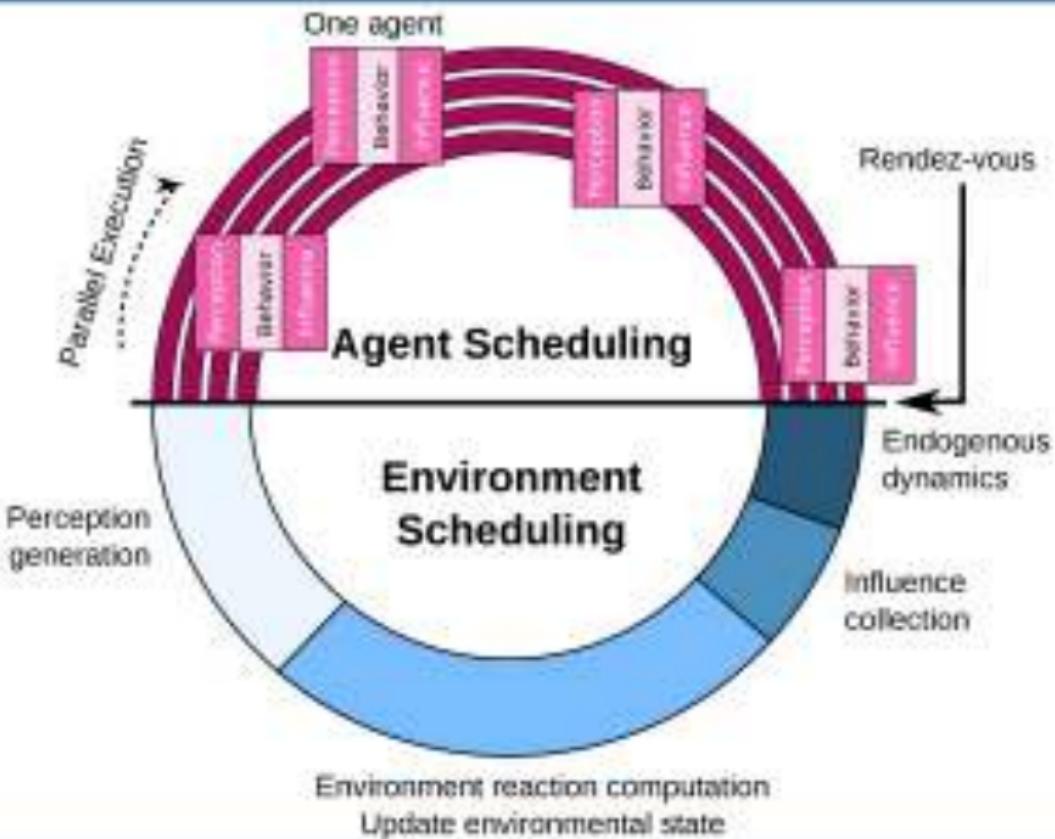


Agent C



Each agent runs:

```
set messageMemory to Map(double => List{Message}) ;
set t to 0 ;
while true do
    set msgs to getMessages() ;
    set tm to min(timestamps of msgs) ;
    if tm < t then
        restoreEnvironmentStateAt(tm- 1) ;
        removeMessagesAfter(tm) ;
        forall the m in messageMemory | timestamp of m > tm do
            sendMessage(invert(m), timestamp of m) ;
            set messageMemory to messageMemory- { m } ;
        end
        set t to tm
    end
    set p to getEnvironmentPerception(t) ∪ getMessagesAt(t) ;
    set (i, msgs) to decide(t, p) ;
    exitInfluences(i, t + δ) ;
    sendMessages(msgs, t + δ) ;
    set messageMemory to messageMemory ∪ { (t + δ) => msgs } ;
    set t to t + δ ;
end
```





Section 3

Examples of Multiagent-based Simulations with SARL



Subsection 1

Carpooling Simulation



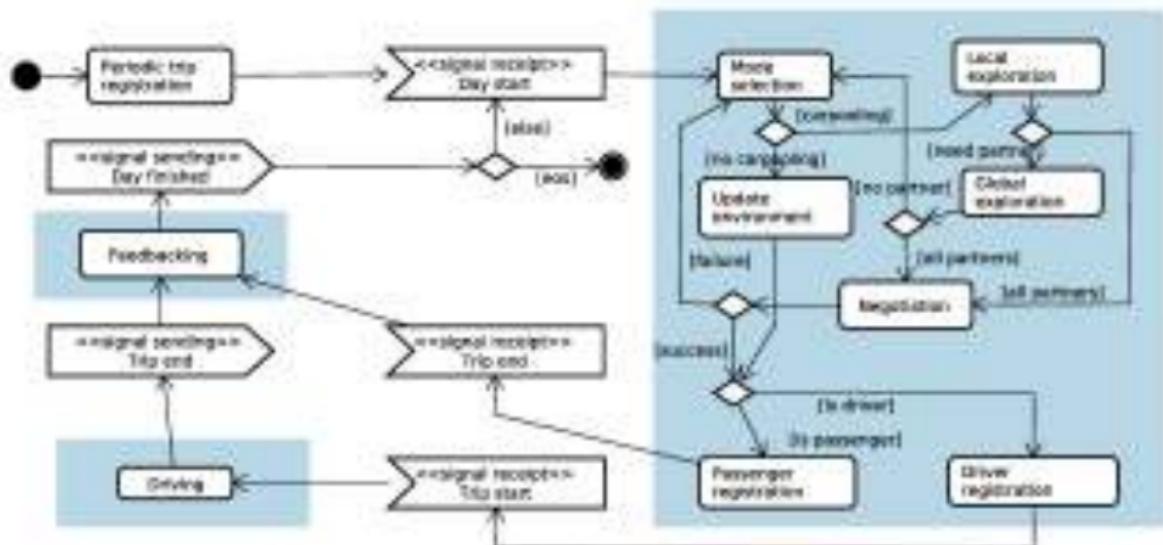
For each day, for each individual

- 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, 2013]

Activity Diagram for an Agent

54





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_s and finishing at D_s :

$$p_s = \langle O_s \rangle \cdot p_s \cdot \langle D_s \rangle$$

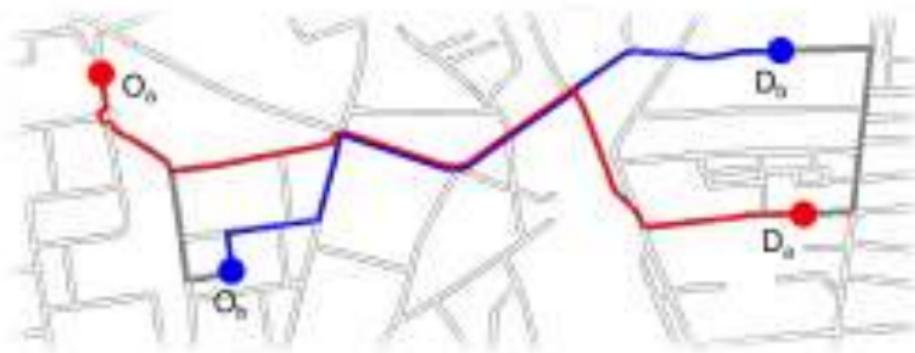
- Most of the time, the shortest paths between O_s and D_s .

* See A* algorithm



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

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

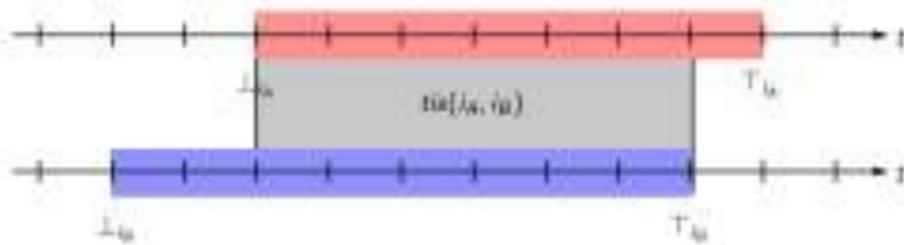


Time window of a trip

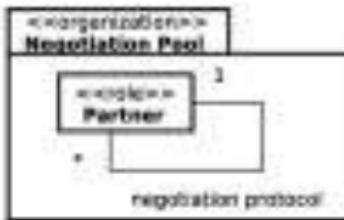
- a trip A is defined by a path and a time window $i_A = [\perp_{i_A}, \top_{i_A}]$

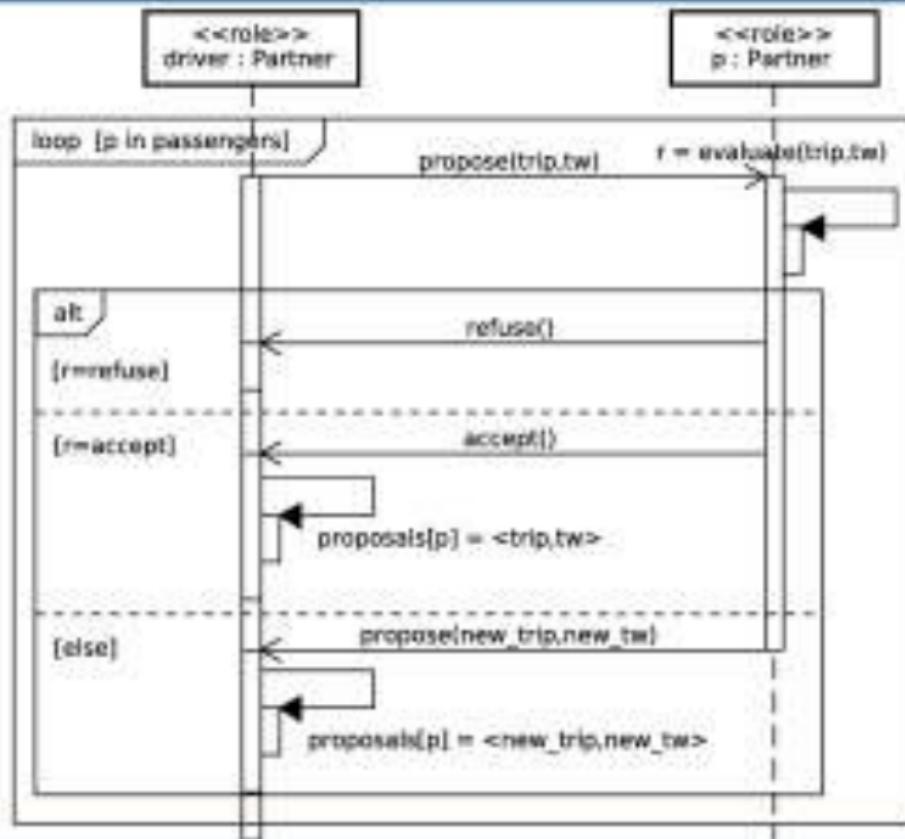
Time Interval Similarity

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

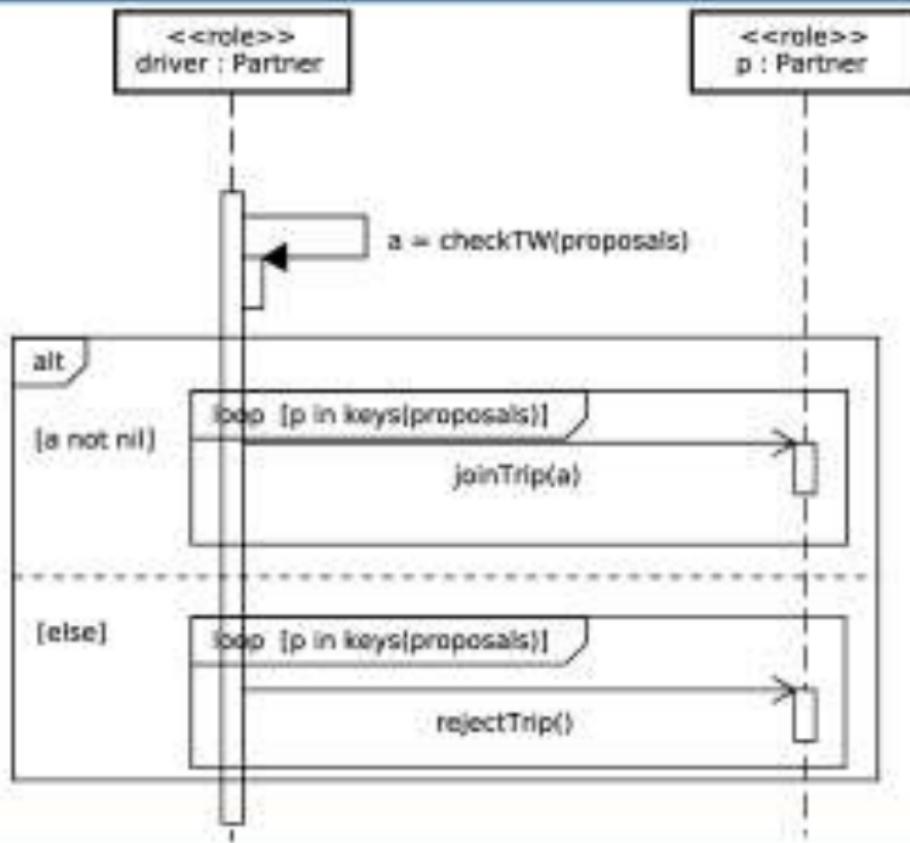


- Each potential partner is invited to play the role Partner in the Negotiation Pool organization.
 - Organizational metamodel
Capacity-Role-Interaction-Organization (CARIO)
[Cossentino, 2010]
- They interact together to negotiate the time window of the trip.



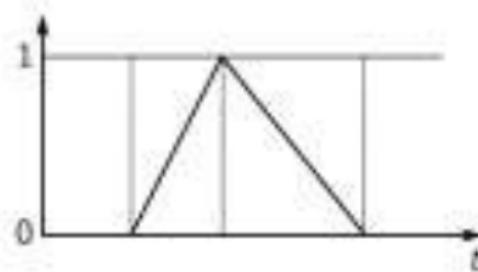


Negotiation Protocol (cont.)



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

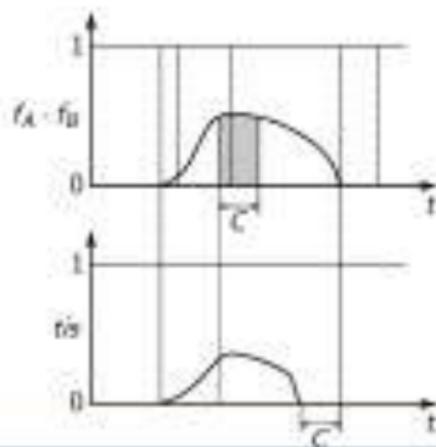
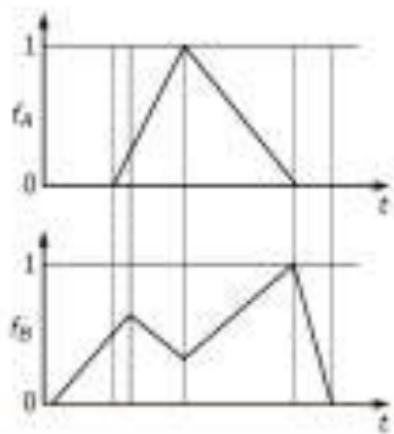
$$f_A : \mathbb{R} \Rightarrow \mathbb{R} : t \mapsto f_A(t) \in [0, 1]$$



- Evaluate the suitability of times for boarding/alighting:

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

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



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

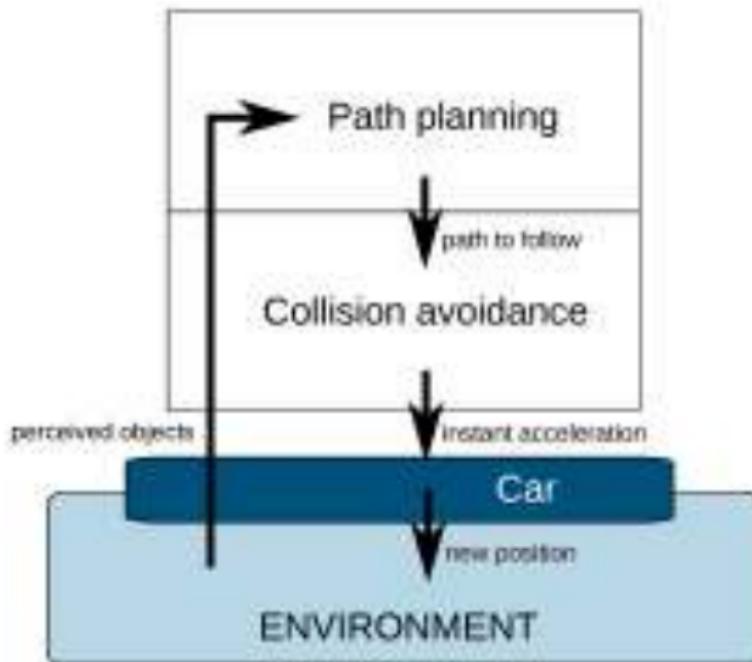
Operations

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

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

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

- Move the vehicles, and avoid physical collisions.



JaSIS 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.

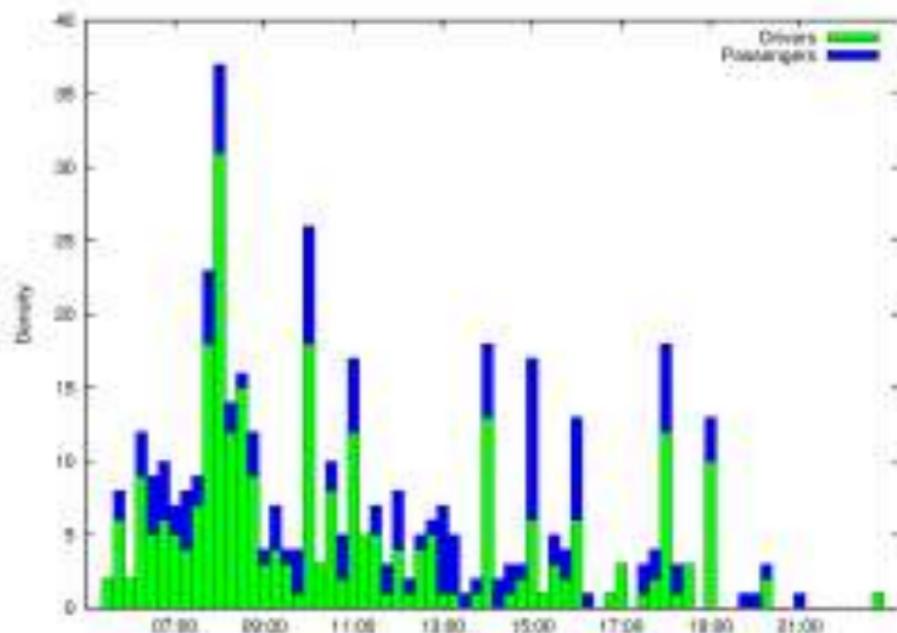
- Principle: compute the acceleration of the vehicle to avoid collisions with the other vehicles.
- Intelligent Driver Model [Treiber, 2000]

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

- Free driving:

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

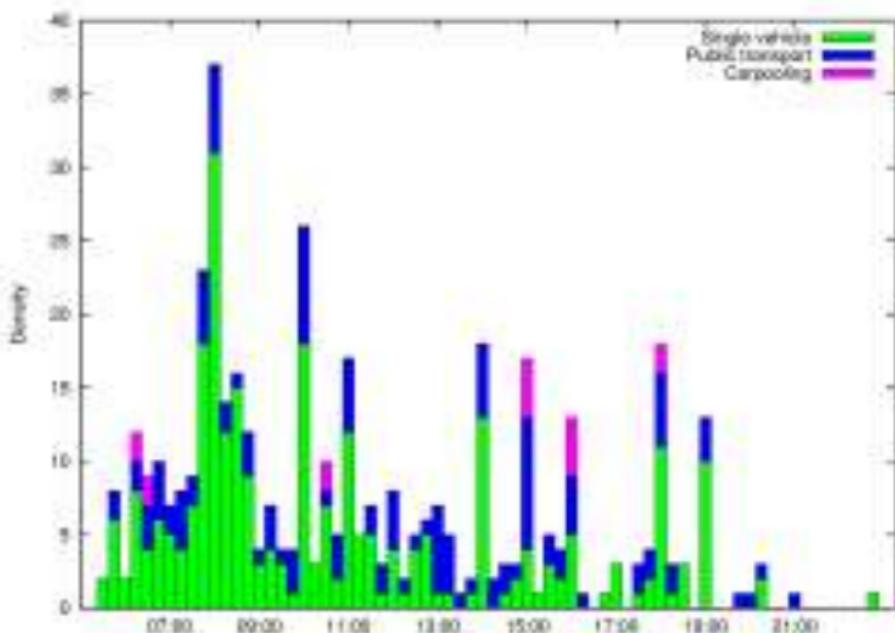
Results: Types of Travellers



- Total population in this scenario: 500.

Results: Selection of the Travelling Modes

m



- Total population in this scenario: 500.



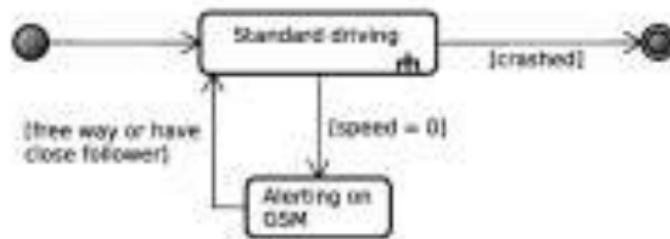
Subsection 2

Rescue on Highway



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.





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

Subsection 3

Pedestrian Simulation



What is simulated?

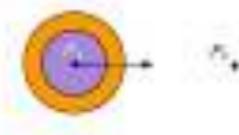
- 1 Movements of pedestrians at a microscopic level.
- 2 Force-based model for avoiding collisions.

[Buisson, 2013]



- The force to apply to each agent is:

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



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

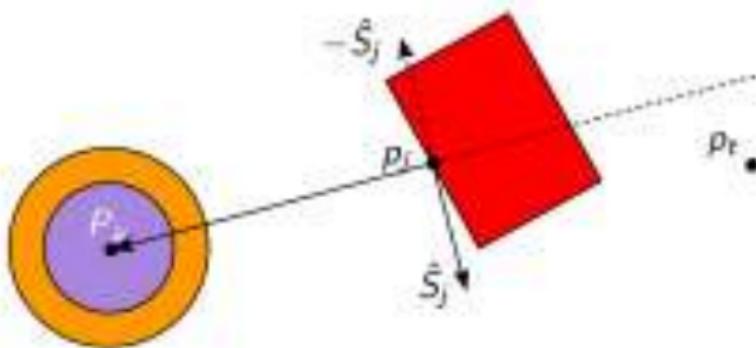
- \vec{F} : collision-avoidance force.
- \hat{S}_i : a sliding force.
- t_c^i : time to collision to object i .
- $U(t)$: scaling function of the time to collision.
- M : set objects around (including the other agents).
- w_a : weight of the attractive force.
- $\delta_{\|\vec{F}\|}$: is g if $x \leq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

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

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

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

- \hat{y} : vertical unit vector.



- How to scale 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.
- σ and ϕ are constants, such that $U(t_{max}) = 0$.



These videos were realized on the SIMULATE[®] tool © Vomela S.A.S.

Subsection 4

Intelligent Vehicle

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 Multiagent-based simulation model is composed of three parts:
 - the behaviors of the agents,
 - the interactions among the agents,
 - the definition of the environment,
- 2 Organizational approach may break down the complexity of the design.
- 3 The simulator should limit/avoid the bias introduced by the running of the model.
- 4 Multiagent-based simulation is well-adapted for transport/mobility system analysis and simulation.



Chapter 3: Agent-Oriented Software Engineering

Stéphane GALLAND



Section 1

Fundamentals of Software Engineering

A methodology ?

- Concepts and abstractions
- Process Model (Cascade, Spiral, Fountain) and development process
- Methodological guidelines for steps of the process
- Languages : modeling (UML), specification (B, Z) , constraint (OCL), programming (Java, C++, SQL)
- Associated tools:
 - Project, Document and Source Management Systems, etc.
 - IDE, Implementation and deployment platforms, etc.

Metamodel [Bézivin, 2005]

- a formal specification of an abstraction, usually consensual and normative.
- acts as a filter on a given system, precisely defined/expressed in a given formalism to extract a particular model.
- strongly links to the notion of ontology

⇒

- a view of the system.
- a formal or semi-formal set of concepts to describe problems and their solutions.

Software (Development) Process [Fuggetta, 2000]

The coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product.

Method [Cernuzzi, 2005]

- prescribes a way of performing some kind of activity within a process, in order to properly produce a specific output (i.e., an artifact or a document) starting from a specific input (again, an artifact or a document).
- Any phases of a process, to be successfully applicable, should be complemented by some methodological guidelines (including the identification of the techniques and tools to be used, and the definition of how artifacts have to be produced) that could help the involved stakeholders in accomplishing their work according to some defined best practices.

Methodology [Ghezzi, 2002]

- A collection of methods covering and connecting different stages in a process
- **Goal** : prescribe a certain coherent approach to solving a problem in the context of a software process by preselecting and putting in relation a number of methods.

⇒ Guide to produce a model of a solution from the model of a problem

The fundamental activities of a software development process
[Sommerville, 2004]

1 Specification

- Formal approaches : based on the Z language [Luck, 1995], DESIRE methodology [Brazier, 1998], OZS [Gruer, 2004].
- Semi-formal approaches : Functional approaches [Jacobson, 1992] (use cases), goal-oriented approaches [Van Lamsweerde, 2001], role-oriented approaches [Kendall, 2000]

2 Design and Implementation

3 Validation and Verification (and Software Tests)

4 Evolution and Maintenance

Evolution of agent-oriented methodologies

- **Initial vision** : system mainly focuses on the agent and its individual aspects.
- **Current vision** : system considered as an organization in which agents form groups and hierarchies, and follow specific rules, norms and behaviors.

ACMAS (Agent-centered) \Rightarrow OCMAS (Organization-centered)

Section 2

Agent-oriented Modeling Languages

A methodology is generally composed of a **process** and its **products** [Rolland, 1999]. These products are documented, implemented or represented using specific languages.

The different types of languages

- Specification languages (formal or not, e.g. Z, Object-Z, B),
- Modeling languages (e.g. OMT^a, UML^b, OML^c),
- Programming languages (e.g. Java, C++, SQL, IDL, Lisp, etc),
- Constraint languages (e.g. OCL^d)
- Natural language

^aObject Modeling Technique of James Rumbaugh.

^bUnified Modeling Language.

^cOpen Modeling Language of Brian Henderson-Sellers, Donald G. Firesmith et Ian Graham.

^dObject Constraint Language [OCL, 2006]

Object-oriented Modeling Languages

- THE standard: UML

UML is a metamodel, itself derived from a meta-metamodel: the MOF^a

- Pb/Challenge: does not provide all the necessary features for agent-oriented modeling
- UML defines a range of tools to facilitate its extension (the profiles that defines specific stereotypes, constraints, etc..).

^aMeta-Object Facility : <http://www.omg.org/mof/>



Agent-oriented Modeling Languages

No defined standard.

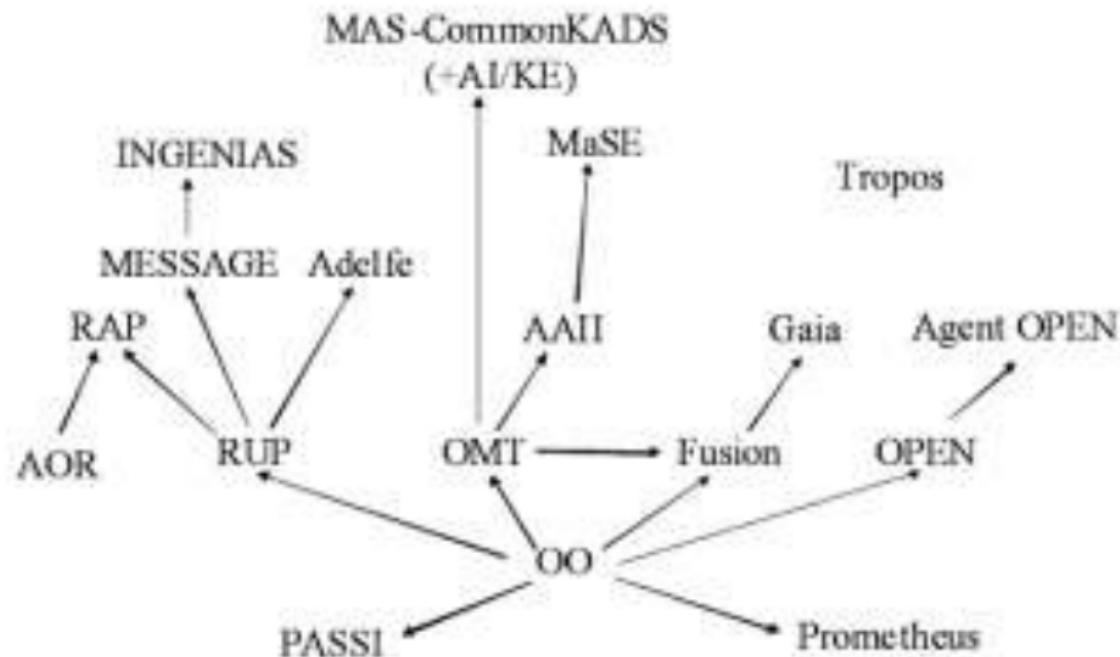
- AUML [Odell, 2001, Odell, 2000, Bauer, 2001, Bergenti, 2000]² — Agent Unified Modeling Language
- AML [Trencansky, 2005] — Agent Modeling Language
- AORML [Wagner, 2003] — Agent-Object-Relationship Modeling Language

²AUML : <http://www.euml.org>

My tips: reuse as much as possible UML and specific profiles

Section 3

Agent-oriented methodologies



Subsection 1

AAII

AAII [Kinny, 1996]

- Developed by the Australian Artificial Intelligence Institute (AAII)
- iterative methodology
- Two points of Viewpoint:
 - **External:** system decomposed into agents: purpose, responsibilities, services they perform, external interactions
 - **Internal:** agent design at individual level using the BDI* Architecture

*Belief-Desire-Intention

External Viewpoint

- **Agent Model:** describes the hierarchical relationship among different agent classes, and identifies agent instances which may exist within the system
- **Interaction Model:** describes the responsibilities of an agent class, the services it provides, associated interactions and control relationships between agents classes.

External Viewpoint

Steps:

- 1 Identify the roles (functional or organizational) of the application domain. Elaborate an agent class hierarchy.
- 2 For each role, identify its associated responsibilities, and the services provided and used to fulfill those responsibilities. AS well as services (includes interaction with external envt or uses) provided to/by other agents upon request.
- 3 For each service, identify the interactions associated with the provision of the service, the performatives (speech acts) requires for those interactions and their information content.
- 4 Refine the agent hierarchy

Internal Viewpoint

BDI Architecture

- **Belief Model:** describes the information about the environment, the agent's internal state, the action it may perform.
- **Goal Model:** describes the goals that an agent may possibly adopt, and the events to which it can respond.
- **Plan Model:** describes the plans that an agent may possibly employ to achieve its goals.

Subsection 2

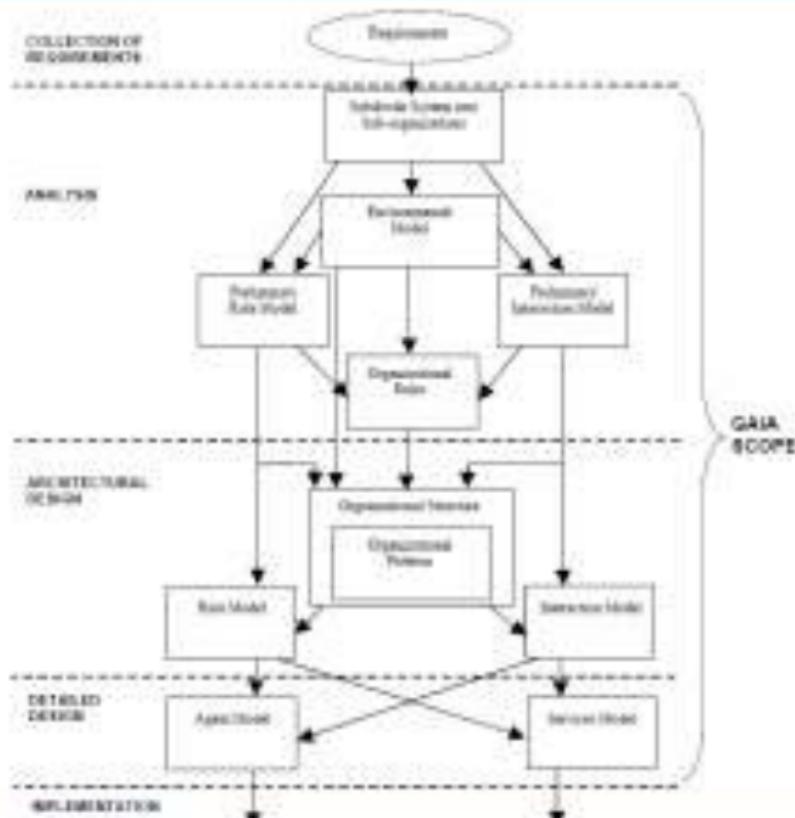
Gaia

Object-oriented

Methodology

Tool support

Implementation





Analysis phase

- 1 Identify the role in the system
- 2 For each role, identify and document the associated protocols
- 3 Using the protocol model as a basis, elaborate the roles models
- 4 Iterate stages (1)-(3)

Design phase

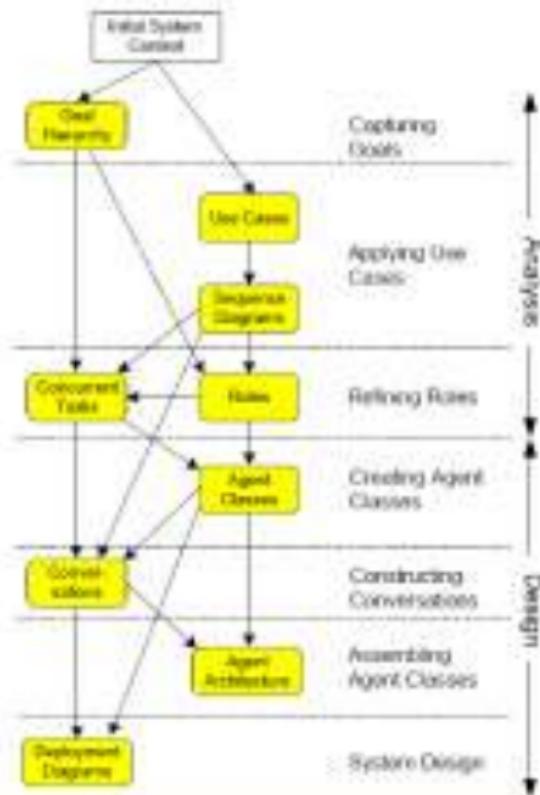
- **Agent Model:** document the various **agent types** that will be used in the developed system, and the **agent instances** that will realize these agent types at run-time.
- **Service Model:** identifies the **services** associated with each agent role, and to specify the main properties of these services
- **Acquaintance Model:** simply define the communication links that exist between agent types.

Subsection 3

MASE

MASE ?

- a more Global Vision
- Strongly associated with UML
- Evolution: O-MaSE, organizational version.
- Tool associated: agentTool



Section 4

Focus on the ASPECS methodology

Subsection 1

Introduction

Main goal

Propose a software development process and associated tools for the analysis, design, implementation of holonic multi-agent systems (HMAS).

How ?

- 1 An organizational metamodel : **CRIO**

Capacity, Role, Interaction, Organization

- 2 A methodological process : **ASPECS**

Agent-oriented Software Process for Engineering Complex Systems

- 3 A implementation/deployment platform : **JANUS**

<http://aspecs.org>

Scientific Context:

Modeling of Complex Systems

General Principle of the approach

- Exploit the intrinsic hierarchical structure of Complex Systems [Simon, 1996].
- Recursively decompose a system into subsystems.

Constraints

Modularity and reusability of models.



3 main mechanisms

- Decomposition
- Abstraction
- Organization

Decomposition

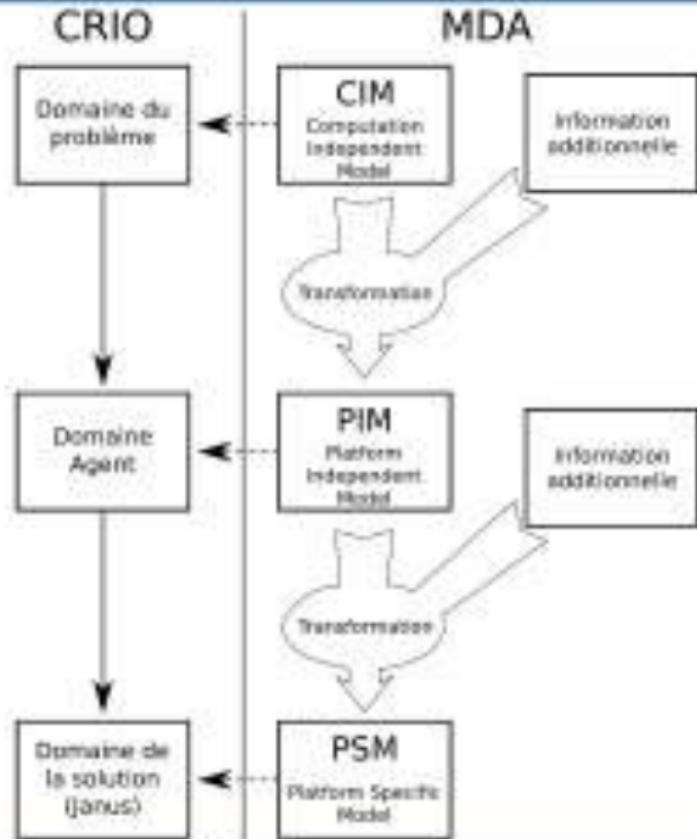
- Divide the problem into sub-problems.
- Each sub-problem being easier to study, relatively isolation of each sub-pb.
- Manages complexity by limiting the view of the designer: at any time, only one part of the problem is studied.

Abstraction

- Process of defining a simplified model of the system focusing only on certain details or properties while hiding others.
- Managing complexity by limiting the view of the designer: it focuses on the main aspects of the problem, at the expense of less relevant ones.

Organization

- Process of identifying and managing the relationships between the different components of the problem.
- Managing complexity in
 - grouping a number of basic elements and treating them as a "whole" at a higher level of abstraction.
 - providing the means to describe the high-level relationships between the various components. e.g. a number of elements may need to work together to provide a particular functionality.





Overall Description

- Provides an organizational approach and a set of methodological guidelines from the analysis to the implementation.
- Iterative Process
- respects current Standards :
 - Process : SPEM^a.
 - Products and Modeling Languages : UML 2.0
 - Communication Protocols : FIPA.

^aSoftware Process Engineering Metamodel [SPEM, 2007]

http://www.omg.org/technology/documents/modelling_spe_mining.htm#spe

Subsection 2

System Requirements Phase

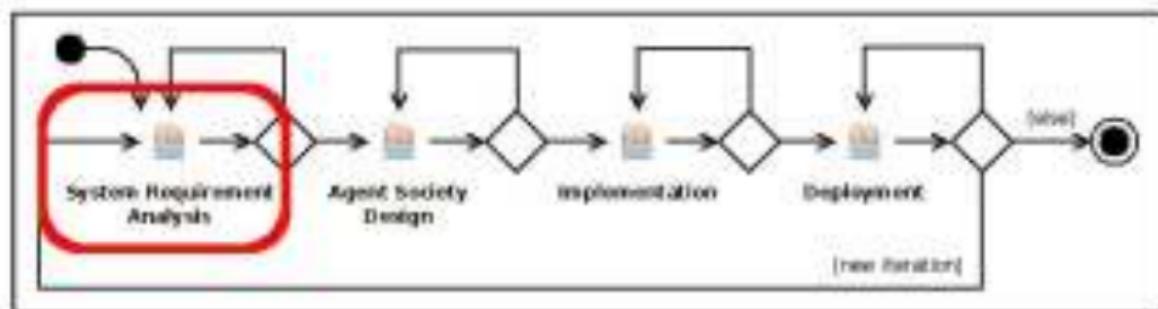


ASPECS

Analysis Phase

System requirements

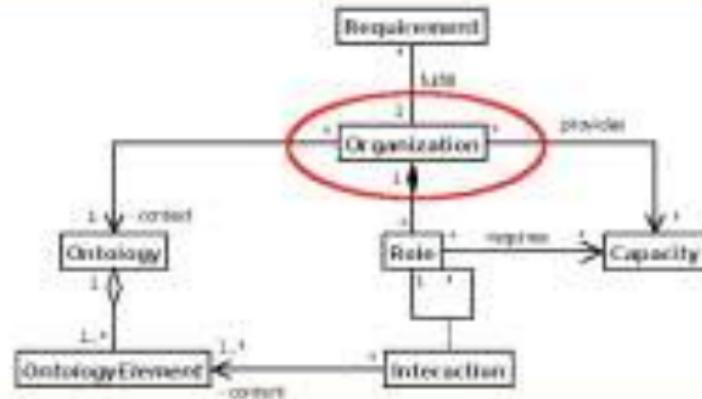
- Identifying a hierarchy of organizations, whose global behavior may fulfill the system requirements under the chosen perspective independently of a specific solution.
- Collecting knowledge on the problem domain and organize it within an ontology.



Concepts : CRIo Problem Domain.

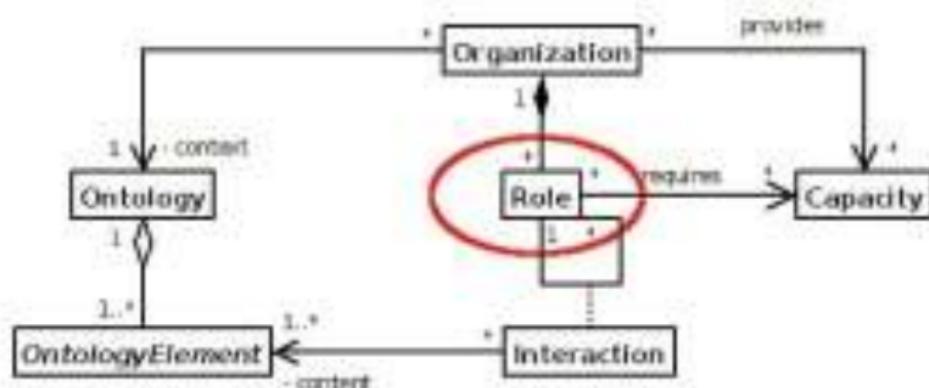
Organization

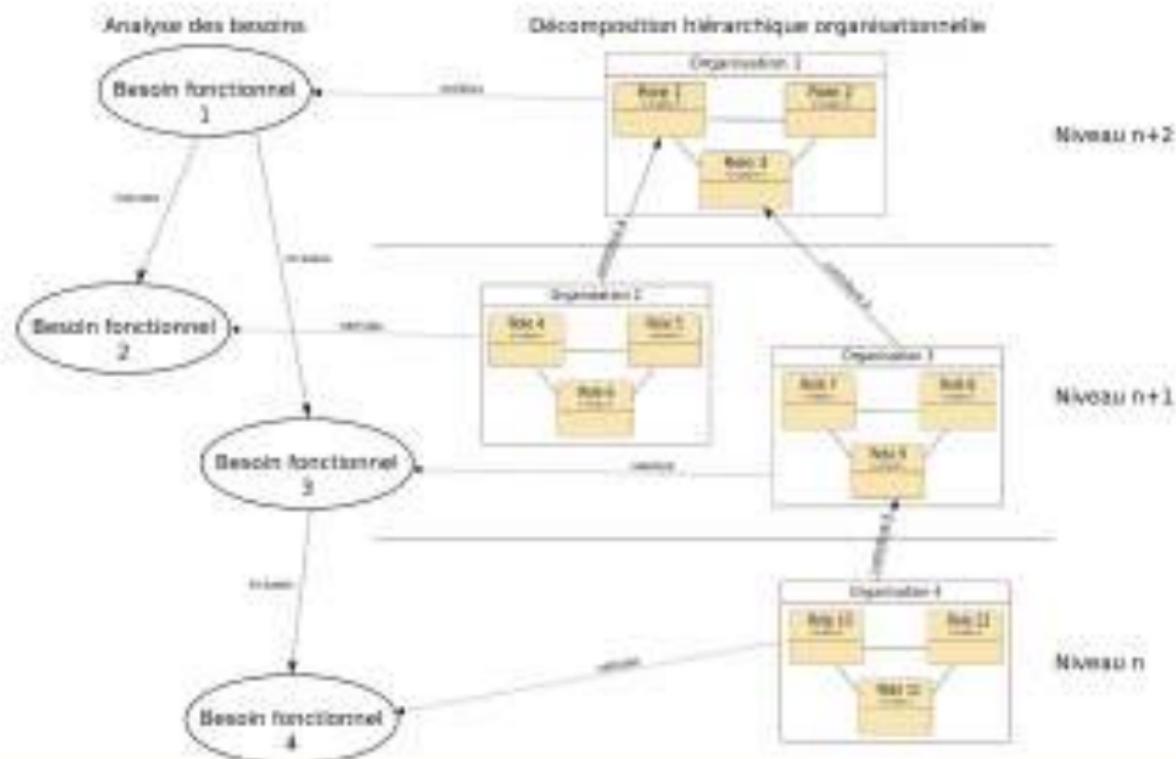
- a collection of roles that take part in systematic institutionalized patterns of interactions with other roles in a common context.
- this context consists in shared knowledge and social rules/norms, social feelings and it is defined according to an ontology
- aims at fulfilling some requirements.
- may itself be decomposed into sub-organizations.



Role:

- defines an expected **behavior** as a set of role tasks ordered by a plan, and a **status**: set of rights and obligations in the organization context.
- contribute to the fulfillment of (a part of) the requirements of its organization
- A role can interact with other roles defined within the same organization.



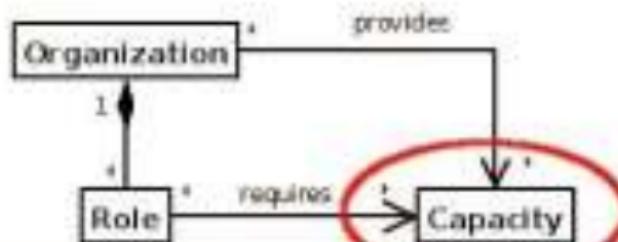


Capacity

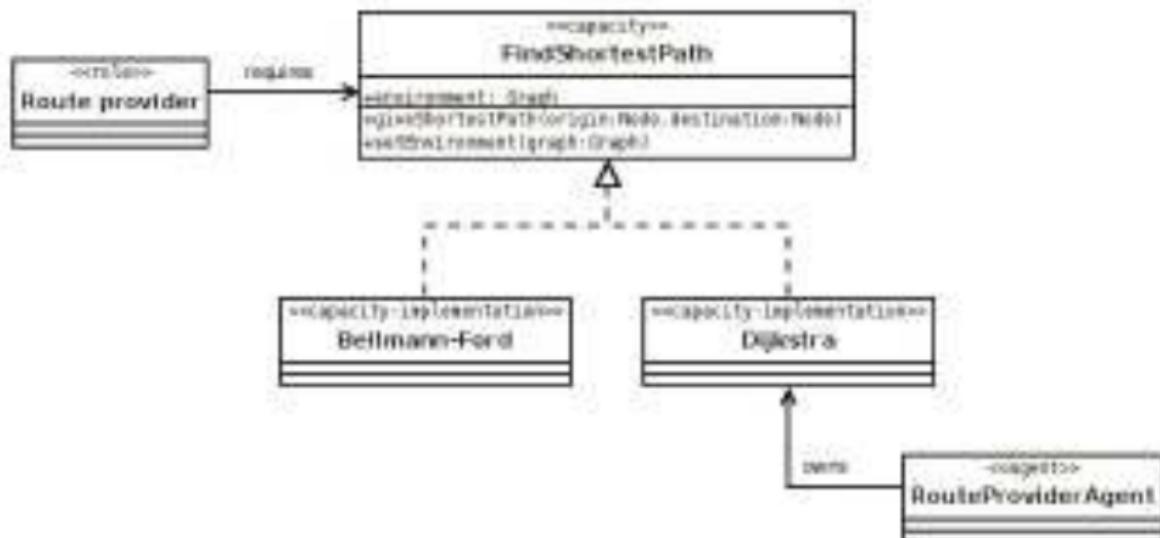
- describes what a behavior is able to do or what a behavior may require to be defined
- Requires by a role
- Provides by an agent or an organization

Double function within CRIo:

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

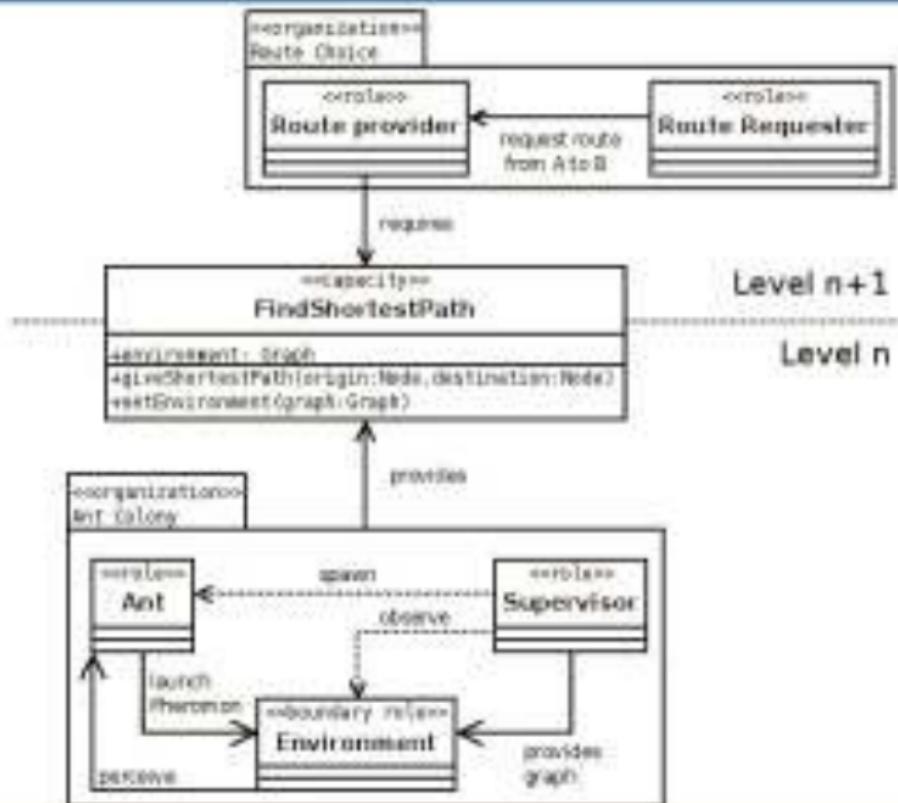


Capacity : Interface between agent and role



Capacity : Interface between two abstraction levels

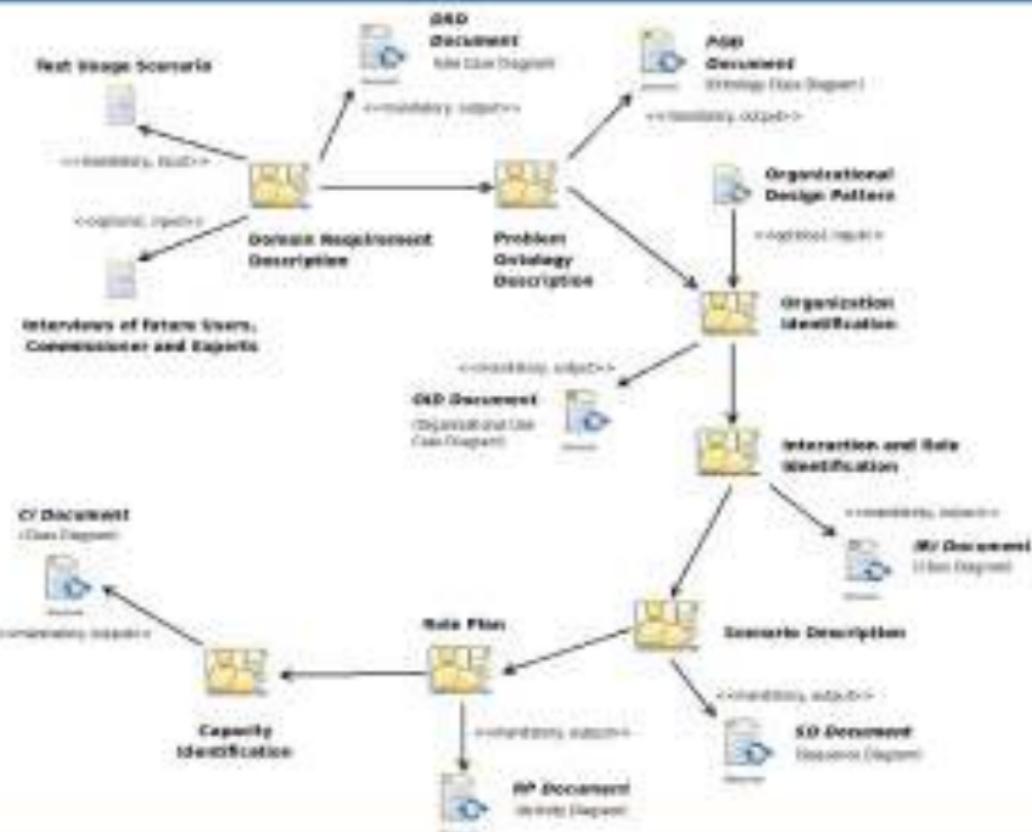
46





System Requirements Phase

47



Subsection 3

Example

Example

Simulation of an Industrial Plant

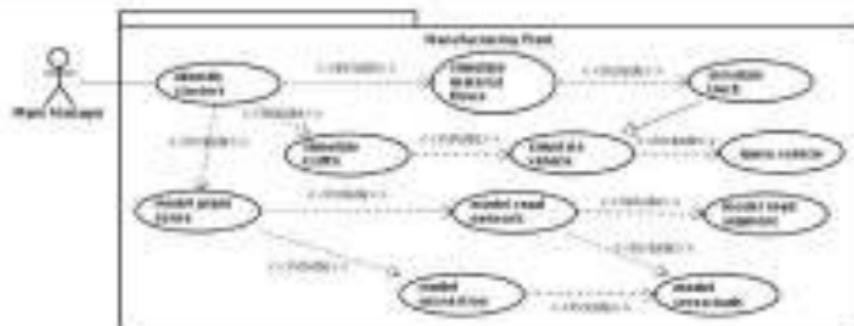
Analysis

Goal

To design a robot soccer simulator capable of simulating team's behaviors and thus provide a platform to test different strategies of coordination between the robots.

Goals

- Provide a first description of the context of the application and its features.
- Identify, classify and prioritize all the functional and non-functional requirements.
- Provide a first estimate of the scope/perimeter of the application, its size and complexity.



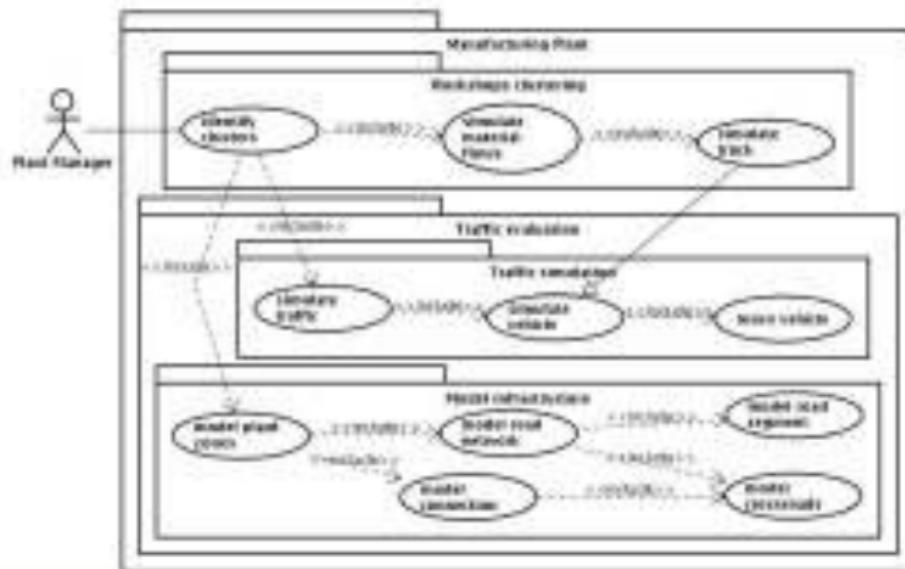
Goals

- Capitalize the knowledge available on the application and its environment.
- Describe the domain concepts and their relationships.
- see also <http://protege.stanford.edu>



Goal:

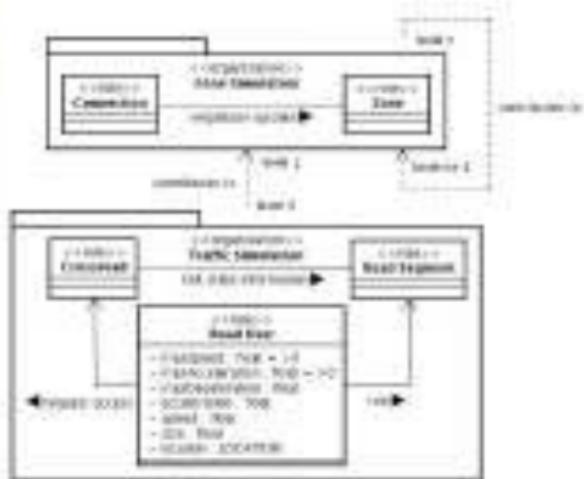
- Establish a first organizational decomposition of the system.
- Define the objectives of each organization.





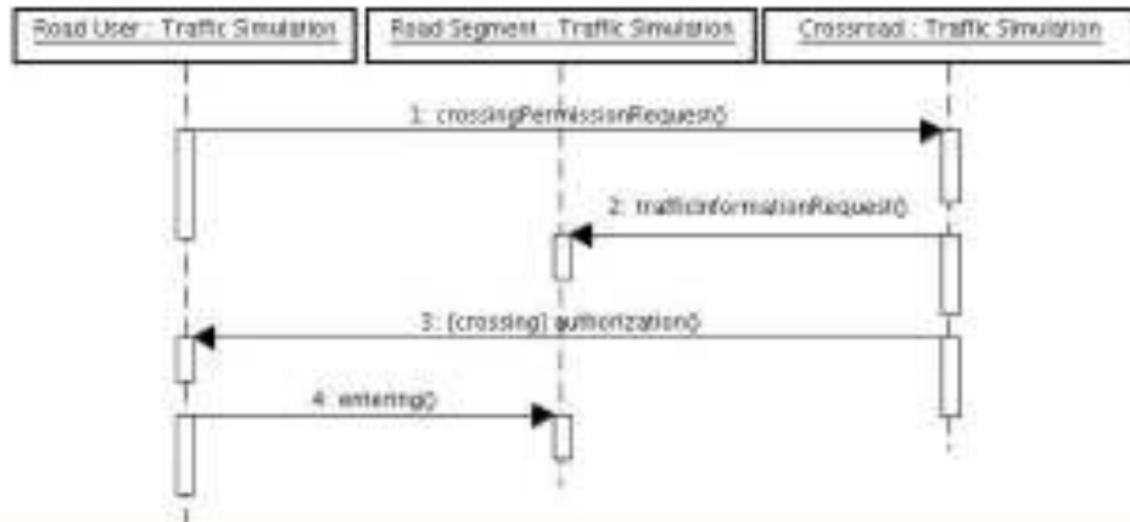
Goals

- Decompose the overall collective behavior embodied by an organization into a set of interacting roles.
- Describe the responsibilities of each role.



Goals

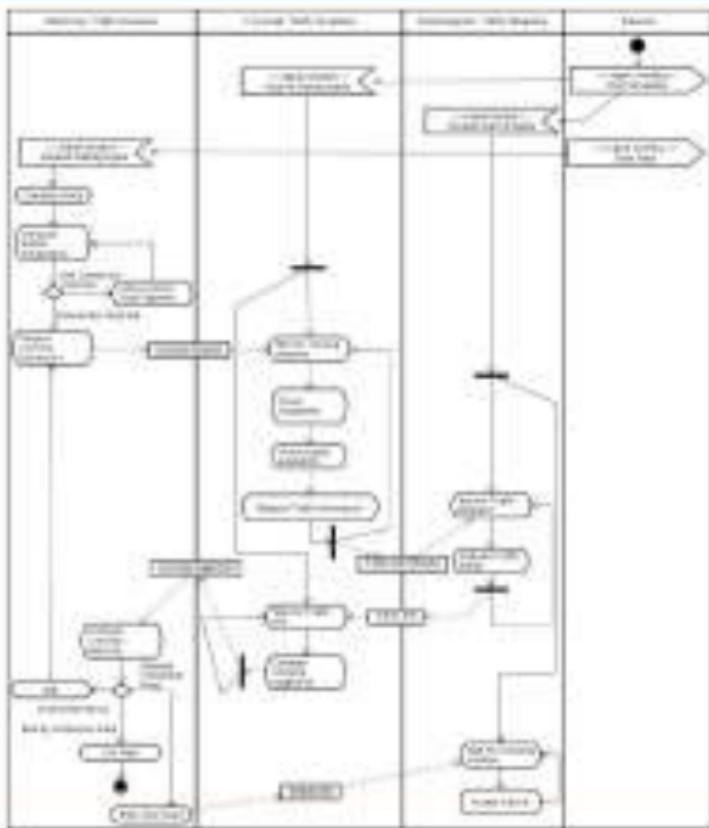
- Describe the sequence of interactions between the roles of an organization
- Clarify their means of coordination to meet the objectives of their organization.





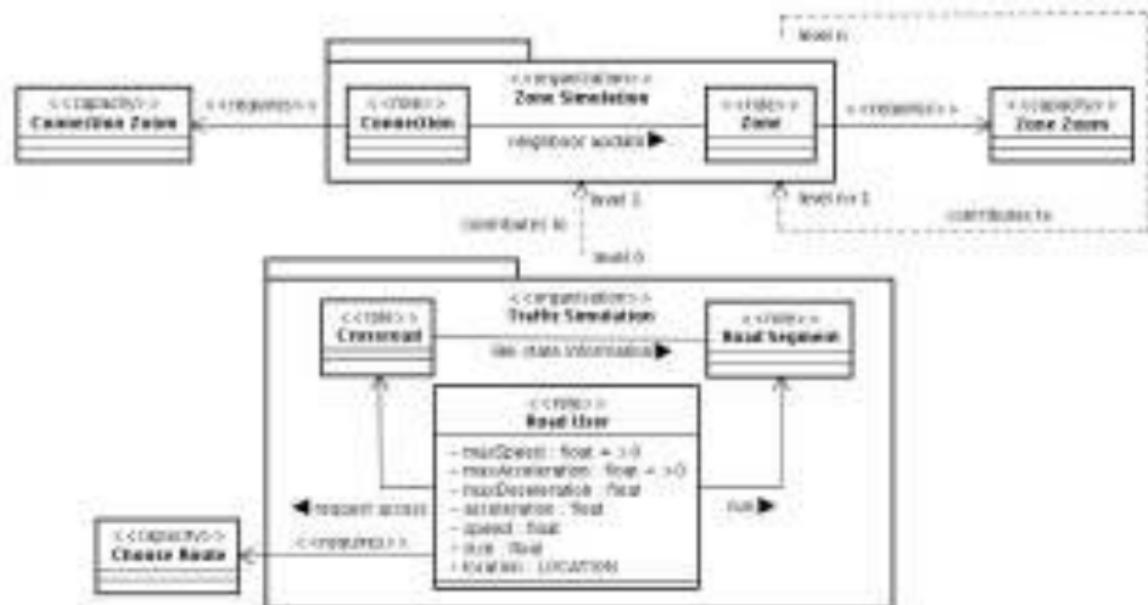
Goals

- Specify the behavior of each role with its goals and interactions.
- Describe the combination and scheduling of interactions, external events and tasks of the behavior of each role.



Goals

- A description of what an organization is able to do without any kind of specification on how to do it.
- it means that the results described by a capacity may be reached by adopting different strategies.
- definition of generic role behaviors by identifying which know-how a role requires from the individual that will play it.



Subsection 4

Agent Society Design Phase

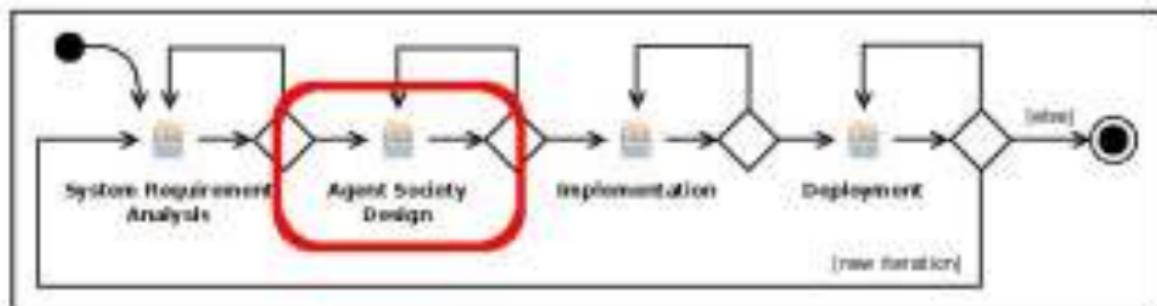


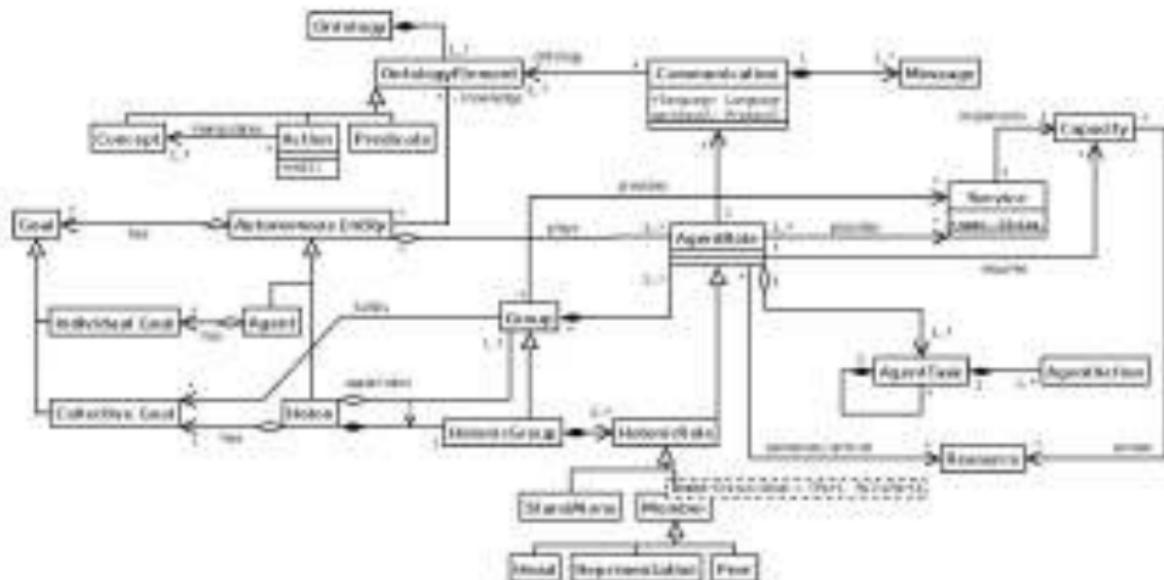
ASPECS

Design Phase

Agent Society Design Phase

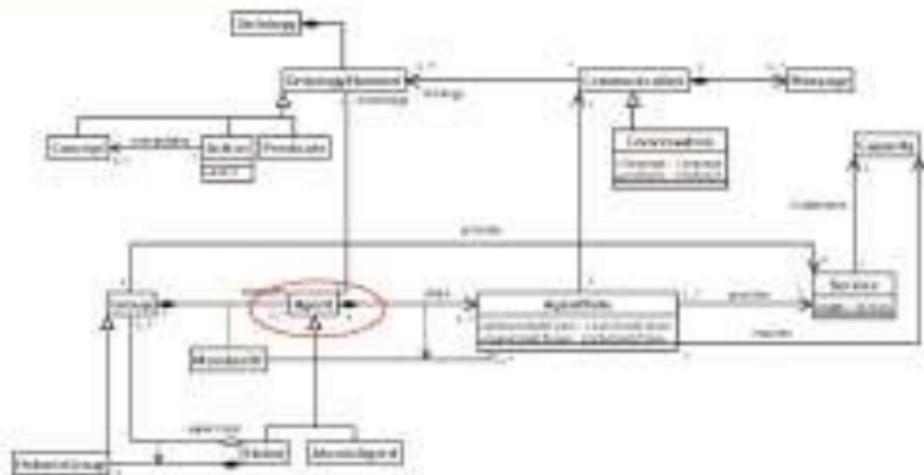
- designing a society of agents, whose global behavior is able to provide an effective solution to the problem described in the previous phase and to satisfy associated requirements.
- a model of the agent society involved in the solution in terms of social interactions and dependencies among entities (Holons and/or Agents).

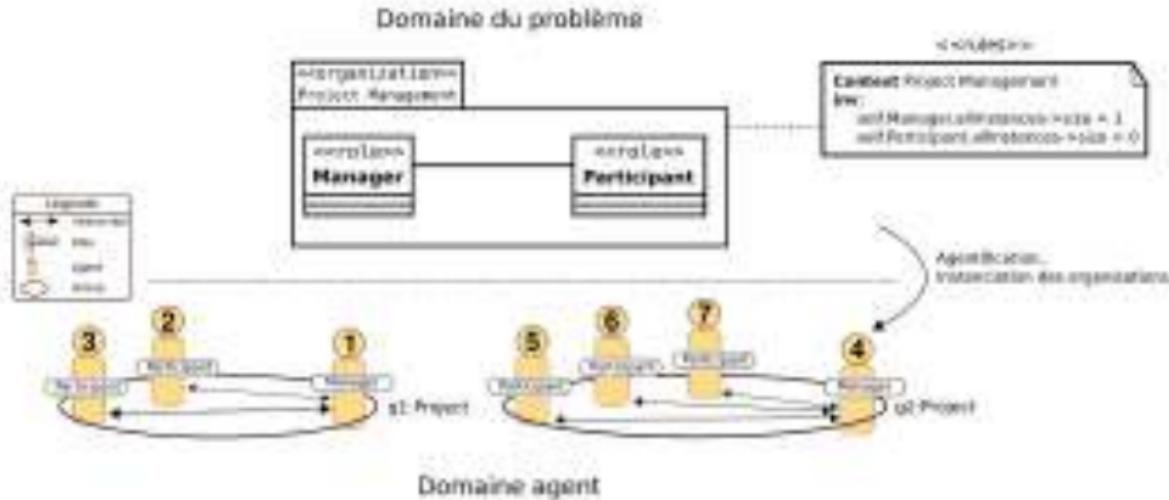


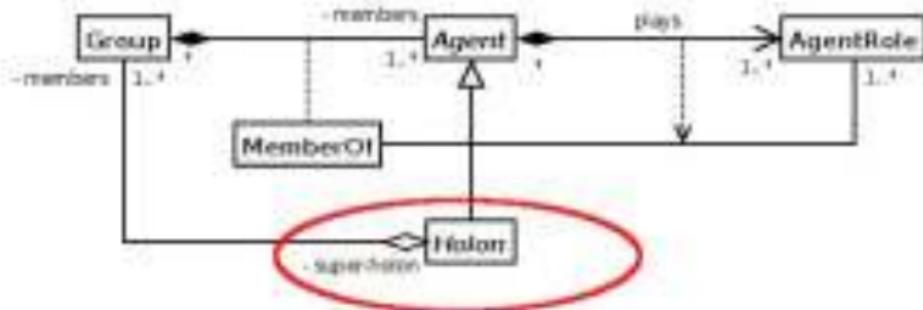


Agent

An autonomous entity that has specific individual goals and the intrinsic ability to realise some capacities.

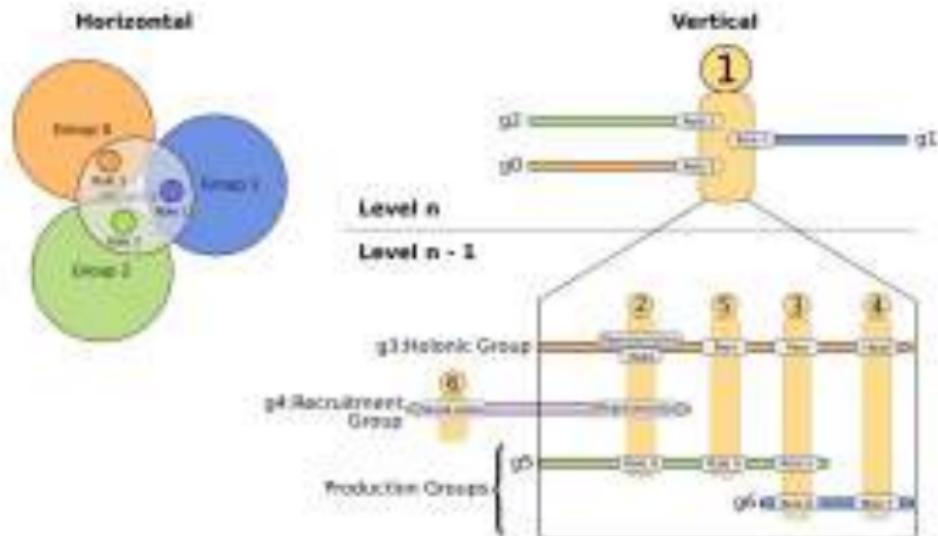


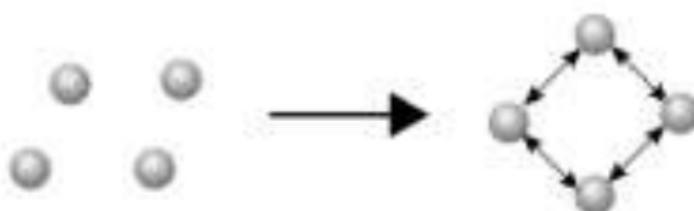




Holon

- Each holon is an autonomous entity that has collective goals (shared by all members) and may be composed by other holons, called members or sub-holons.
- A composed holon is called super-holon.
- A super-holon is not only characterised by its members but also by their interaction patterns.
- Two super-holons may be created from the same set of sub-holons if the ways their members are interacting differ.
- A super-holon contains at least one single *holonic group* to define how members get organised and govern the super-holon, and a set of *production groups* (at least one) to describe how members interact with and coordinate their actions to fulfil the super-holon objectives.





Federation

- All agents are equal
- Requires algorithms for planning and decision-making as a standard MAS



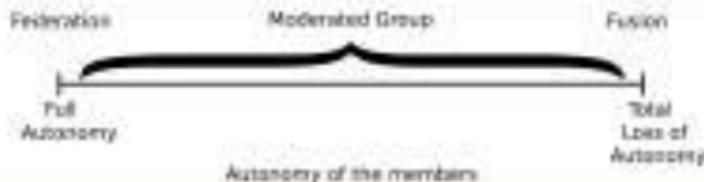
Fusion

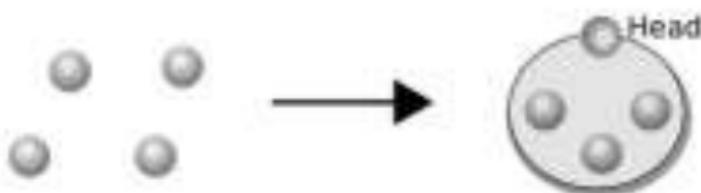
- Merges the agents into a new entity (as a whole)
- Mechanisms required for the fusion/merging of powers/capacities/abilities
- Difficult to implement



Moderated Group [Gerber, 1999b]

- A sub-group are representatives of the holon, of the community with a flexible level of representativity/authority.
- Wide range of possible configurations of agents' autonomy/commitments.

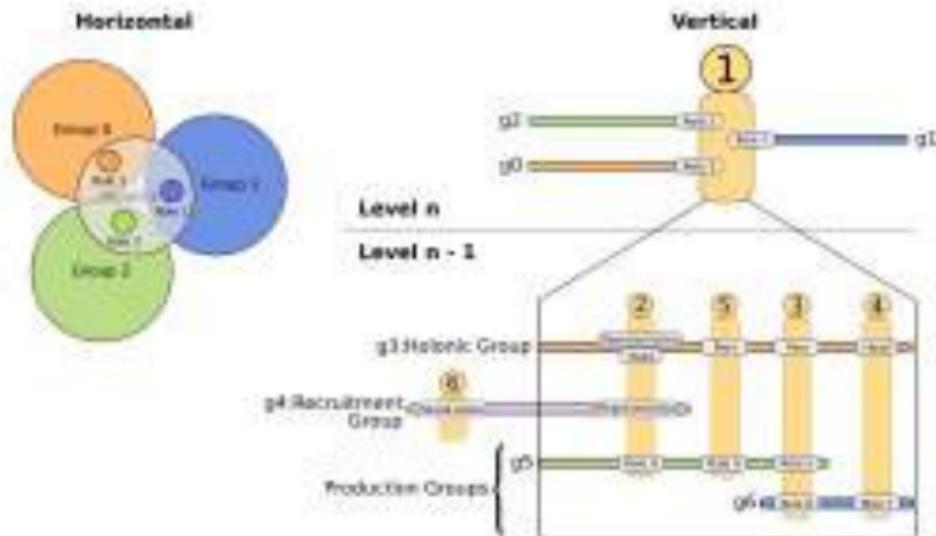




Moderated Group [Gerber, 1999b]

- A sub-group are representatives of the holon, of the community with a flexible level of representativity/authority.
- Wide range of possible configurations of agents' autonomy/commitments.

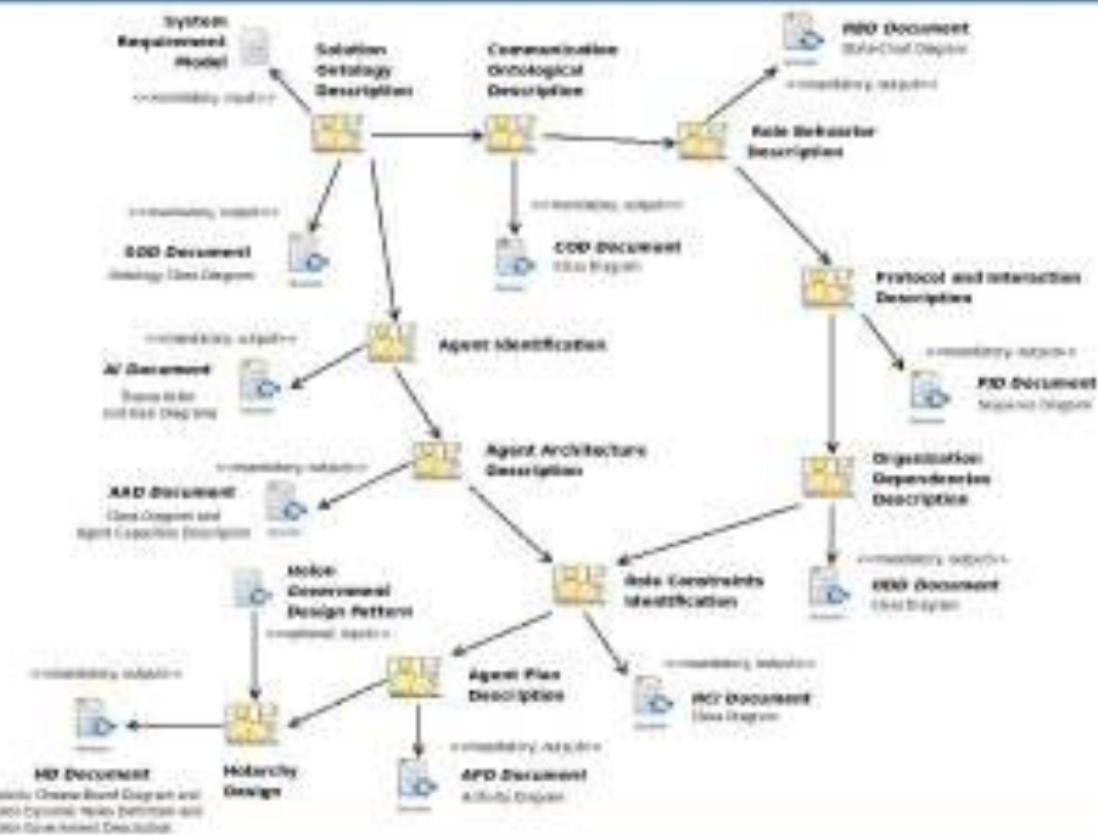
Governmental Structure	Configuration
Monarchy	one Head, one voting head, no voting Part
Oligarchy	n Heads, n voting heads, no voting Part
Polyarchy	n Heads, n voting heads, k voting Parts (for certain decisions)
Anarchy	Everybody is Head, everybody is voting





Agent Society Phase

29



MDD Document
Holistic Design Board Diagram and
MDD Diagram: Holistic Development and
Holistic Assessment Description

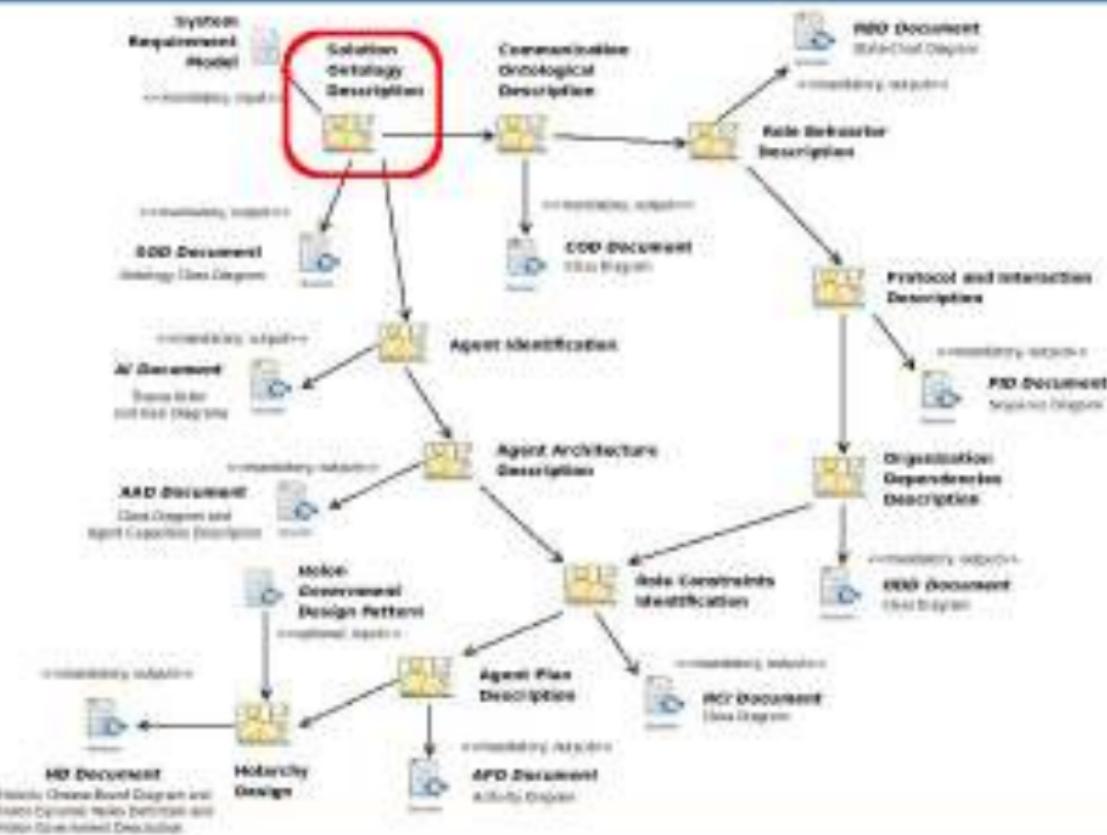
Subsection 5

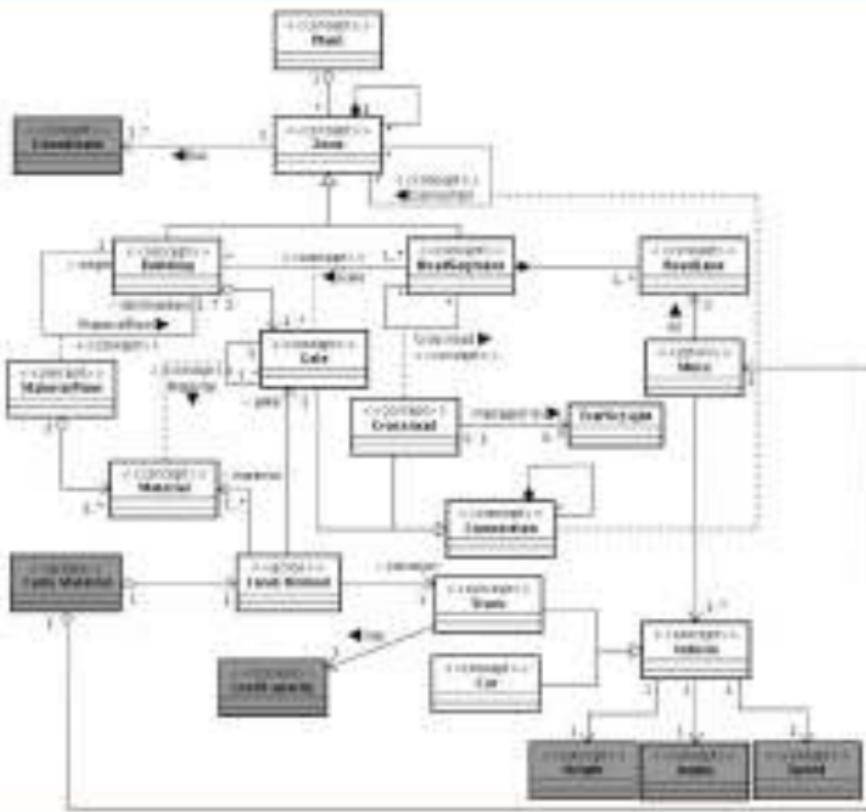
Example

Example

Simulation of an Industrial Plant Design

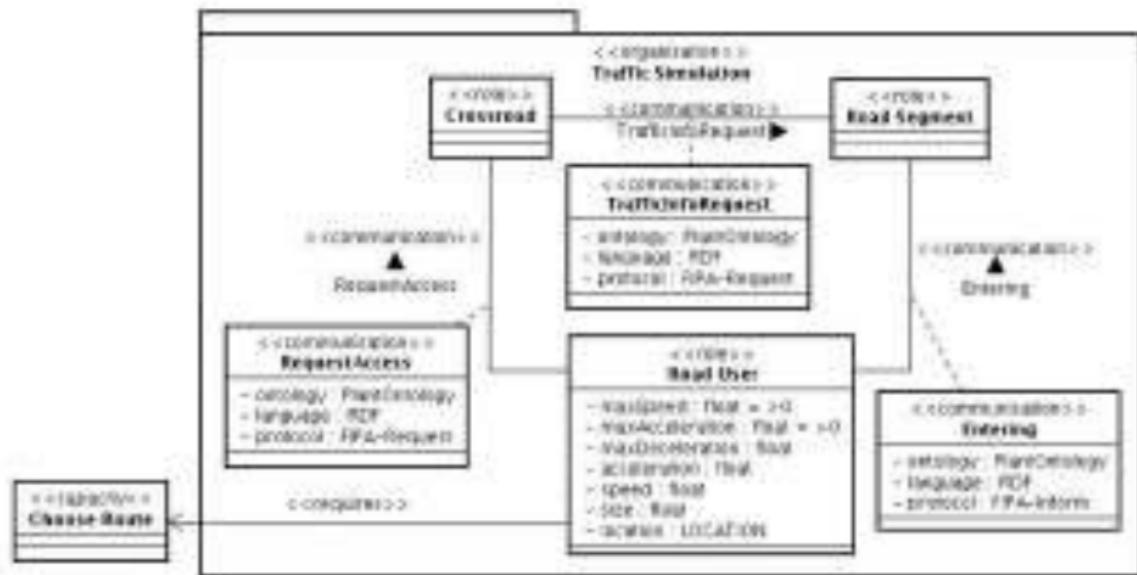
Agent Society Phase – Solution Ontology Description



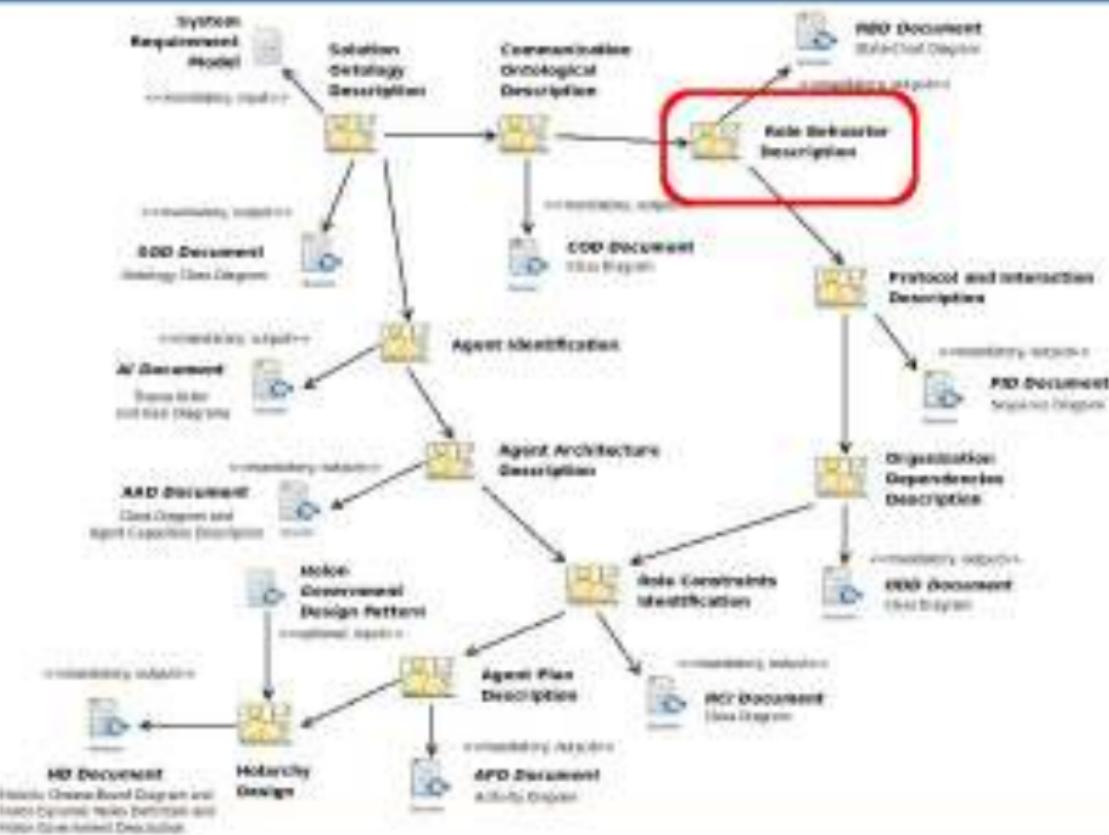


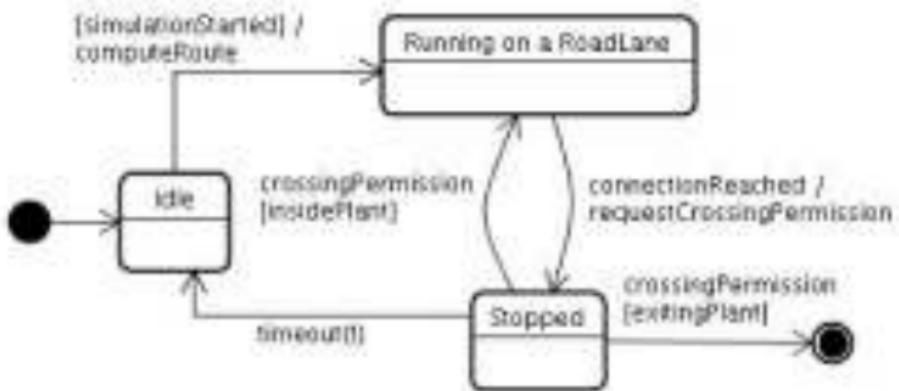
Agent Society Phase – Communication Ontological Description





Agent Society Phase – Role Behavior Description

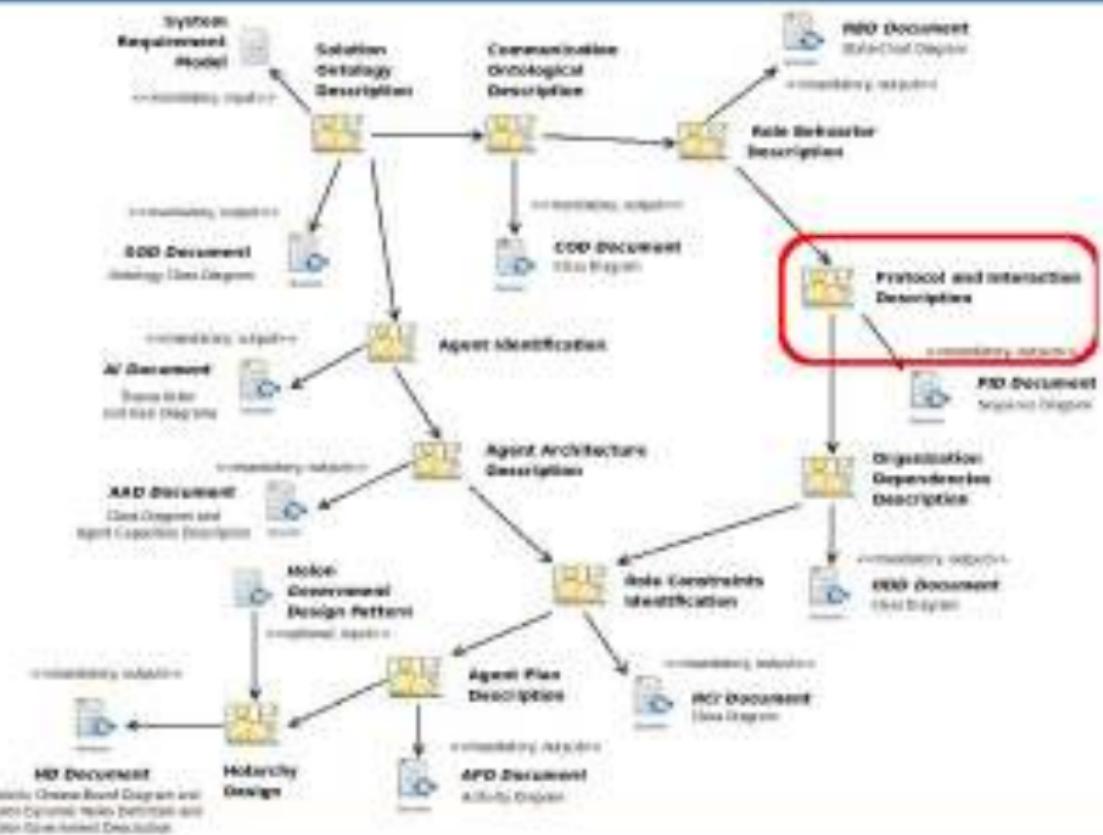




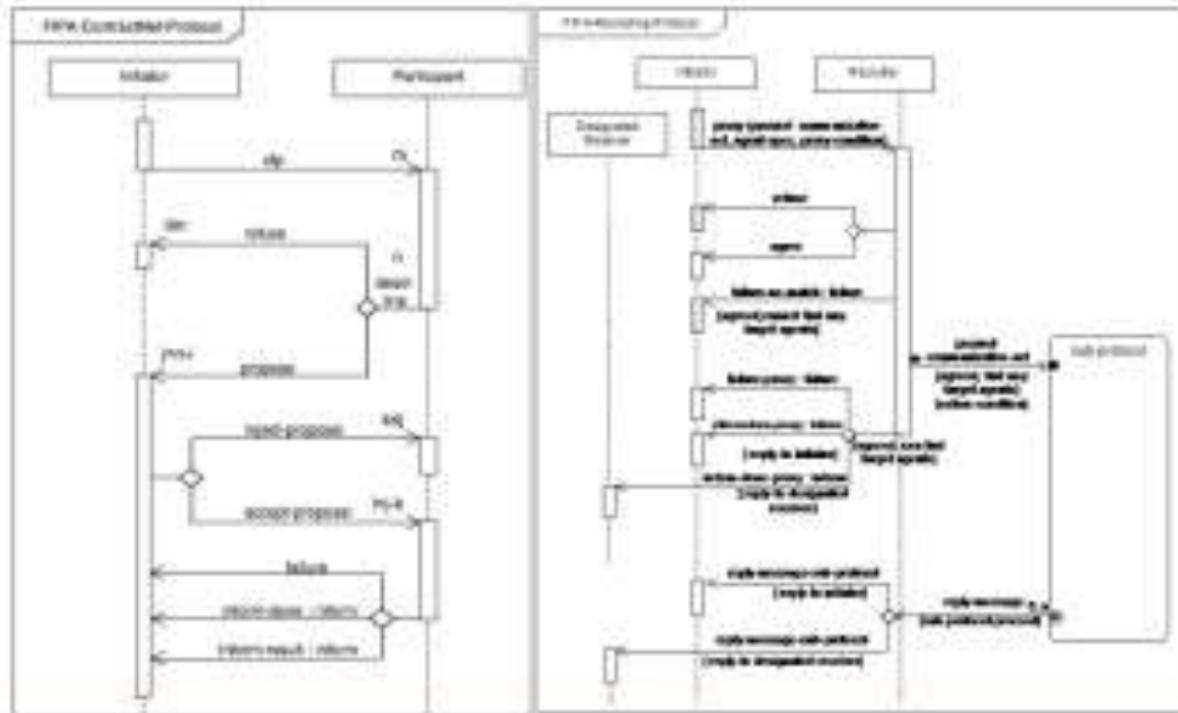


Agent Society Phase – Protocol and Interaction Description

33

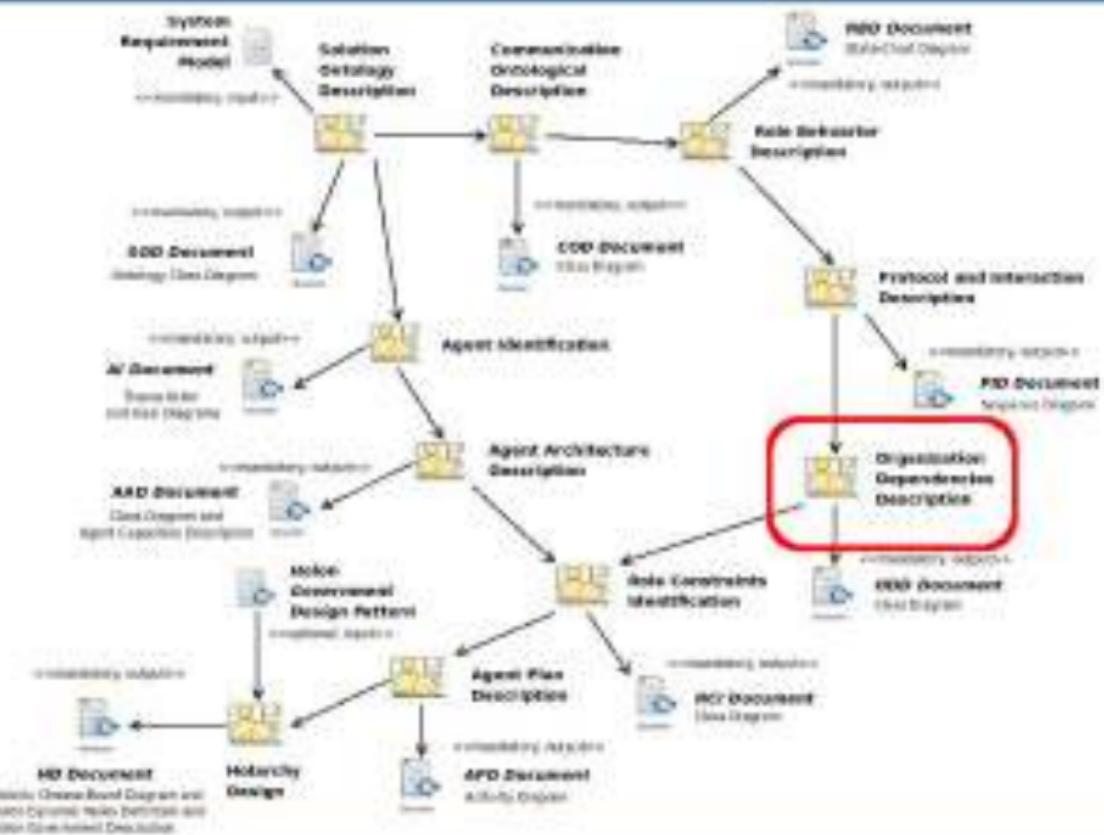


see also <http://fipa.org/repository/ips.php3>



Agent Society Phase – Organization Dependencies Description

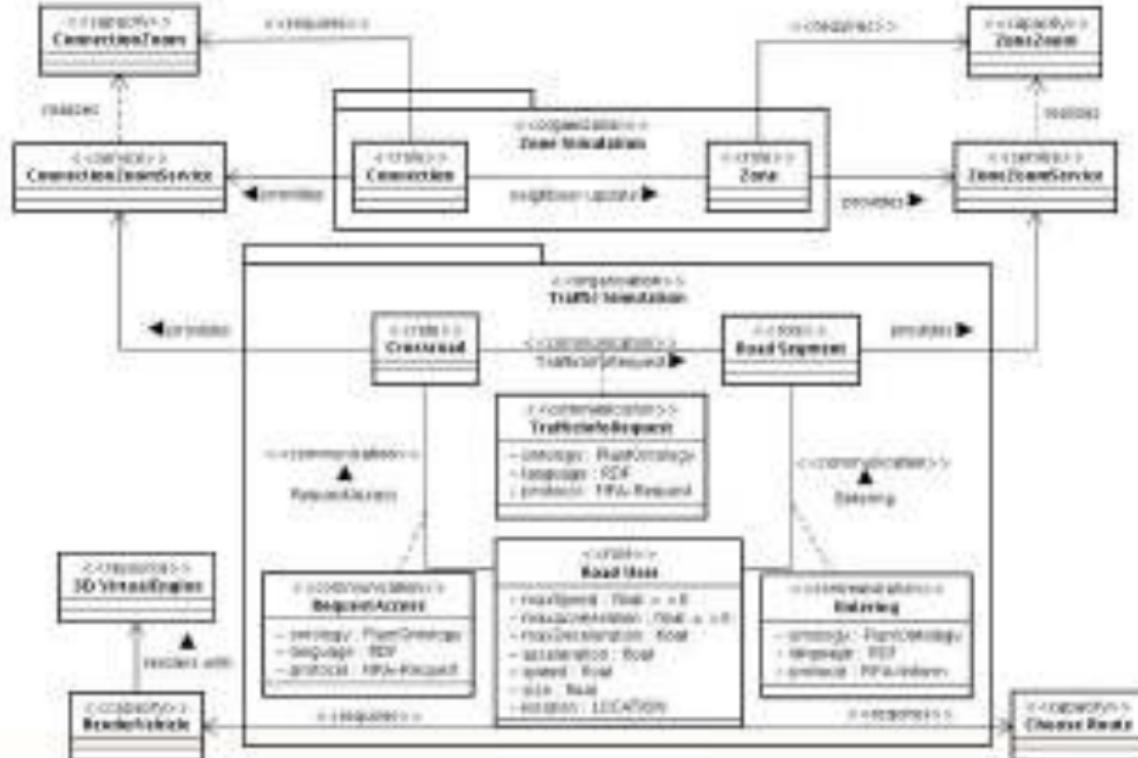
89



MD Document
Holonic Object Diagram and
Mobile Dynamic Holon Interaction and
Holon Governance Description



Organization Dependencies Description

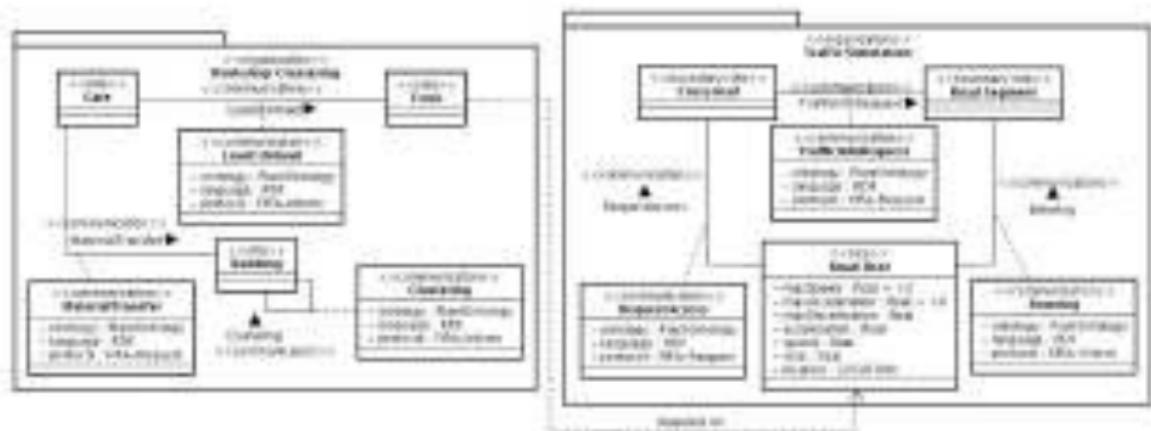




Agent Society Phase – Role Constraints Identification

87

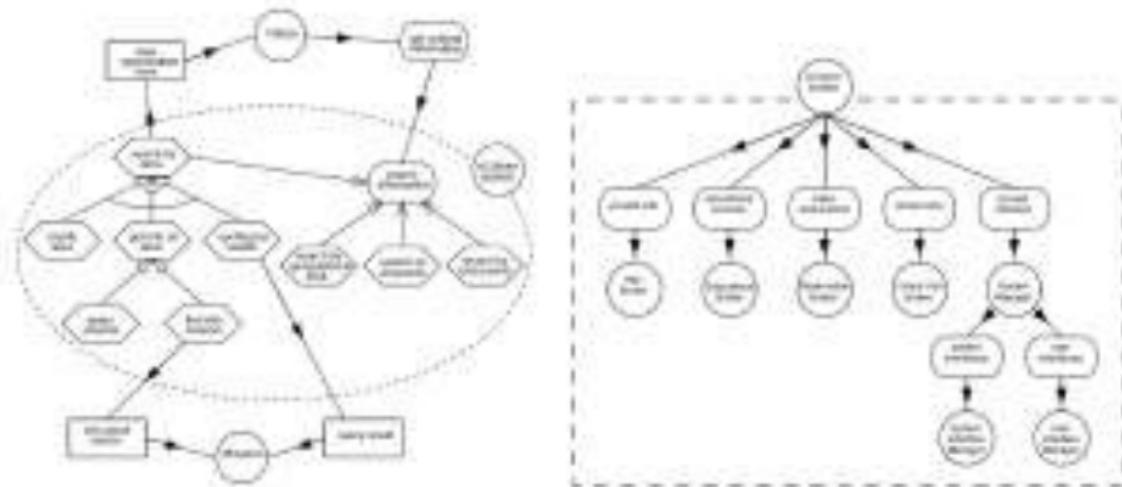




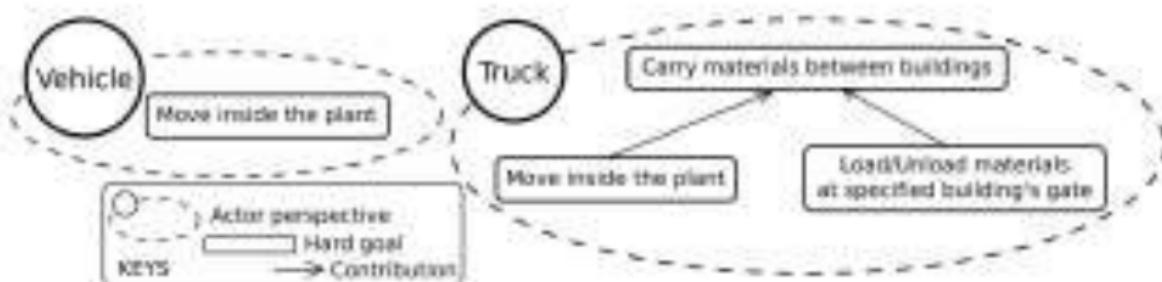
Agent Society Phase – Agent Identification



see also <http://www.troposproject.org>



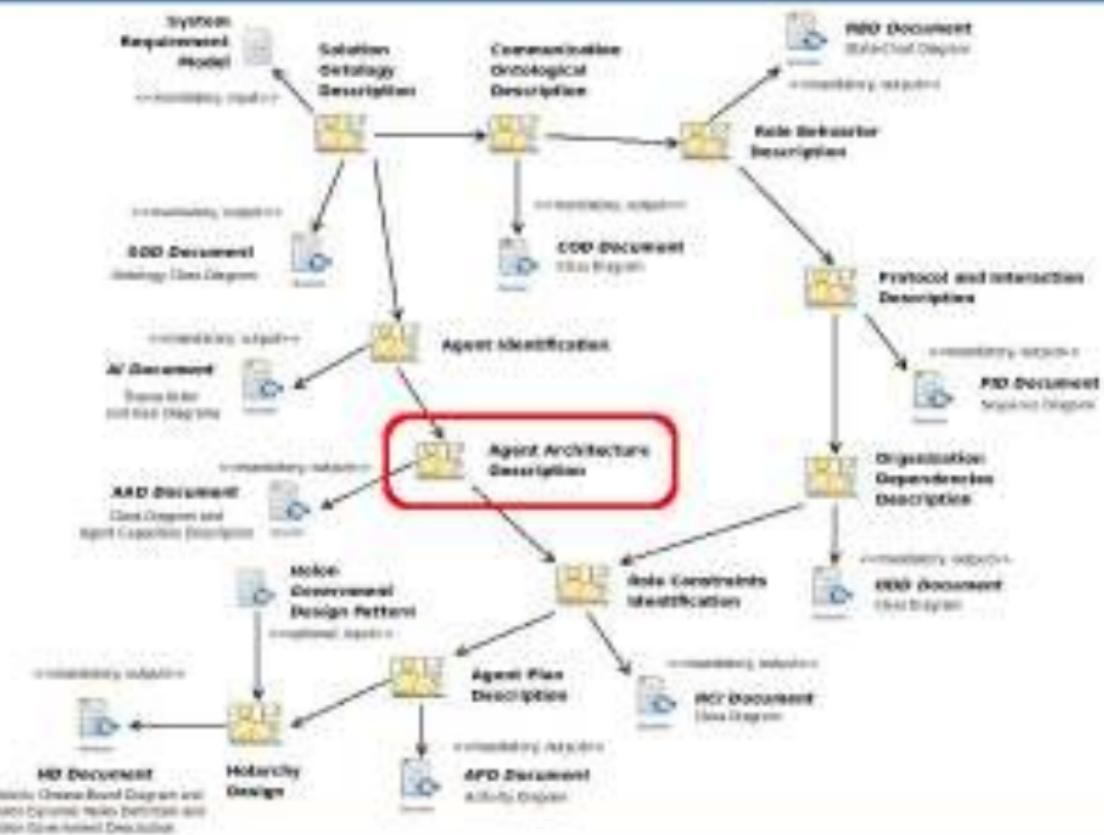
see also <http://www.troposproject.org>



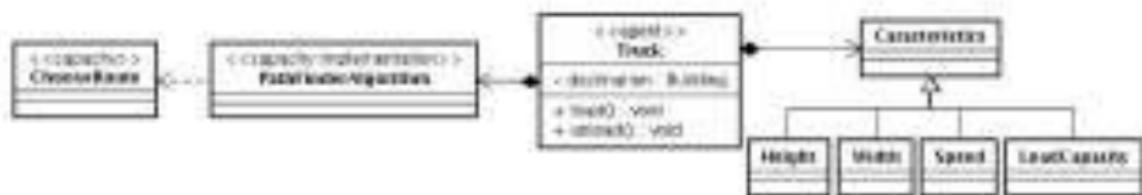


Agent Society Phase – Agent Architecture Description

92

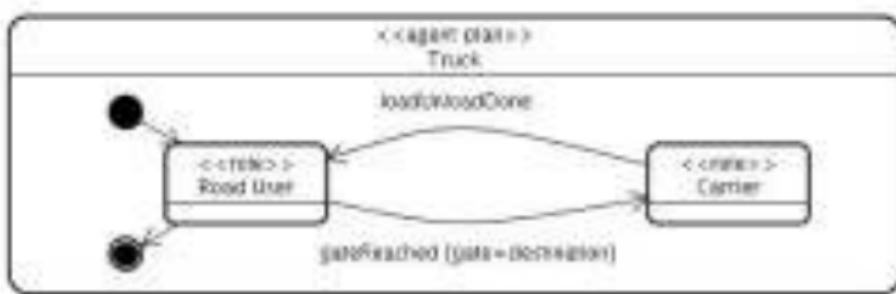


HMT Document
Holistic Design Board Diagram and
Model-Driven Holistic Descriptions and
Agent Environment Description



Agent Society Phase – Agent Plan Description

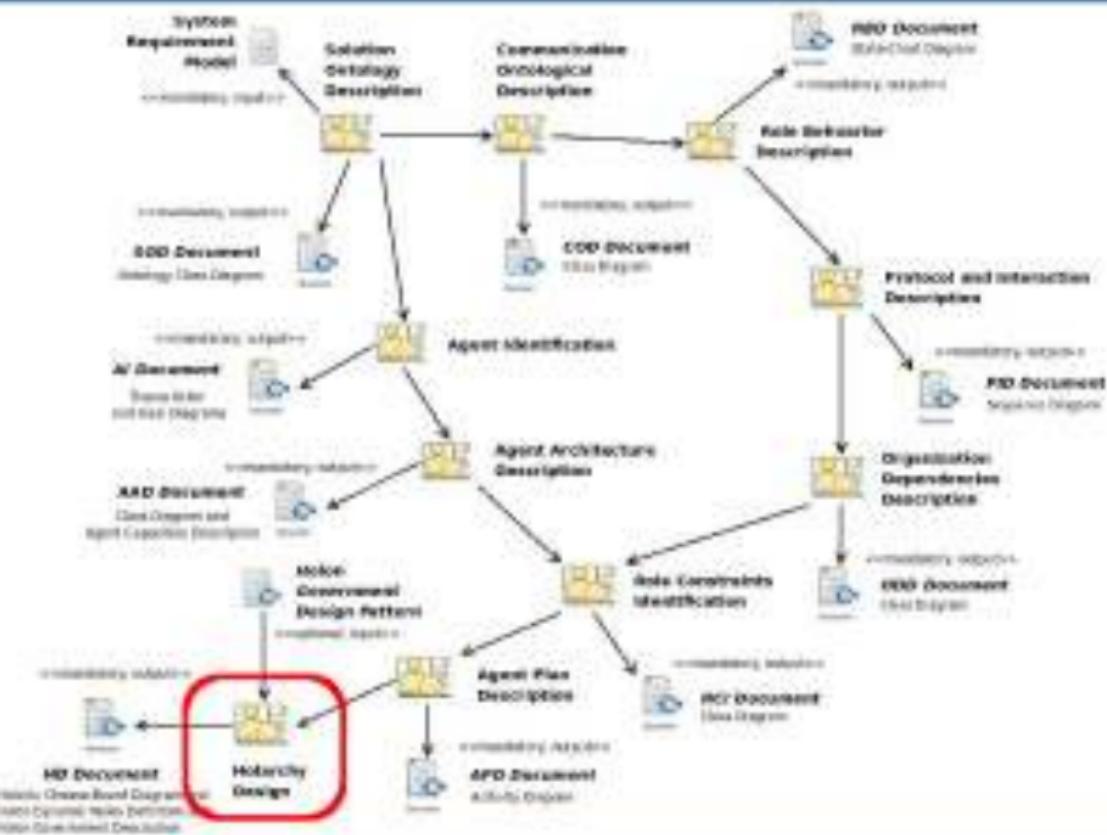


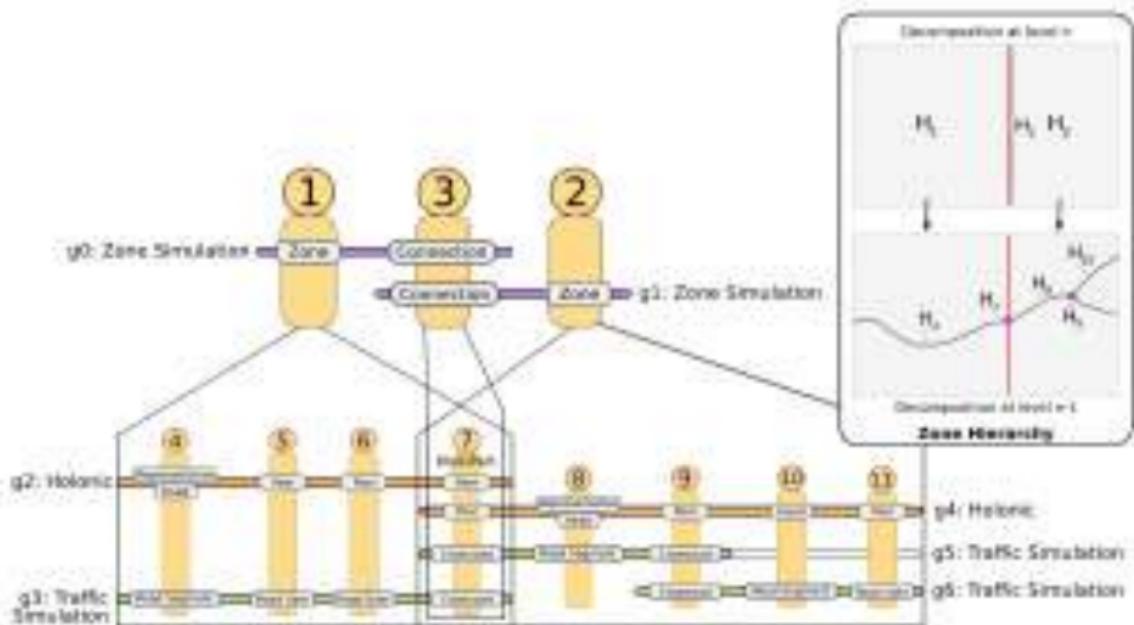




Agent Society Phase - Hierarchy Design

95





Subsection 6

Implementation and Deployment Phase

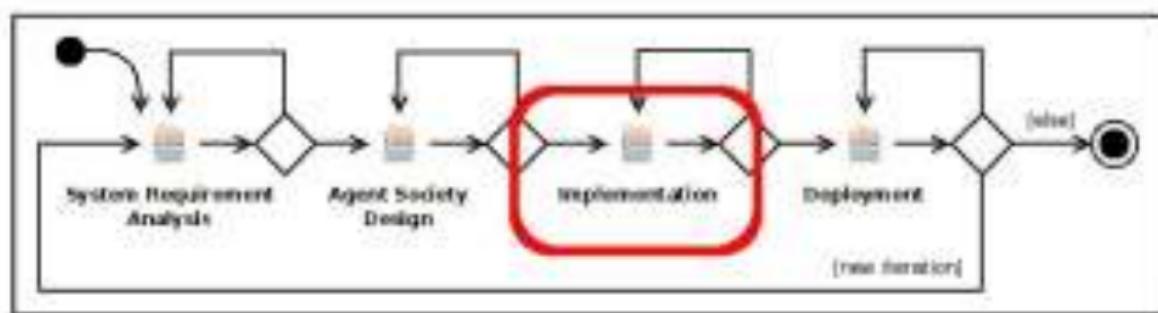


ASPECS

Implementation and Deployment Phase

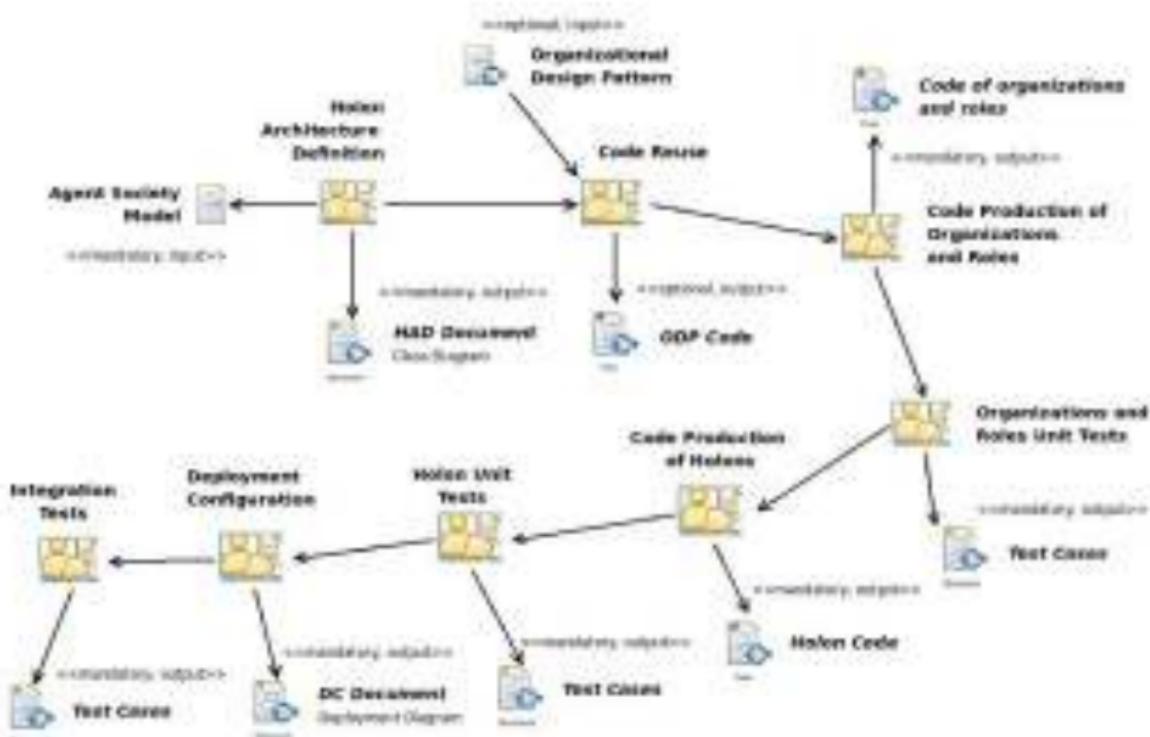
Solution Implementation and Deployment

- implementing and deploying the agent-oriented solution designed in the previous phase by adapting it to the chosen implementation platform
- Describe holon architecture



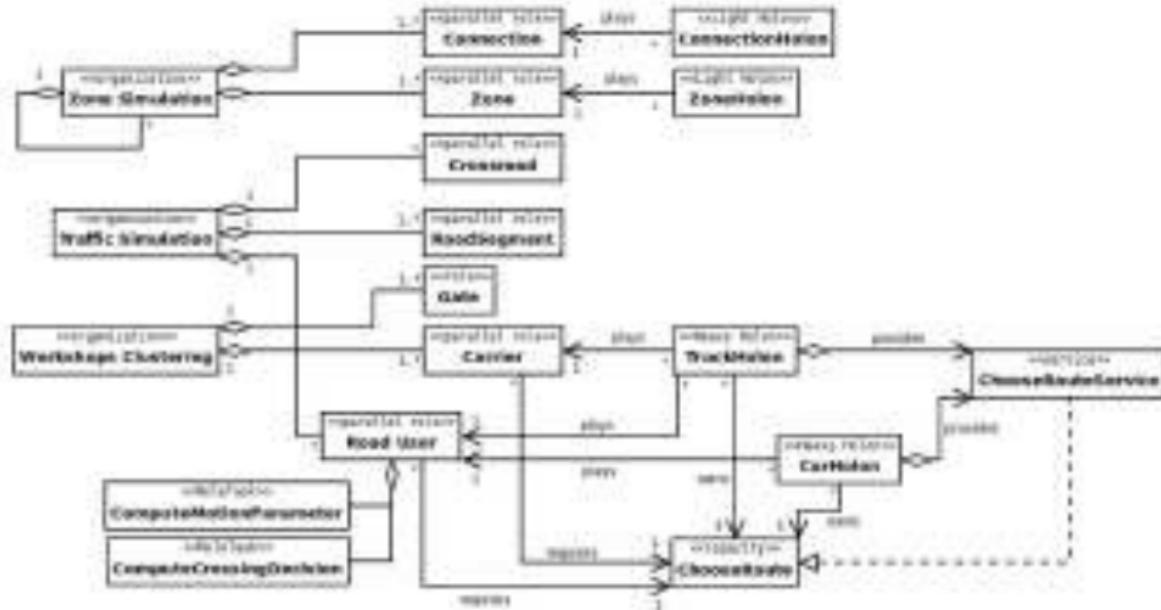
Implementation and Deployment Phase

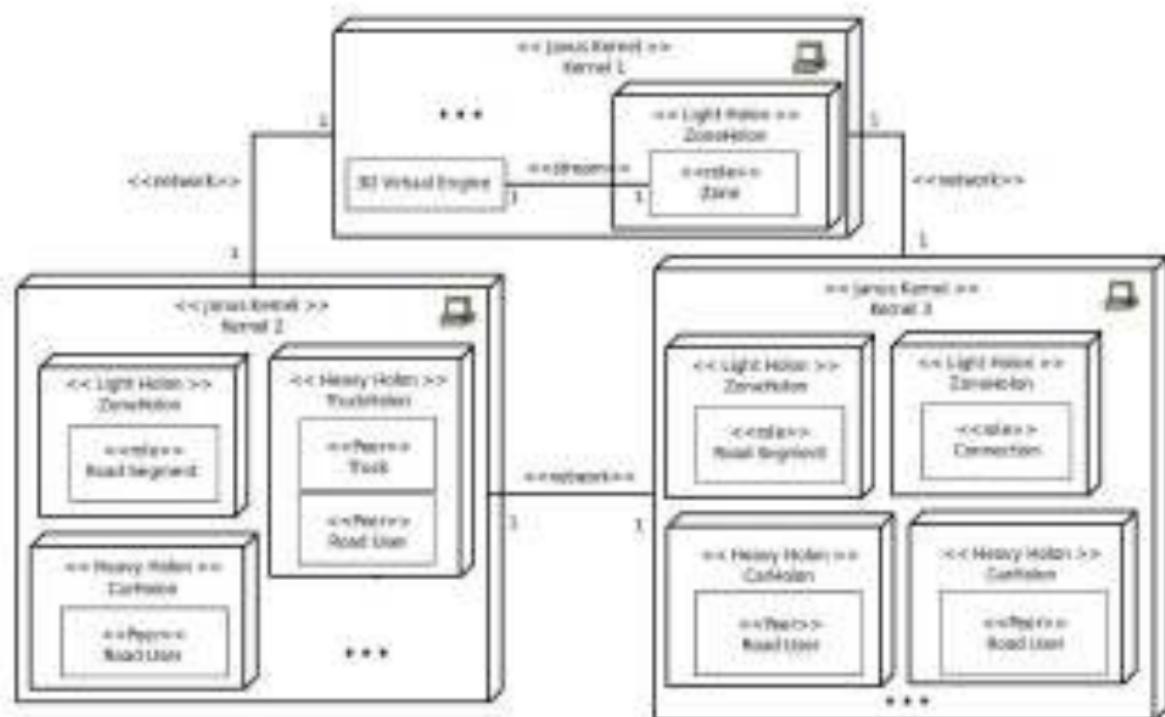
101



Example

Simulation of an Industrial Plant Implementation and Deployment





Section 5

Conclusion

Principles (process-related)

A methodology should cover the entire software development process. In addition, the model of the process must be specified :

- 1 organization and coordination of the various development phases and associated sub-phases (activities and tasks);
- 2 the various workproducts to provide and the associated language to describe them;
- 3 the different persons who are involved in each of these phases and when they must intervene;
- 4 technologies and tools that must be used in each activity;
- 5 the resources involved in each stage of the process.

Each process step must be associated with a set of methodological guidelines describing the work to be done.

Principles (cont.)

- **Concepts** : Approach (MDE, Multi-view, Organizational)?
Does it provide the tools to manage complexity
(Decomposition, Abstraction, Organization)? Composition of agents?
- **System structure** : arbitrary number of levels of abstraction,
wide range of possible structures.
- **Compliance with standards** : in terms of process and product
description (languages).
- **Modeling the System-Environment Relationship**:
- **Modularity, reusability, extensibility** :
 - of models, design pattern.
 - of domain knowledge (Ontology).
- **Associated tools** : IDE, platforms.



Chapter 4: Action Selection

Stéphane GALLAND



- 1 Architecture BDI
- 2 Architecture de Subsomption
- 3 Architecture ANA de P. Maes
- 4 Systèmes Auto-organisé
- 5 Systèmes Auto-organisé: SMA Holonique
- 6 Systèmes Auto-organisé: Système immunitaire

Section 1

Introduction

Comment choisir une action qui va mener à l'accomplissement des buts de l'agent

- De manière autonome
- De manière réactive
- De manière pro-active

Comment éviter les oscillations entre actions

⇒ Architecture de Sélection d'Actions



Plusieurs techniques possibles dépendantes

- du type d'architecture des agents
- des applications



Section 2

Architecture BDI

Subsection 1

Introduction



BDI [Rao, 1995a]

L'architecture BDI est issue de l'étude philosophique du "raisonnement pratique". Et consiste à décider à tout moment :

- Quoi faire ?
- Comment le faire ?

- Quand vous avez une décision à prendre vous examinez les choix qui s'offrent à vous (les options ou **Désirs**) et vous en choisissez un dans lequel vous vous engagez.
- Les choix offerts sont fonctions de vos croyances (**Beliefs**). L'option choisie devient une **Intention**.
- "Tom a un final le lendemain. Il a le choix de faire la fête ou de rester chez lui pour bosser son examen. S'il choisit de faire la fête il va s'investir consciencieusement dans ce choix."



Les hypothèses

- L'intention mène à des actions.
- L'agent va essayer d'agir pour aboutir à la satisfaction de l'intention.
- L'intention va avoir une persistance dans le temps... mais pas trop.
- L'intention est en lien avec mes croyances (Beliefs) futures.



En résumé [Wooldridge, 2003b]

- Les intentions dirigent le raisonnement moyen-fins
- Les intentions contraignent les délibérations futures
- Les intentions persistent
- Les intentions influencent les croyances futures



Subsection 2

Logique modale pour agents BDI



Logique modale pour agents BDI

Logique modale pour agents BDI [Weiss, 1999]

- En logique classique la sémantique d'une formule renvoie vraie ou faux.
- La logique modale propose d'autre type de vérité comme " il est possible que " ou " il est nécessaire que"

Logique modale: 4 modalités

- nécessaire (ce qui ne peut pas ne pas être vrai), noté \Box
- contingent (ce qui peut être vrai ou faux), noté $\neg\Box$
- possible (tout ce qui peut être, sauf impossible), noté \Diamond
- impossible (ce qui ne peut pas ne pas être faux), noté $\neg\Diamond$

- En fait on considère que la vérité d'une formule dépend d'un contexte qu'on appelle monde possible
- Une des utilisations de ces mondes possibles est de les considérer comme un historique ~ Logique du temps arborescent

Opérateurs temporels: les quantificateurs

- $A\phi$ sur toutes les branches ϕ (inévitablement)
- $E\phi$ sur certaines les branches ϕ (possiblement)



BDI Logic

- $(Bel \ i \ \phi)$ i croit que ϕ est vraie
- $(Des \ i \ \phi)$ i désire que ϕ est vraie
- $(Int \ i \ \phi)$ i a l'intention de réaliser ϕ

- $(Des \ i \ \phi) \implies (Bel \ i \ \phi)$
- $(Int \ i \ \phi) \implies (Des \ i \ \phi)$
- et donc $(Int \ i \ \phi) \implies (Bel \ i \ \phi)$

- B relation d'accessibilité des croyances ou opérateur modal \Box . Pour chaque agent et chaque moment tout ce que l'agent croit possible.
- D ou désirs associe à tout moment ce que veut l'agent. Un agent a un désir ϕ à un moment ssi ϕ est vraie dans les mondes D-accessibles de l'agent.
- I ou intentions associe à tout moment pour chaque agent les conditions vraies pour les "futurs" de l'agent ce qui définit un ensemble de chemins sélectionnés/préférés.

Subsection 3

Principes des agents BDI

Principes des agents BDI



- Des croyances courantes (Beliefs)
- une fonction de révision des croyances (brf)
- une fonction de génération d'options (options)
- des options courantes qui représentent les actions possibles
- une fonction de filtre qui représente le processus de délibération de l'agent et qui calcule les intentions de l'agent
- un ensemble d'intentions courantes
- une fonction de sélection d'actions (execute) qui détermine l'action à exécuter en fonction des intentions courantes.

```
B:=B0;
l:=l0;
tant que True faire
    p:=getPercept();
    B:=brf(B,p);
    D:=options(D,l);
    l:=filter(B,D,l);
    π:=plan(B,l);
    execute(π);
fin
```

Algorithm 1: Algorithme naïf d'un agent BDI

Beliefs	Desires	Intentions	exécuté	accompli
verre	-	-	-	-
verre	boire	-	-	g-add(boire)
verre	boire	{ soda , boisson }	-	g-add(soda)
aucun- soda	boire	-	frigo	frigo, g- add(boire)
verre	boire	{ boisson }	prendre- boisson	prendre- boisson
boire	-	-	boire	boire

- Les intentions posent le problème de comment les réaliser
- Les intentions fournissent un filtre pour adopter d'autres intentions
- S'il y a échec lors de la réalisation d'une intention l'agent retente
- Les agents croient que leurs intentions sont possibles $E\Diamond\phi$
- Les agents ne croient pas qu'ils ne peuvent pas réussir leurs intentions $A\Box\neg\phi$
- Les agents croient que sous certaines conditions ils peuvent réussir leurs intentions $A\Diamond\phi$
- Les agents ne s'attendent pas à tous les effets de leur intention

```
B:=B0;
l:=l0;
tant que True faire
    p:=getPercept();
    B:=brf(B,p);
    D:=options(D,l);
    l:=filter(B,D,l);
    π:=plan(B,l);
    execute(π);
fin
```

*tant que min-val(v) <=
minVal(l,B) ou
maxVal(l,B) >= val(v)*

*v := head(v);
available(v),
go-to(l,v),
p := getPercept(),
B := brf(B,p),
D := consider(l,B) alors
Options(D) :=
filter(p,D))*

*si non sound(π,B)
alors π := plan(B,l)*

```

B:=B0;
l:=l0;
tant que True faire
  p:=getPercept();
  B:=brf(B,p);
  D:=options(D,l);
  l:=filter(B,D,l);
  π:=plan(B,l);
  execute(π);
fin

```



```

tant que non vide(π) ou
succès(l,B) ou
impossible(l,B) faire
  α:=head(π);
  execute(α);
  π:=tail(π);
  p:=getPercept();
  B:=brf(B,p);
  si reconsider(l,B) alors
    D:=options(π);
    l:=filter(B,D,l);
    ;
  si non sound(π,l,B)
  alors π:=plan(B,l);
  ;
fin

```

- Architecture pour SMA cognitifs
- Comportements fortement prédictibles
- Problèmes de sémantique et d'efficacité
- Complexité d'analyse et conception

Subsection 4

Exemple d'implémentation d'agents BDI

IRMA

Exemple d'implémentation d'agents BDI

IRMA [Chaib-Draa, 2003]

- Une librairie de plans
- Beliefs ou croyances
- Desires sous forme de tâches
- Intentions sous-ensemble de désirs que l'agent a choisi et sur lesquels il s'est engagé

IRMA

- un moteur d'inférence pour raisonner sur le monde
- un analyseur moyen-fin détermine quels plans sont les mieux adaptés pour réaliser les intentions
- un analyseur d'opportunité monitore l'environnement et peut générer de nouvelles options
- un processus de filtrage détermine quelles options sont compatibles avec les intentions courantes
- un processus de délibération pour décider quelles sont les intentions à adopter

Subsection 5

Conclusion

Conclusion



Architecture BDI

- Une des architectures agent les plus utilisées jusque là
- Lien intuitif avec actes de langages
- Formalisation de nombreux aspects

Section 3

Architecture de Subsomption



Architecture de Subsomption [Brooks, 1986]

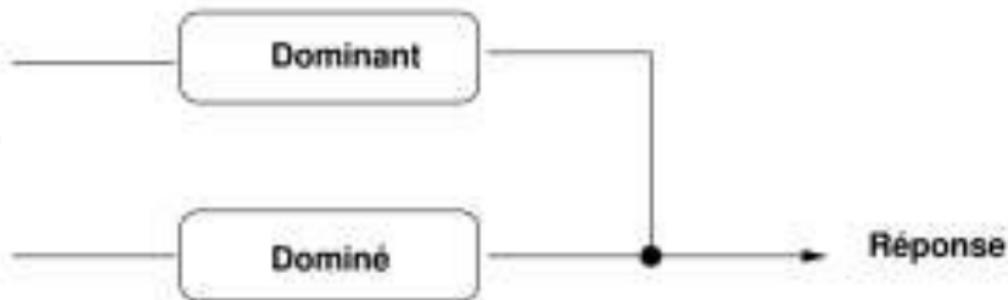
- Architecture modulaire verticale
- Interaction par rapport dominance/dominé

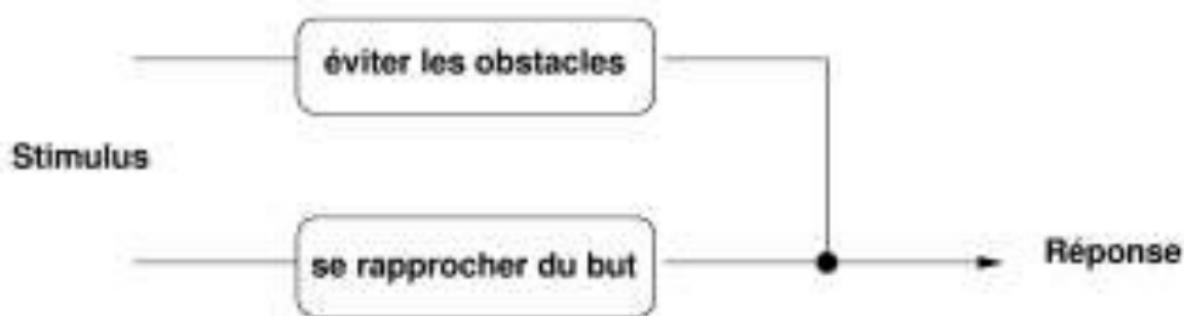
Stimulus

Dominant

Dominé

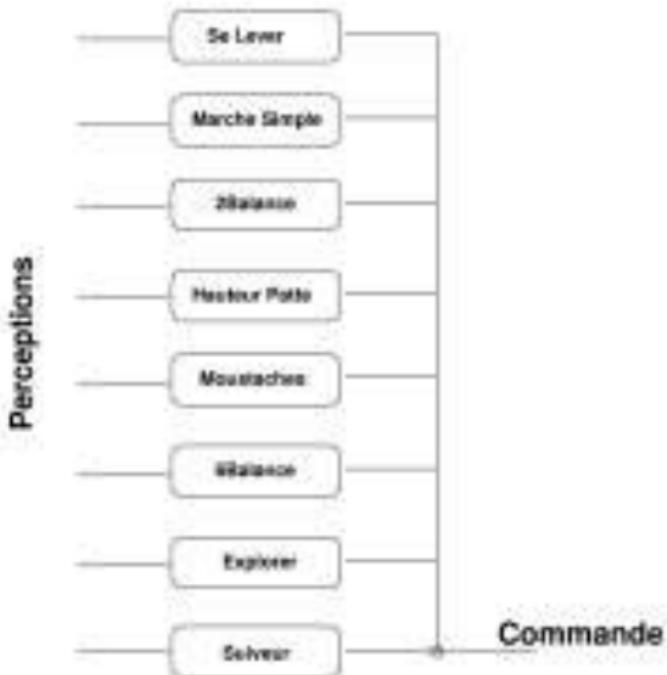
Réponse





MIT AI Lab, February 1989 [Brooks, 1989]







Section 4

Architecture ANA de P. Maes

Agent Network Architecture (ANA) [Maes, 1989]

Un module i est défini par un quadruplet $(c_i, a_i, d_i, \alpha_i)$. Où

- c_i est une liste de pré-conditions à remplir avant d'activer le module
- a_i et d_i sont les effets attendus du module sous la forme de listes d'ajouts et de retraits
- α_i est le niveau d'activation du module.

3.2.

- Il y a un arc successeur d'un module x vers un module y pour tout $p \in a_x \cap c_y$
- Il y a un arc prédecesseur d'un module x vers un module y pour tout $p \in c_x \cap a_y$
- il y a un arc inhibiteur d'un module x à un module y pour tout $p \in c_x \cap d_y$

- Activation par les perceptions : ajout d'énergie aux modules qui matchent partiellement l'état de l'environnement
- Activation par les buts : ajout d'énergie aux modules qui accomplissent un des buts de l'agent (add-list)
- Inhibition par buts protégés : on enlève de l'énergie à un module si un des buts déjà réalisé fait partie de sa liste des retraits (delete-list)

- Activation des successeurs : on augmente l'énergie des successeurs d'un module actif
- Activation des prédecesseurs : on augmente l'énergie des prédecesseurs d'un module non actif
- Inhibition des modules conflictuels : chaque module diminue l'énergie des modules qui sont en conflit avec lui

tant que True faire

Impacter niveaux d'action par env et buts ;

Propagation d'énergie par les arcs du graphe;

Degrader énergie;

si \exists_1 module exécutable et niveau d'énergie \geq seuil d'activation **alors** ce module devient actif;

remise à 0 du niveau d'énergie de ce module;

si deux modules répondent à ces conditions **alors** choix aléatoire;

remise à 0 du niveau d'énergie de ce module;

si aucun module **alors** seuil diminué de 10% ;

fin

- θ : le seuil pour qu'un module devienne actif
- ϕ : l'énergie qu'une proposition vraie observée injecte dans le réseau
- γ : l'énergie qu'un but injecte dans le réseau
- δ : l'énergie qu'un but protégé enlève du réseau



" Soit un robot avec deux mains qui doit se peindre lui-même avec un spray et poncer une planche. Une fois peint le robot n'est plus opérationnel."

- **PICK-UP-SPRAYER**

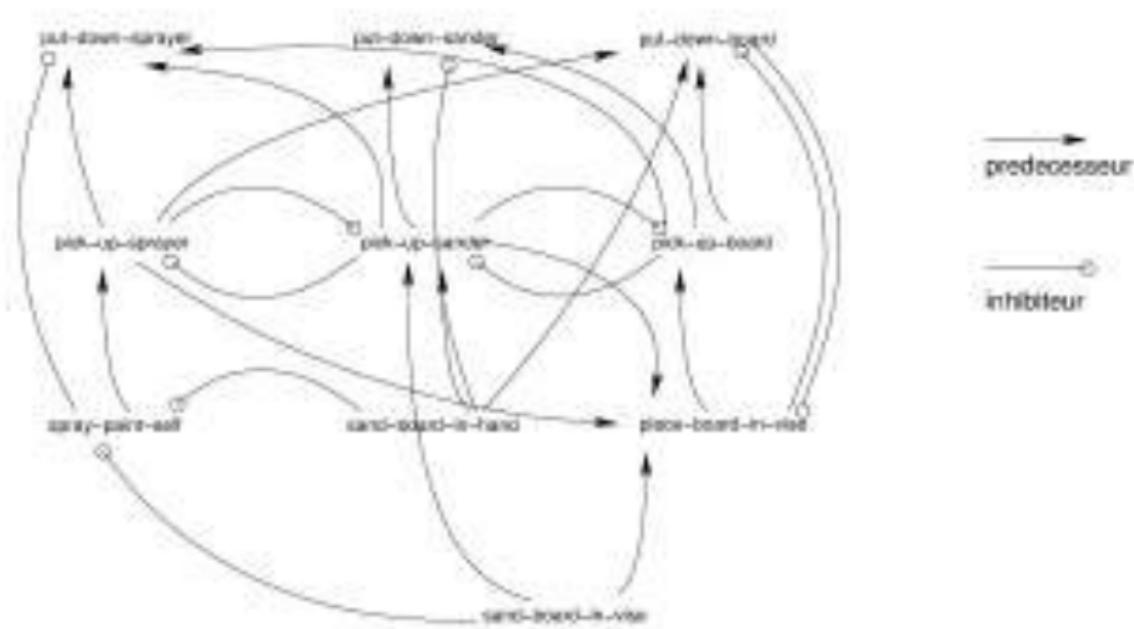
condition-list : sprayer-somewhere hand-is-empty
add-list : sprayer-in-hand
delete-list : sprayer-somewhere hand-is-empty

- **PICK-UP-SANDER**

condition-list : sander-somewhere hand-is-empty
add-list : sander-in-hand
delete-list : sander-somewhere hand-is-empty

- **PICK-UP-BOARD**

condition-list : board-somewhere hand-is-empty
add-list : board-in-hand
delete-list : board-somewhere hand-is-empty



- Architecture de sélection d'actions connectioniste et délibérative ⇒ Beaucoup de similarités avec les réseaux de neurones
- Apprentissage des paramètres possible
- Problème de choix entre actions avec niveau d'énergie équivalent
- Difficulté de prise en compte de buts multiples

Section 5

Systèmes Auto-organisé

Systèmes Auto-organisé

- Apparition dynamique de structures intéressantes
- Etat stable vus comme des solutions (Eco-résolution)
- Modélisation intentionnelle ⇒ on va provoquer l'émergence de structures organisationnelles pertinentes

Plusieurs niveaux d'interprétation/modélisation nécessaire pour expliquer l'émergence en terme d'organisations.

"Les choses n'existent que de la façon dont on les perçoit à une certaine échelle."

Benoît Mandelbrot

Section 6

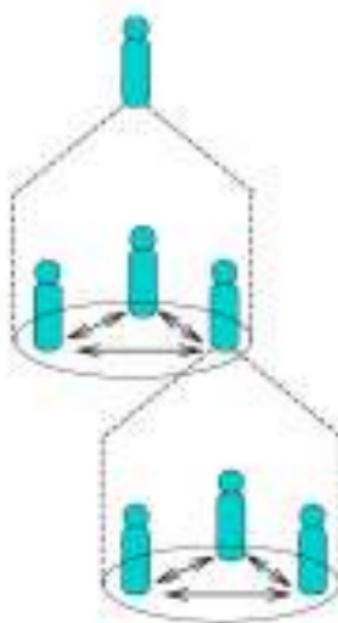
Systèmes Auto-organisé: SMA Holonique

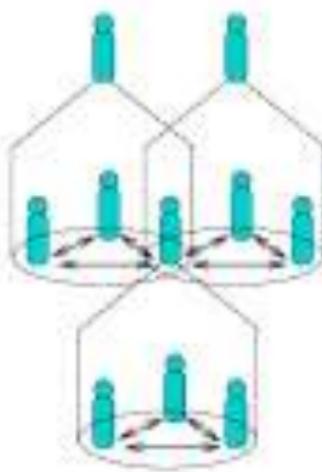


Définition

Structures naturelles ou artificielles qui ne sont ni tous ni parties au sens absolu. (Kostler, 1967)

- Chaque holon peut être composé d'un ensemble de sous-holons et/ou être membre d'un holon qui le contient
- Cette holarchie définit une structure organisationnelle apte à réagir de manière dynamique aux changements de l'environnement
- Chaque holon utilise au mieux les ressources et répond au mieux aux buts du système





Subsection 1

Framework générique

1/1



Framework générique pour les SMAH

Comment les sous-holons sont organisés pour former un super-holon ? trois solutions [Gerber, 1999a]

•

•

•

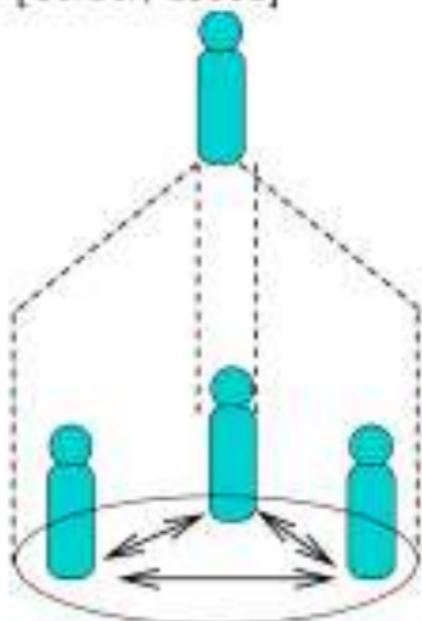
Comment les sous-holons sont organisés pour former un super-holon ? trois solutions [Gerber, 1999a]

- Fusion
-
-



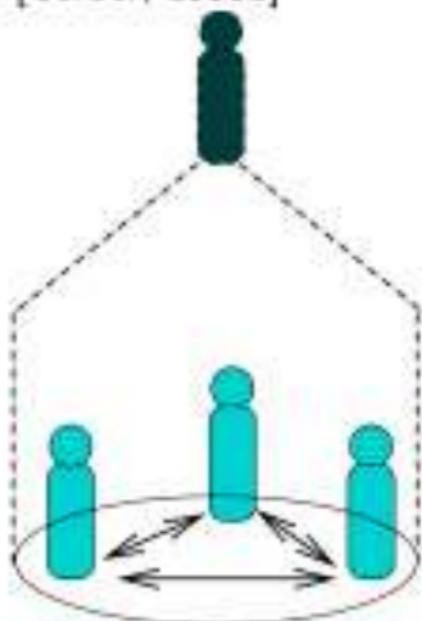
Comment les sous-holons sont organisés pour former un super-holon ? trois solutions [Gerber, 1999a]

- Fusion
- Groupes modérés
-



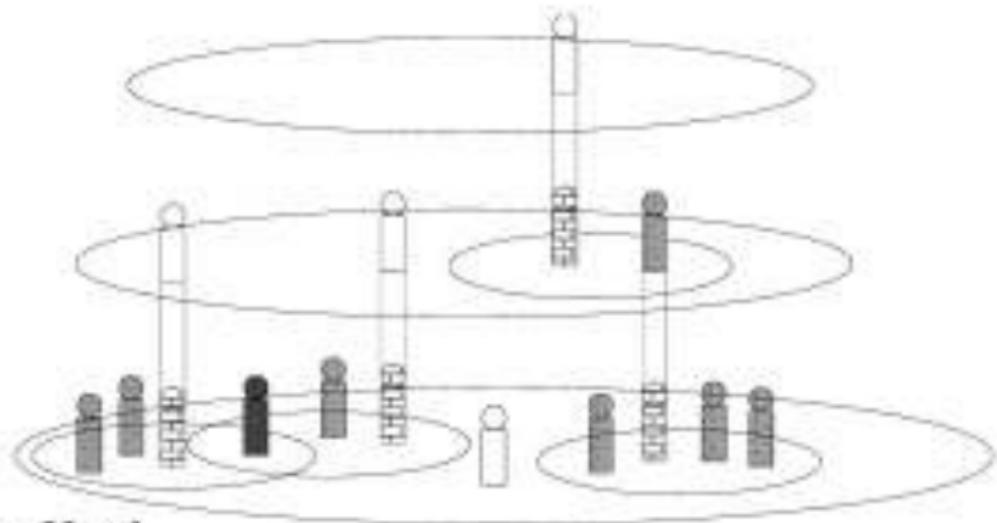
Comment les sous-holons sont organisés pour former un super-holon ? trois solutions [Gerber, 1999a]

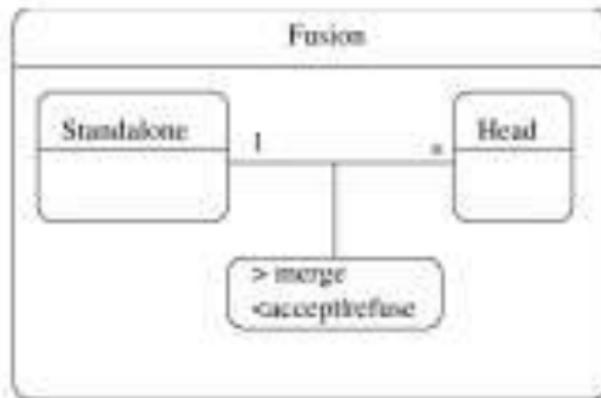
- Fusion
- Groupes modérés
- Fédération

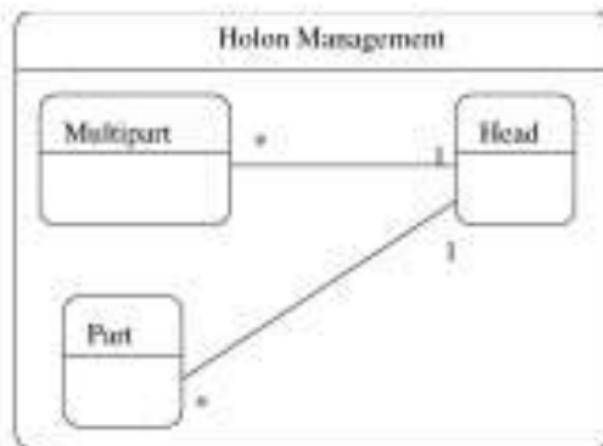


[Rodriguez, 2003]

- Standalone : initialisation, un holon "tout seul"
- Head : représentant d'un holon à l'extérieur, gère le holon
- Part : membre d'un holon
- Multipart : membre de plusieurs holons







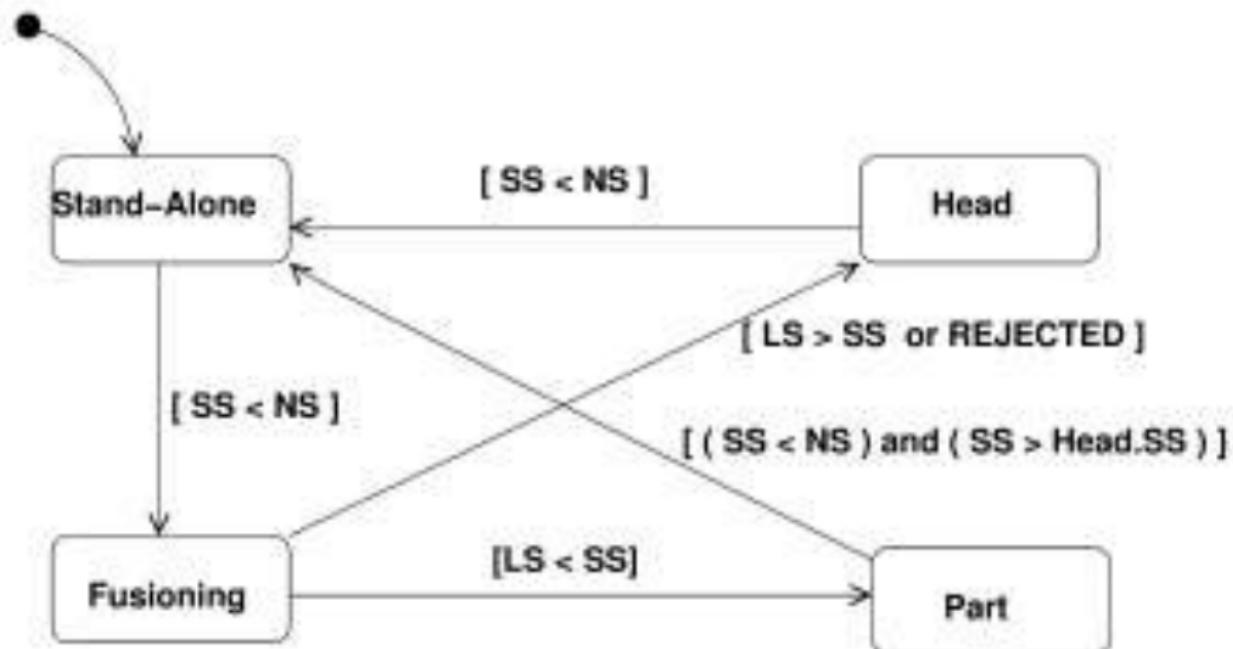
- **Self Satisfaction (SS_i):** produite pour un holon i par ses actions.
- **Collaborative Satisfaction (CS_i^H):** produite pour un holon i par ses collaborations avec les autres membres du super-holon H .
- **Accumulative Satisfaction (AS_i):** produite pour un holon i par ses collaborations avec les membres des super-holons.

$$AS_i = \sum_p CS_i^p \quad \forall p \in superholon(i) \quad (1)$$

- Instant Satisfaction (IS_i): Current satisfaction of holon i

$$\forall i \in HMAS \quad IS_i = \begin{cases} CS_i + SS_i & R_i = Part|Head \\ AS_i + SS_i & R_i = MultiPart \\ SS_i & R_i = Stand - Alone \end{cases} \quad (2)$$

R_i : role joué par le holon i .

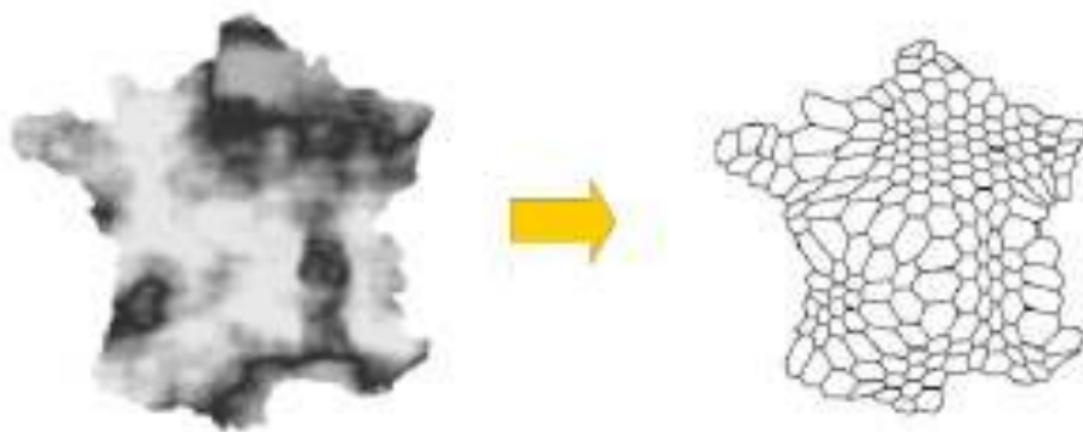


- Critère de décision pour le choix du holon avec qui fusionner
- Heuristique pour l'émergence de structures organisationnelles
- Attrarance/complémentarité de services/ressources

Subsection 2

Exemple : le maillage adaptatif

Exemple : le maillage adaptatif

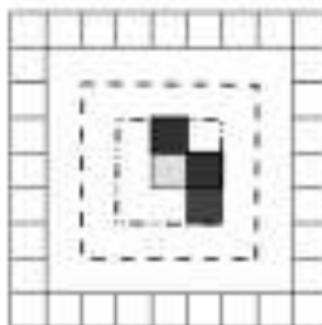


Satisfaction

- NS = couverture maximale des stations émettrices
- SS = ressource d'un holon
- Head.SS = ressource d'une maille

Affinité

- Affinité liée à la distance (contrainte géométrique)
On conserve des mailles convexes
- Affinité liée à la ressource (couverture)
On tente d'obtenir des mailles de ressource homogène



■ Head

■ Parts

■ Holon trying to fusion

--- Acceptance distance

----- Average Distance

Subsection 3

SMA Holonique: Conclusion

- Framework générique pour SMA auto-organisés
- Difficulté de conception de la distance et du paramétrage des satisfactions

Section 7

Systèmes Auto-organisé: Système immunitaire

Subsection 1

Introduction



Système immunitaire [Farmer, 1986]

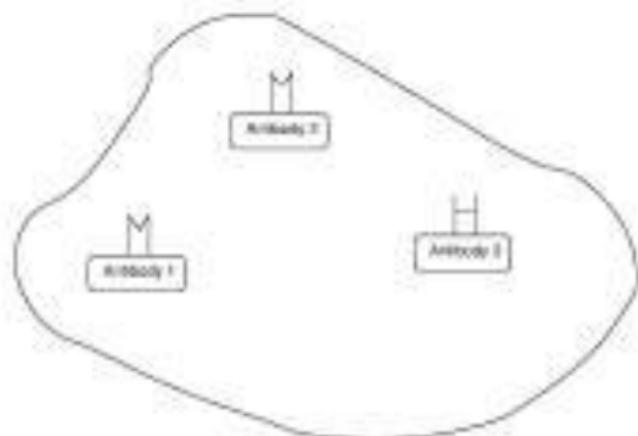
- parallèle,
- distribué,
- système complexe (non prévisible).

- Collectif : La réponse du SI est collective c'est l'anticorps dont le paratope correspond le mieux à l'épitope de l'antigène. Le paratope est le site de reconnaissance de l'anticorps qui assure la fonction de reconnaissance de l'antigène. Chaque paratope reconnaît de façon spécifique une partie de l'épitope d'un antigène.
- Auto-régulé : au travers de stimulation-inhibition la population d'anticorps évolue selon les agressions.
- Auto-organisation : la structure du SI varie selon les évolutions de l'environnement.

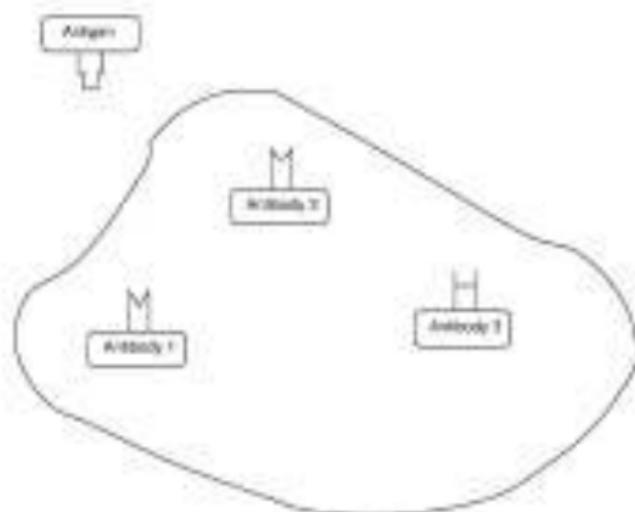
Le SI fonctionne selon trois grands principes

- Reconnaissance (intrus/non intrus)
- Apprentissage
- Mémoire associative

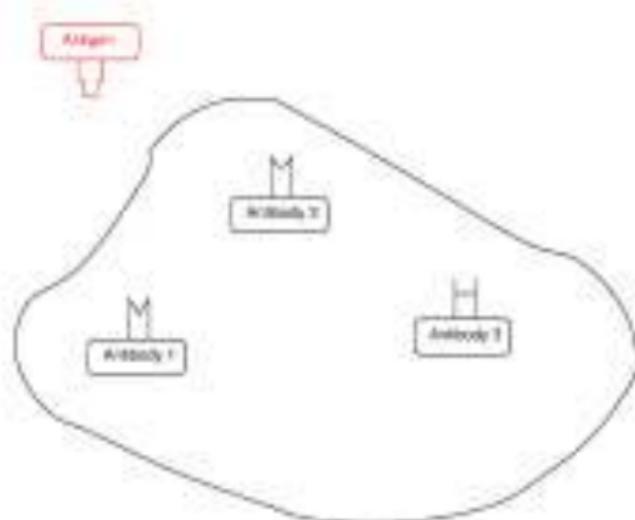
Système
Immunitaire
"au repos"



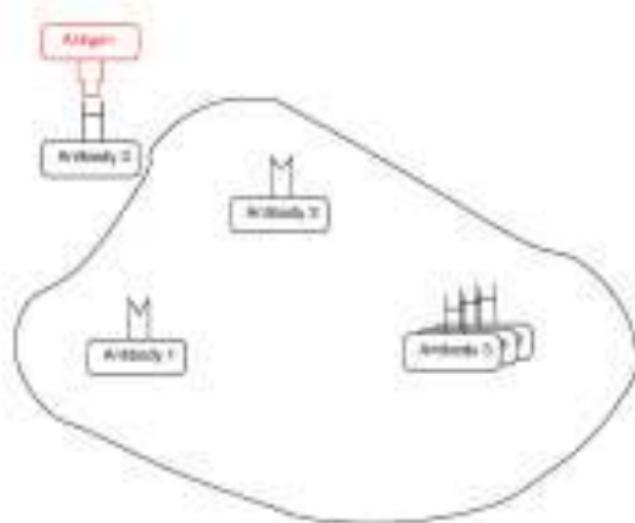
Un Antigène arrive



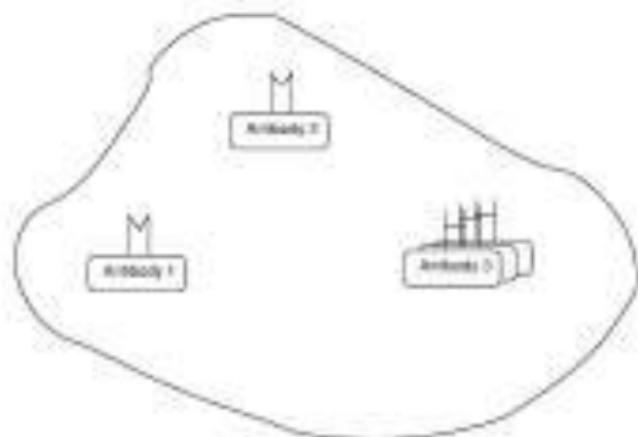
I'Antigène
est reconnu
(self/non
self)



Mémoire Associative



Immune
System "au
repos" et
amélioré
(apprentissage)



Subsection 2

Réseau idéotypique



Réseau idéotypique



Réseau idéotypique [Jerne, 1974]

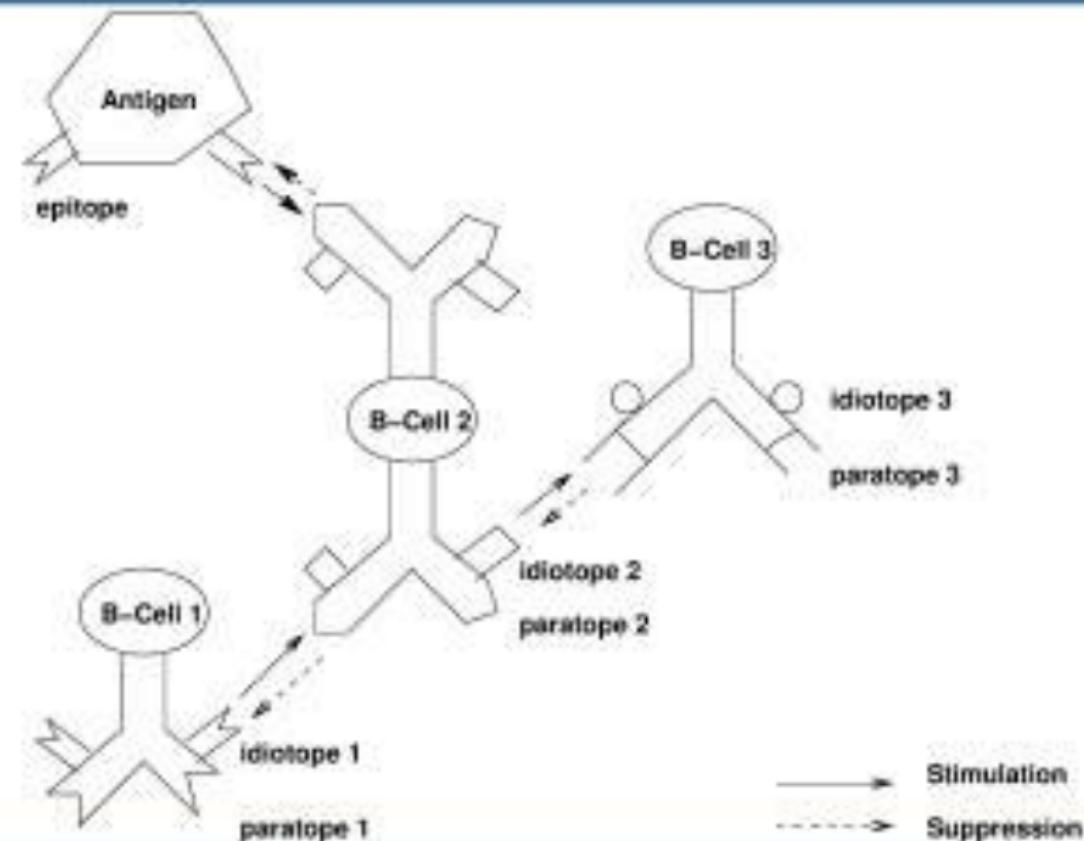
- Les anticorps communiquent par des liens de stimulation-inhibition et identifie les intrusions (antigènes)
- L'idéotope d'un anticorps est reconnu par les autres anticorps comme un antigène.

Subsection 3

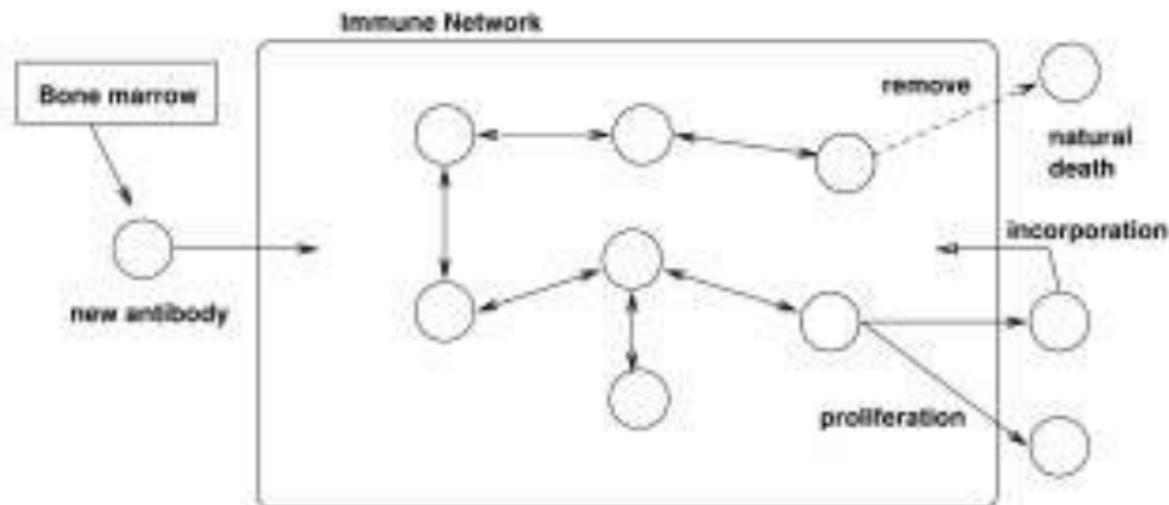
Structures

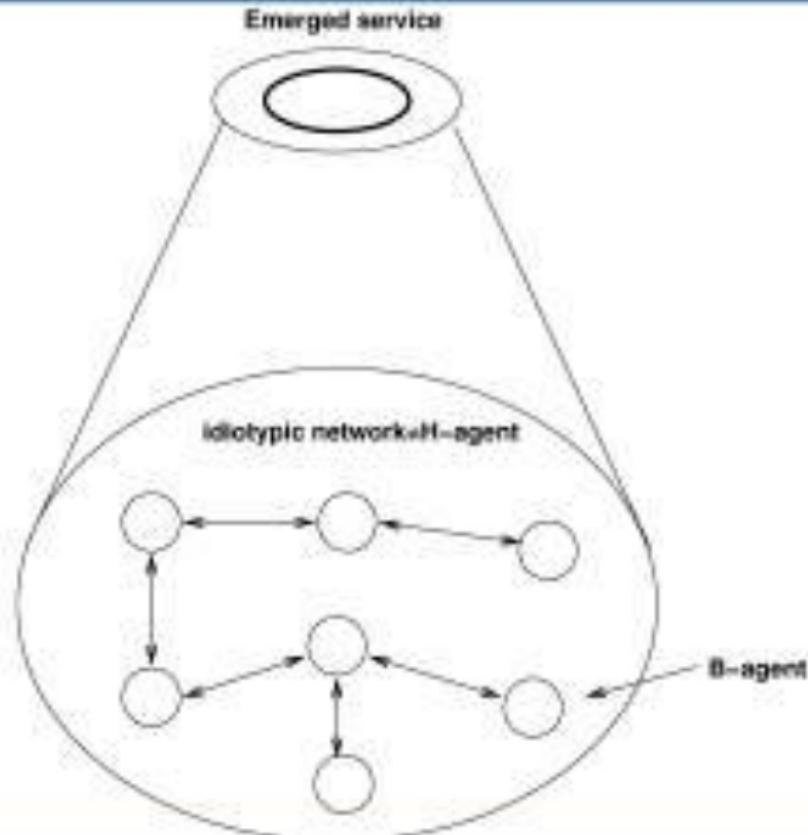


Structure du Réseau idéotypique



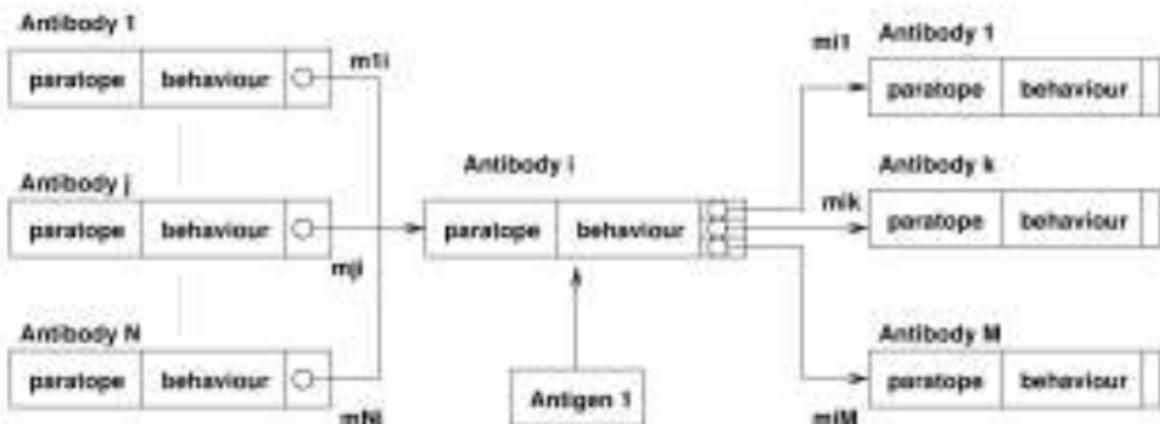
- Mécanisme d'auto-régulation.
- Maintenance d'un ensemble approprié de cellules.
- Optimisation de la réaction à l'environnement.





[Watanabe, 1999]

précondition	attributs, code, données, comportement	références au à d'autres B-Agent à stimuler ou inhiber avec le degré correspondant (affinité)
Paratope	Spécification de l'agent	Idéotope



- Ab_i est choisi, parmi les $\{Ab_{j_1}, \dots, Ab_{j_n}\}$ qui sont sélectionnés (concentration > seuil)
- Ab_i reçoit des récompenses : les affinités de $\{Ab_{j_1}, \dots, Ab_{j_n}\}$ vers Ab_i augmentent
- Ab_i reçoit des pénalités : les affinités de Ab_i vers $\{Ab_{j_1}, \dots, Ab_{j_n}\}$ augmentent

$$m_{12} = \frac{T_p^{Ab1} + T_r^{Ab2}}{T_{Ab1}^{Ab2}} \quad (3)$$

- T_p^{Ab1} is the number of times penalty reinforcement signals were received when $Ab1$ was selected.
- T_r^{Ab2} is the number of times reward reinforcement signals were received when $Ab2$ was selected.
- T_{Ab1}^{Ab2} is the number of times both, $Ab1$ and $Ab2$, have reacted to specific antigens.

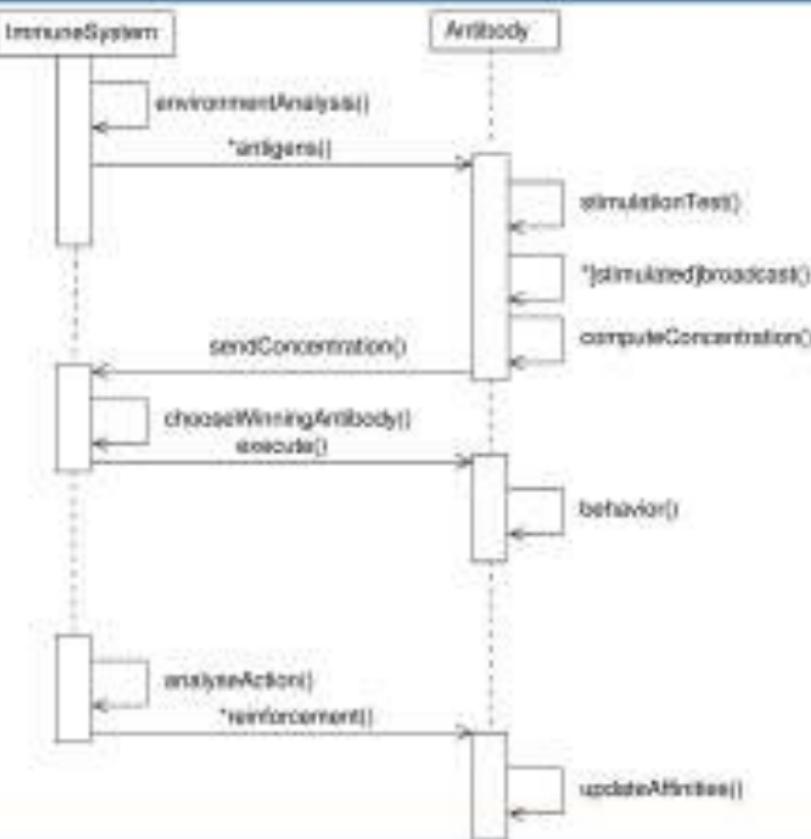
$$\frac{dA_i(t)}{dt} = (\alpha \frac{1}{N} \sum_{j=1}^N m_{ji} a_j(t) - \alpha \frac{1}{M} \sum_{k=1}^M m_{ik} a_k(t) + \beta m_i - k_i) a_i(t)$$

stimulation par anticorps ayant une affinité positive	inhibition par anticorps ayant une affinité négative	$m_i = 0 \text{---} 1$ (stimulé ou non)	facteur de dissipation
---	--	--	------------------------

La valeur calculée est rapporté à un intervalle [0,1]

$$a_i(t) = \frac{1}{1 + \exp(0.5 - A_i(t))}$$

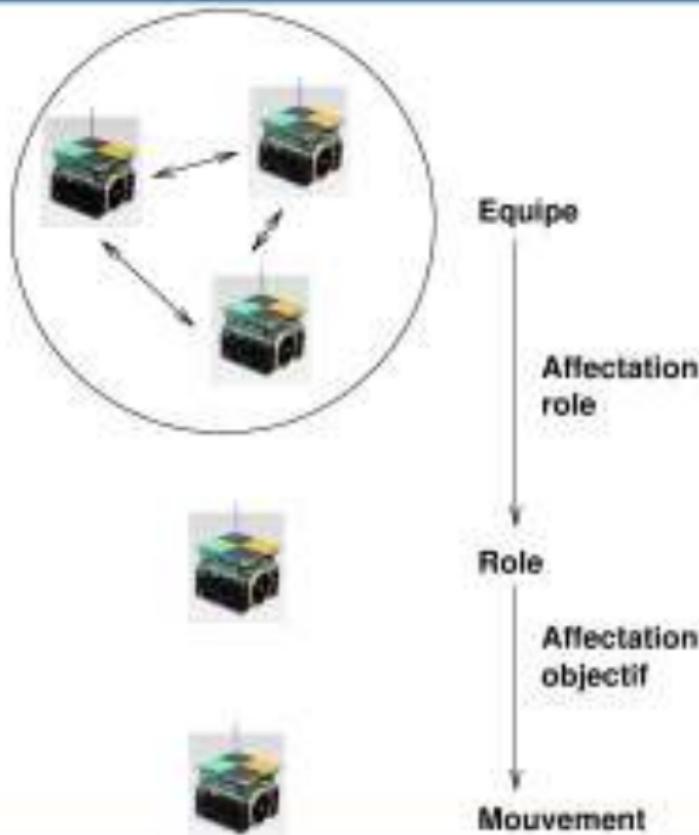
Scénario: Séquence typique d'interactions



Subsection 4

Exemple : robots footballeurs

Exemple : robots footballeurs



Quelques anticorps :

Paratope	Spécification de l'agent	Idéotope
AimFront	MoveFront	
ObstacleLeft	MoveFront	
ObstacleRight	TurnLeft	

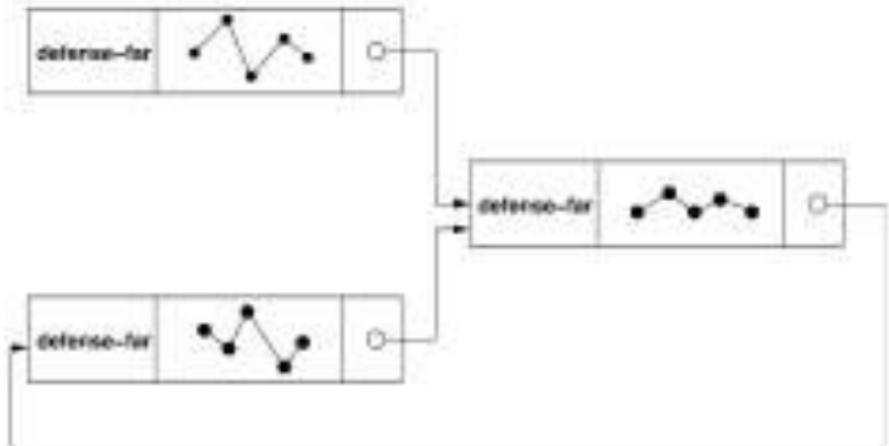


Figure : Examples of medium level antibodies

Precondition	Behavior	Affinities
defense-far	NorthEast	

Table : Medium level immune system antibody example

Precondition	Behavior	Affinities
TeamInControl and (BallZone = 1 or BallZone=2 or BallZone=3)	goalkeeper, near-defender, midfielder, left-attacker, right-attacker	

Table : High level immune system antibody example

Subsection 5

Sélection par clonage



Sélection par clonage



- Stimulation par antigène(s)
- Anticorps = réponses aux stimulations
- Codage des antigènes/anticorps

- A un antigène/anticorps on associe une séquence $\{b_1, \dots, b_n\}$ de valeurs numériques
- L'intervalle des valeurs dépend du problème
- L'affinité entre un antigène et un anticorps est calculé selon une fonction objectif

- Le clonage reproduit certains anticorps (solutions)
- Les techniques de sélection sont multiples
 - Les meilleurs anticorps
 - Aléatoire
 - Les meilleurs anticorps tout en gardant une population variée
 - ...

- Chaque clonage peut introduire une mutation i.e. un petit changement dans l'anticorps
- Il faut définir la fréquence (probabilité) de ces changements
- Il faut définir l'opération de mutation
 - Définir les changements possibles (une ou plusieurs positions du codage par ex)
 - Assurer la cohérence des solutions

- Un anticorps non stimulé voit le nombre de ses occurrences diminuer
- Si le nombre d'occurrence d'un anticorps est nul ou faible il est mort.

```
initialiser les anticorps B(0);  
calculer les affinités;  
tant que non(condition d'arrêt) faire  
    k:=k+1;  
    cloner(B(k-1));  
    muter(B(k-1));  
    sélectionerClones(B(k-1));  
    simulerMort(B(k-1));  
    calculer les affinités de B(k-1);  
    B(k):=B(k-1);  
fin
```

Algorithm 2: Algorithmme de sélection par clonage

Subsubsection 1

Conclusion

Conclusion

Différentes applications :

- Architecture pour la robotique mobile collective [Kim, 1996]
- Découverte adaptative de services dans des réseaux larges [Bakhouya, 2004]
- Détection de détérioration d'outils [Dasgupta, 1995]

Section 8

Conclusion générale

- Panel d'architectures pour la sélection d'action
- Degrés de dépendance de la représentation interne des agents
- Dédiées à un domaine d'application



Chapter 5: Rational agents

Stéphane GALLAND



- 1 Agents rationnels
- 2 Rationalité multi-agents
- 3 Confiance et Réputation dans les SMA



Rationalité dans les SMA



Subsection 1

Introduction



SMA

Les SMA sont composés d'agents autonomes chargés de résoudre des tâches pour le compte d'utilisateurs humains.

Exemples

- Transaction sur le web,
- Entreprise virtuelle,
- Assistant de voyage
- Simulation
- ...

- Ces domaines impliquent des buts propres à chaque agent.
- Ces buts peuvent être différents voire contradictoire et/ou en conflits.
- Chaque agent pris séparément ne peut accomplir ses buts.
- On veut obtenir une solution de ce "magma originel".



Dans les cas cités, il est nécessaire que les agents

- coopèrent pour accomplir leurs tâches/buts respectifs
- et négocient pour résoudre les conflits potentiels.

Mais chaque agent a des buts et il ne sait pas a priori comment les réaliser.



Quelques pistes pour donner aux agents les moyens de coopérer/négocier afin d'accomplir leurs buts

- Donner aux agents de la rationalité
- Donner aux agents des protocoles de coopération/négociation



Section 1

Agents rationnels

Subsection 1

Notions de base



Définition [Ferber, 1995]

Un agent est dit rationnel quand il ajuste ses moyens aux fins qu'il se propose d'atteindre.



- Il faut représenter les fins selon Ferber i.e. les buts de l'agent
- Il faut pouvoir prédire quelle est la "meilleure" action à mener
- L'architecture de sélection d'action doit être reconfigurable



Subsection 2

Abstraction d'un agent

Architecture abstraite pour les agents [Wooldridge, 2003a]

- On suppose que l'environnement peut-être décrit par un état parmi un ensemble fini d'états $E = \{e, e', \dots\}$
- Les agents ont un répertoire d'actions possibles, ces actions transforment l'état de l'environnement $Ac = \{a, a', \dots\}$
- La vie d'un agent dans un environnement est une séquence d'état de l'environnement et d'action :

$$r : e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} e_n$$

Soient

- \mathcal{R} l'ensemble de toutes les séquences finies possibles
- \mathcal{R}^{Ac} le sous ensemble de \mathcal{R} qui se termine par une action
- \mathcal{R}^E le sous-ensemble de \mathcal{R} qui se termine par un état de l'environnement

- Soit τ une fonction de transformation d'état

$$\tau : \mathcal{R}^{Ac} \rightarrow 2^E$$

- si $\tau(r) = \emptyset$ alors il n'y a pas de successeur pour r et le système est terminé
- Un environnement est un triplet $Env = \langle E, e_0, \tau \rangle$

- Un agent est une fonction qui associe à une séquence d'état-action une action :

$$Ag : \mathcal{R}^E \rightarrow Ac$$

Un agent prend la décision en fonction de son historique.

- Cas particulier des agents purement réactifs :

$$Ag : E \rightarrow Ac$$

L'agent prend sa décision sans référence à son historique.

- Soit I l'ensemble des états internes de l'agent
- La perception est vue comme une fonction

$\text{see} : E \rightarrow Per$

- La fonction de sélection d'action est définie par

$\text{action} : I \rightarrow Ac$

- et la fonction $next$ permet de connaître l'état interne suivant

$\text{next} : I \times Per \rightarrow I$



Subsection 3

L'Utilité

Économie et gestion

100



- On évalue les préférences de l'agent
- On évalue le lien entre une action et l'accomplissement des buts de l'agent

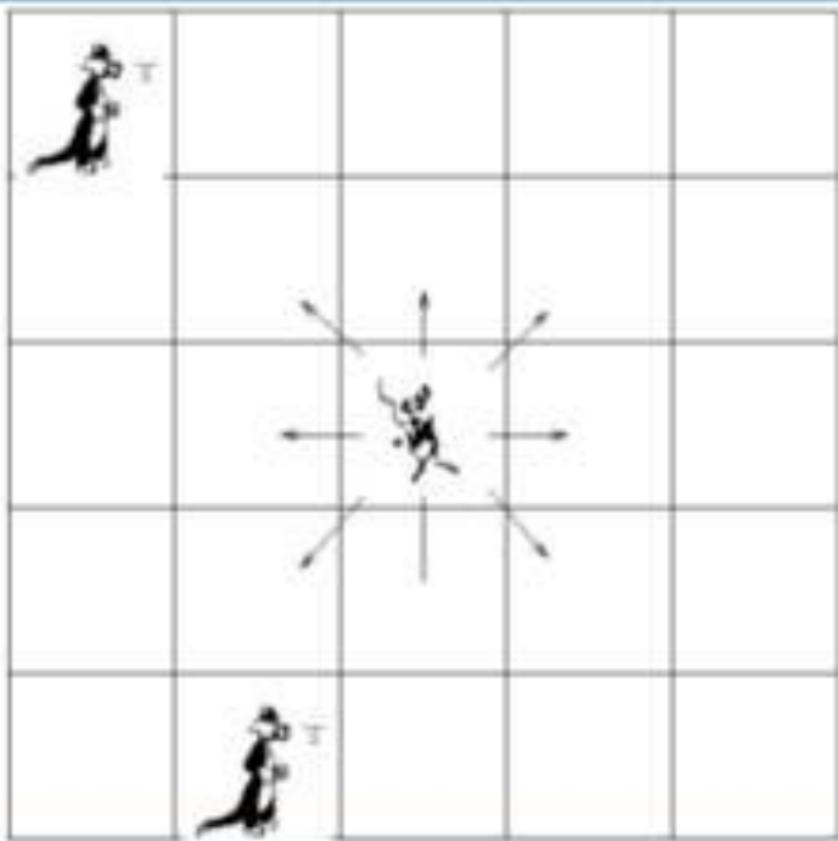
$$u : E \rightarrow \mathbb{R}$$

- On associe une utilité aux états individuels
- Un agent doit alors faire évoluer l'état de manière à maximiser l'utilité



Exemple : Proie et Prédateurs

22



- Chaque action se ramène à un déplacement
- Les états de l'environnement sont les positions pour les proies et les prédateurs
- Soit d la distance minimale entre la proie et les prédateurs

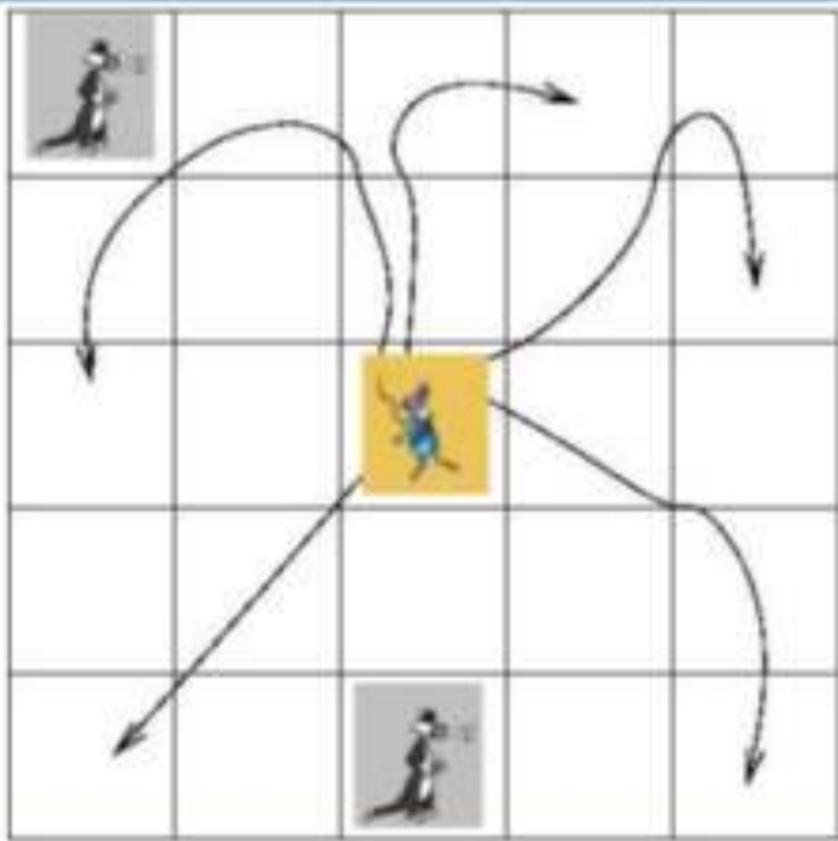
$$u(E) > u(E') \implies d(E) > d(E')$$



- Difficile de prévoir sur le long terme
- Difficile de trouver "la bonne utilité"

$$u : \mathcal{R}^E \rightarrow \mathbb{R}$$

- On associe une utilité aux séquences
- On a une notion d'utilité sur le long terme
- L'agent doit choisir une séquence d'actions qui maximise son utilité



- On évalue une séquence de déplacements proie/prédateurs
- Arbre de déplacements alternés proie puis prédateurs
- Chaque branche de l'arbre est valuée par l'utilité minimale des sous-branches



- On limite la fonction d'utilité à un prédicat

$$u : E \rightarrow \{0, 1\}$$

- Une action ou une séquence d'action amène une utilité vraie ou fausse
- Quid de l'exemple proie-prédateurs ?



- Très difficile de définir une utilité sur les séquences
- Problème d'infinitude des séquences



Subsection 4

La rationalité limitée

• Les agents rationnels sont des êtres raisonnables et logiques.

• Ils sont capables de faire des prévisions et de prendre des décisions.

• Ils sont capables de faire des prévisions et de prendre des décisions.

- Le temps de calcul de solutions précises ou optimales peut être prohibitif
- Le degré de précision nécessaire de la solution peut être dépendant du contexte
 - ~ on va se limiter au niveau de la rationalité et envisager des solutions imparfaites ou incomplètes.

- Limiter la taille des séquences,
- Simplifier le calcul de la fonction d'utilité.
- Solutions adaptées pour la production de solution en environnement contraint (ex Algo Anytime)



Algorithme Anytime [Zilberstein, 1996]

Les algorithmes Anytime sont des algorithmes où la qualité des résultats s'améliore graduellement quand le temps de calcul augmente.

Données: I un ensemble de données

Résultat: R le résultat

$R = f(I);$

Données: I un ensemble de données

Résultat: R une séquence de résultats, telle que

$$\forall i \leq j, Q(R_i) \leq Q(R_j)$$

i:=0;

$R_i :=$ solution-Initiale;

tant que non interrompu **faire**

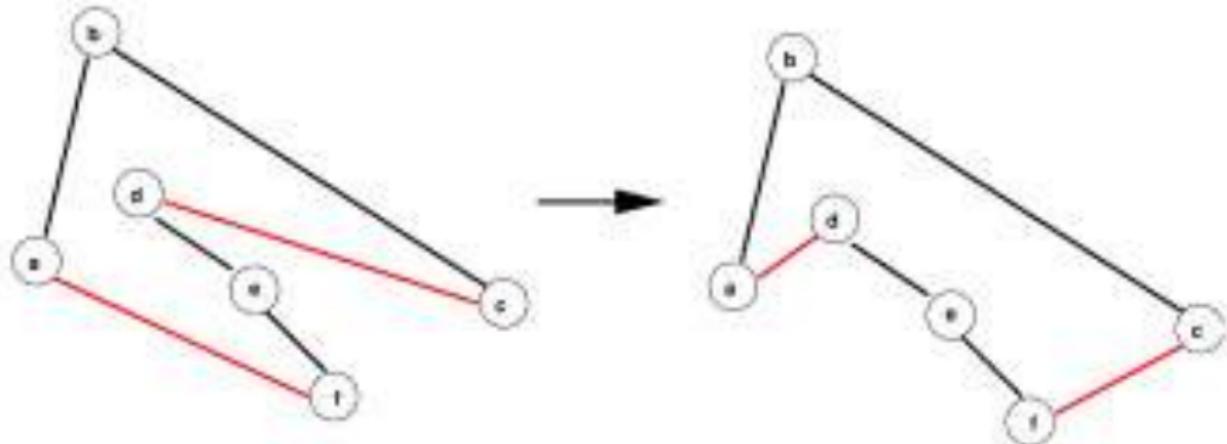
i:=i+1;

$R_i := f(R_{i-1})$;

fin

Exemple : Voyageur de commerce

38



$$\text{cout}(c, f) + \text{cout}(d, a) < \text{cout}(c, d) + \text{cout}(f, a)$$

Données: V un ensemble de données

Résultat: Tour

Tour:=TourInitial(V);

cout:=Cout(Tour);

pour $i:=1$ à iter **faire**

$e_1:=\text{areteAlea}(\text{Tour});$

$e_2:=\text{areteAlea}(\text{Tour});$

$\gamma:=\text{Cout}(\text{Tour})-\text{Cout}(\text{Echange}(\text{Tour}, e_1, e_2));$

si $\gamma \geq 0$ **alors**

 Tour := Echange(Tour, e_1, e_2);

 cout:=cout- γ ;

fin

fin

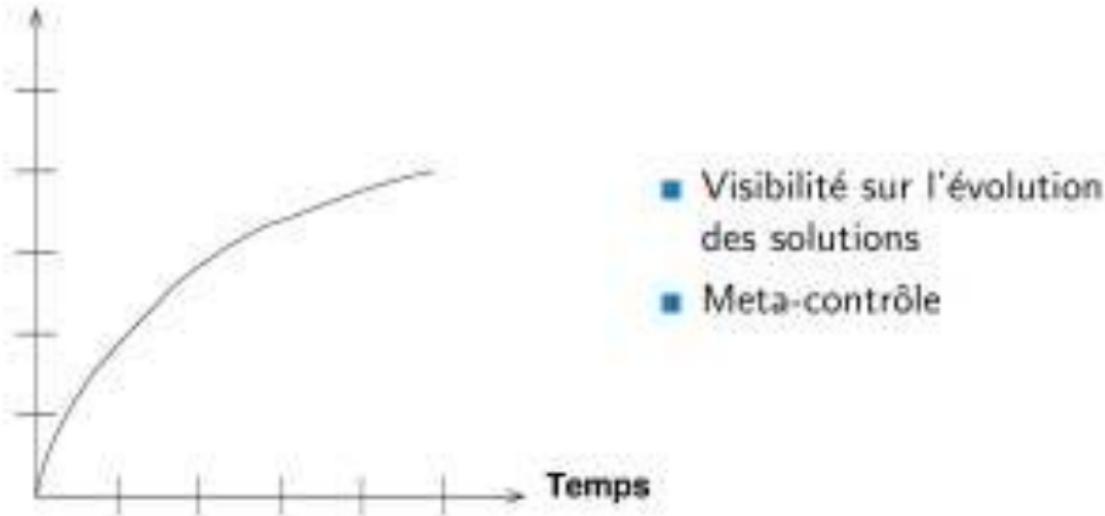
Signal(terminaison);

Stop;

- Recherche itérative
- Algo A* (Phase de calcul du coût des routes)
- Code de Gray: type de codage binaire permettant de ne modifier qu'un seul bit à la fois quand un nombre est augmenté d'une unité.
- ...

- Notion de mesurabilité
- Profil de Performance $Q(t)$
- Profil de performance conditionnel $Pr(q_{out}|q_{in}, t)$

Qualité





- Etude problème par problème
- Monitoring
- Statistiques

- Problème de non réactivité des agents délibératifs
- Difficile de prévoir le futur des agents délibératifs

- Filtrage intelligent et graduels des percepts
- Introduction d'agents "à la demande"
- Contrôle de meta-niveau
- ...



- Filtrage par distance dans environnement situés
- Filtrage par niveaux de détails pour environnements virtuels

- On adapte le SMA aux besoins au temps t (redéploiement des agents)
- On adapte les ressources aux besoins au temps t
- On compose au mieux les algorithmes anytime (par un algo anytime ?)

"On" c'est le meta-niveau



Section 2

Rationalité multi-agents



Subsection 1

Introduction

- Chaque agent est autonome : il va choisir la meilleure stratégie le concernant
- Le système doit remplir les buts qu'on lui a fixé
- Le système est ouvert

- Conception des protocoles d'interaction de manière **non coopérative**
- Stratégie globale → stratégie locale
- Accomplissement des buts globaux au mieux



- Un agent va négocier/coopérer quand ça lui profite
- Un agent va négocier/coopérer quand l'utilité est supérieure au cas non négociation/coopération

Subsection 2

Critères de rationalité multi-agents

[Weiss, 1999]

- Bien-être de la société
- Pareto optimal
- Convergence/succès
- Rationalité individuelle
- Equilibre de Nash
- Stabilité
- Simplicité/Complexité

Subsubsection 1

Deux exemples



Prisoner's Dilemma Game (PDG)

John et Bob se sont fait arrêter après le vol d'une bijouterie. Ils sont placés dans deux pièces différentes et on dit à chacun :

- "Si tu dénonces ton copain, tu seras libre et l'autre sera condamné à 10 ans.
- Si vous vous dénoncez mutuellement vous écoperez de 5 années chacun.
- Enfin si aucun de vous deux ne parle vous aurez 1 an au bénéfice du doute."

John / Bob	coopère	coopère pas
coopère	5,5	0,10
coopère pas	10,0	1,1

Pierre Ciseaux Papier (PCP)

On a deux joueurs. Chacun choisit simultanément une valeur dans l'ensemble { Pierre , Ciseaux , Papier }.

Les résultats sont les suivants.

J_1 / J_2	Pierre	Ciseaux	Papier
Pierre	0,0	1,-1	-1,1
Ciseaux	-1,1	0,0	1,-1
Papier	1,-1	-1,1	0,0

Subsubsection 2

Bien-être - Social Welfare

- Somme des utilités des agents pour une solution donnée
- "Utilité globale" au niveau système
- Le but est de maximiser le bien être donc la somme



- Arbitraire de la comparaison (tous même poids ou poids différents ?)
- Nécessité de comparaison des différentes utilités des agents
- Sémantique difficile à définir

- Pour le PDG, l'optimum est quand les deux prisonniers ne coopèrent pas (en terme du plus petit nombre d'années de prison)
- Pour PCP, il n'y a pas d'optimum c'est un jeux à somme nulle (compétition)

Subsubsection 3

Pareto optimalité

- Ordre sur l'ensemble des solutions des agents prise une à une
- Soit $X = \{x_0, x_1, \dots, x_n\}$ un ensemble de solutions X est dite Pareto optimale s'il n'existe pas de $X' = \{x'_0, x'_1, \dots, x'_n\}$ telle qu'au moins une solution x'_i soit meilleure qu'une x_i et qu'aucune solution ne soit pire qu'en x_i .



- Il n'est pas nécessaire d'homogénéiser les utilités des différents agents
- solutions bien être maximale \subseteq solutions pareto optimales
- Non comparabilité de certaines solutions

- Pour le PDG, l'optimum est quand les deux prisonniers ne coopèrent pas (en terme du plus petit nombre d'années de prison)
- Pour PCP, il n'y a pas d'optimum de pareto les solutions sont non comparables

Subsubsection 4

Equilibre de Nash

- Soit un ensemble d'agents A
- Soit $S_A^* = (S_0^*, S_1^*, \dots, S_{|A|}^*)$ le profil des stratégies des agents de A
- Dans S_A^* , S_i^* est la stratégie de l'agent i
- S_A^* est en équilibre de Nash si pour tout agent i , S_i^* est la meilleure stratégie pour l'agent i en supposant que les autres agents choisissent les stratégies $(S_0^*, S_1^*, \dots, S_{i-1}^*, S_{i+1}^*, \dots, S_{|A|}^*)$

- Chaque agent connaît ou essaye de deviner la stratégie des autres
- La meilleure stratégie d'un agent est la meilleure réponse aux stratégies des autres agents
- les agents essayent de maintenir cet équilibre

- Pour certaines situations, il n'existe pas d'équilibre de Nash
- Pour certaines situations, il existe plusieurs équilibres de Nash

- Un système en équilibre de Nash à un moment t peut ne pas y être à $t + \Delta t$
- L'équilibre de Nash est trop fort : des sous-groupes d'agents peuvent dévier de manière coordonnée,

- Pour le PDG, l'équilibre de Nash est atteint lorsque les deux prisonniers ne coopèrent pas
- Pour PCP, l'équilibre de Nash est atteint pour les situations symétriques (Pierre-Pierre, Papier-Papier, Ciseaux-Ciseaux) comparables

Subsubsection 5

Complexité



- Complexité en terme d'algorithmes locaux
- Complexité en terme d'envoi de messages
- Complexité en terme de structures employés
- ...

Subsubsection 6

Stabilité

- Convergence vers une solution
- Robustesse
- deadlock
- ...



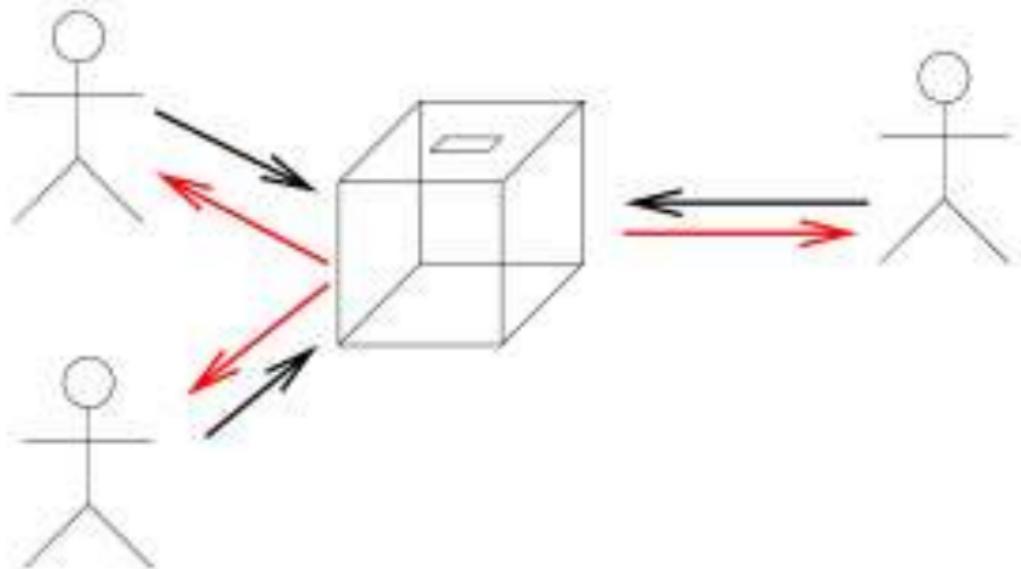
Subsection 3

Protocoles de coopération

Subsubsection 1

Votes

Dans un vote tous les agents donnent une entrée au mécanisme et la sortie choisit une solution basée sur les entrées.



- Définition du mécanisme
- obligation pour les agents d'accepter le choix final

Soit A l'ensemble de agents, $(\preccurlyeq_0^i, \preccurlyeq_1^i, \dots, \preccurlyeq_{|A|}^i)$ les préférences de l'agent i sur les solutions, O l'ensemble des solutions pour le système.

- Un ordre de préférence \preccurlyeq^* devrait exister pour toutes les entrées possibles $(\preccurlyeq_0^i, \preccurlyeq_1^i, \dots, \preccurlyeq_{|A|}^i)$
- \preccurlyeq^* doit être défini pour tout $o, o' \in O$
- \preccurlyeq^* doit être assymétrique et transitive sur O
- La solution devrait être Pareto optimale :
 $\forall i \in A, o \preccurlyeq_i o' \implies o \preccurlyeq^* o'$
- Pas d'agent dictateur : $o \preccurlyeq_i o'$ n'implique pas forcément $o \preccurlyeq^* o'$ pour les autres agents

Ces contraintes ne peuvent pas toujours être satisfaites dans le cas général.

- Restriction du domaine de \preceq^* en imposant des règles sur les préférences des agents et/ou sur les solutions possibles
- Choix du maximum par pré-ordre
- Dictature aléatoire

Subsubsection 2

Enchères - Auctions



Principe des enchères

"Un commissaire-priseur propose un bien/ressource/service/tâche et des soumissionneurs font des propositions. Le commissaire-priseur veut obtenir une contractualisation à un coût minimal, les soumissionneurs veulent obtenir un contrat à un cout maximal."



English Auction

"Chaque soumissioneur est libre d'augmenter sa proposition. Lorsqu'aucun soumissioneur ne veut plus augmenter, l'enquête se termine et la meilleure offre l'emporte."

La stratégie d'un soumissioneur est une fonction de sa valeur d'utilité, des soumissions passées des autres soumissioneurs et de son estimation des autres soumissions.



Dutch Auction

"Le commissaire-priseur descend continuellement le prix jusqu'à ce qu'un des soumissionneurs prennent l'offre au prix courant."

Les soumissionneurs ne connaissent pas les soumissions des autres agents. Chacun attend que l'offre atteigne un niveau immédiatement inférieur à sa valeur utilité.



Vickrey Auction

"Chaque soumissionneur fait une proposition sans connaitre celles des autres. Le gagnant est la meilleure offre au prix de la seconde meilleure."

Même chose que l'enchère anglaise mais sans connaitre les offres passées.

Le lot revient au plus offrant, mais celui-ci doit payer le prix donné par le deuxième meilleur enchérisseur.

Subsubsection 3

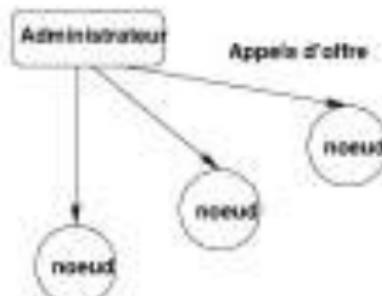
Réseau Contractuel - Contract Net Protocol



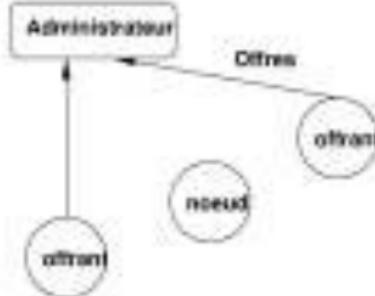
- Noeuds d'un réseau
- Modélisation des tâches
- Modélisation des offres pour ces tâches

QUESTION

1



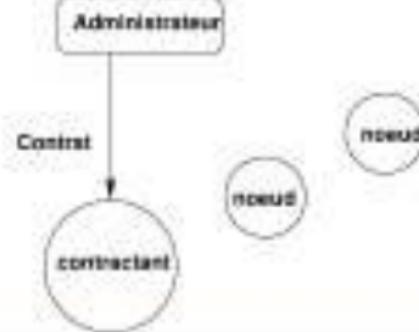
2

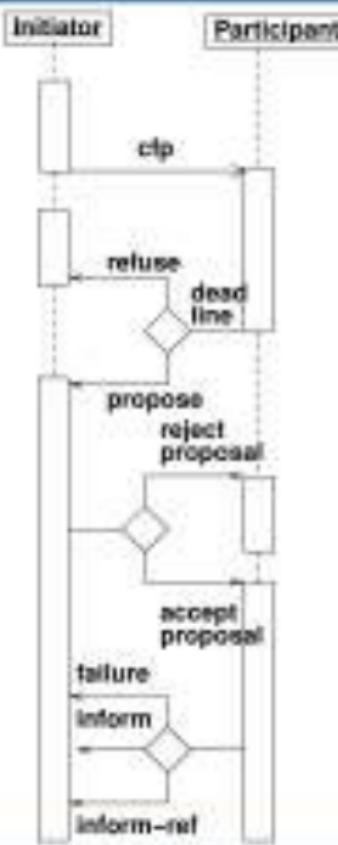


3



4







Section 3

Confiance et Réputation dans les SMA



Subsection 1

Introduction



- Systèmes ouverts ⇒ non prévisibilité des entités entrantes du système
- Possibilité d'agents malveillants
- Possibilité d'agents malfonctionnant

- L'incertitude entraîne le besoin de confiance
- La délégation d'un utilisateur vers un agent engendrent :
 - Un risque
 - Varie selon l'autonomie
- La délégation et la coopération entre agents engendrent :
 - Un risque
 - Le choix de partenaires

- Un agent A cherche à acheter un CD
- Un agent B le propose à 9 € et un agent C à 10 €
- A choisit l'offre de B
- A paye les 9 € et ne reçoit jamais rien

- Un agent A cherche à acheter un CD
- Un agent B le propose à 9 € et un agent C à 10 €
- A choisit l'offre de C
- A paye les 10 € et reçoit un CD rayé



- Comment réduire le risque de choix d'un partenaires malveillant ?
- Comment garantir la qualité des résultats ?



Subsection 2

Les techniques

- Architecture 3-tiers où une entité joue le rôle d'intermédiaire (Garant) entre un acheteur et un vendeur
- Solution centralisée
- Le problème de la confiance envers le Garant se pose
- Difficile d'évaluer la qualité des solutions

- La réputation est une croyance d'un agent sur la fiabilité d'un autre agent
 - Elle peut être inexacte
 - Elle évolue en fonction des perceptions et interactions
 - Peut être assimilée à une probabilité
- Peut être construite et partagée par des groupes d'agents



- Valeur numérique
- Réputation devient une fonction des agents vers un intervalle numérique
- Autres (échelle de valeurs discrète, ...)

- Site de vente en ligne
- Des utilisateurs avec les rôles de vendeur et acheteur s'enregistrent
- Principe de l'enchère à l'anglaise
- A la fin de la transaction une évaluation du vendeur est donnée par l'acheteur
- Ces évaluations sont combinées dans l'eBay Card Id du vendeur



- Carte d'identité
 - Date d'arrivée dans le système
 - Somme des évaluations
 - Résumé des évaluations récentes
- Historique complet des évaluations

- Initialisation de la réputation
- Distinction entre agent inconnu et agent de réputation "moyenne"
- Pérénitée dans les systèmes ouverts
- Falsification des Id



Subsection 3

Réputation et SMA

- Problème de centralisation
- Perception/Connaissance limitée et imprécises
- Hétérogénéité des agents et buts différents

- Subjective et non plus globale
- Construite par coopération
- Différents types de réputation (objectif, moyen d'acquisition, ...)

- Réputation provenant d'autres agents dits témoins
- Réputation provenant de l'environnement proche de l'agent

- Combinaison de plusieurs réputations
- Pondération des relations

- Introduit notion rationalité par le biais d'une mesure de confiance
- Permet de définir des règles "sociales"
- Favorise la conception de systèmes ouverts



Subsection 4

Conclusions Générales

- Le problème de conception d'agents rationnels n'est pas trivial
- Le problème de conception d'agents rationnels génériques est ouvert
- Aucune solution optimale pour un problème NP-difficile

- Pour chaque problème différentes pistes utilité permet hétérogénéité protocoles de coordination permettent prévisibilité
- Limitation de la calculabilité



Chapter 6: Planification

Stéphane GALLAND



- 1 Notions de base
- 2 Planification centralisée
- 3 Architecture de planification avec contraintes temps réel : les STN
- 4 Planification distribuée

Subsection 1

Introduction



- Réactivité :

l'agent doit savoir comment réagir aux changements de son environnement

- Pro-activité :

l'agent doit savoir comment aboutir à la réalisation de son(ses) but(s)

- Comment résoudre des buts à plusieurs \implies distribution de buts/tâches
- Comment coordonner les agents, éviter les conflits, les faire coopérer !



Section 1

Notions de base



1 Notions de base

- Définitions
- Le cas des systèmes multi-agents
- Différentes stratégies

2 Planification centrale

Architecture de planification avec continuels temps réel : les 5 TIR

3 Planification distribuée

Subsection 1

Définitions

Définition: Planification [Chaib-Draa, 2003]

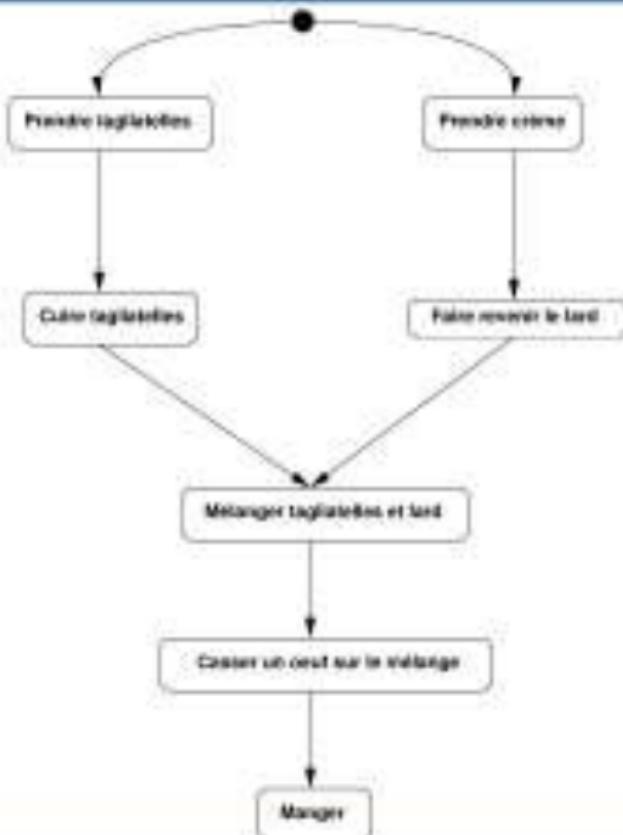
Planifier c'est concevoir une série d'actions liées les unes aux autres en vue de réaliser un but.

~ Notions d'action, de **but** et de **réalisation de ce but**.



Exemple : Un plan pour faire des tagliatelles carbonara

10





- Donner une **représentation du but à réaliser**
"Manger des tagliatelles carbonara"
- Donner une **représentation des actions possibles**
"prendre tagliatelles", "prendre crème", ...
- Donner une **représentation de l'environnement**
fonction de perception
- Donner un **mécanisme de génération de plan**
Le mécanisme qui génère la recette

tant que true faire

- observer l'environnement;
- mettre à jour l'état interne;
- décider quels sont les buts à accomplir;
- calculer un plan;
- exécuter le plan;

fin

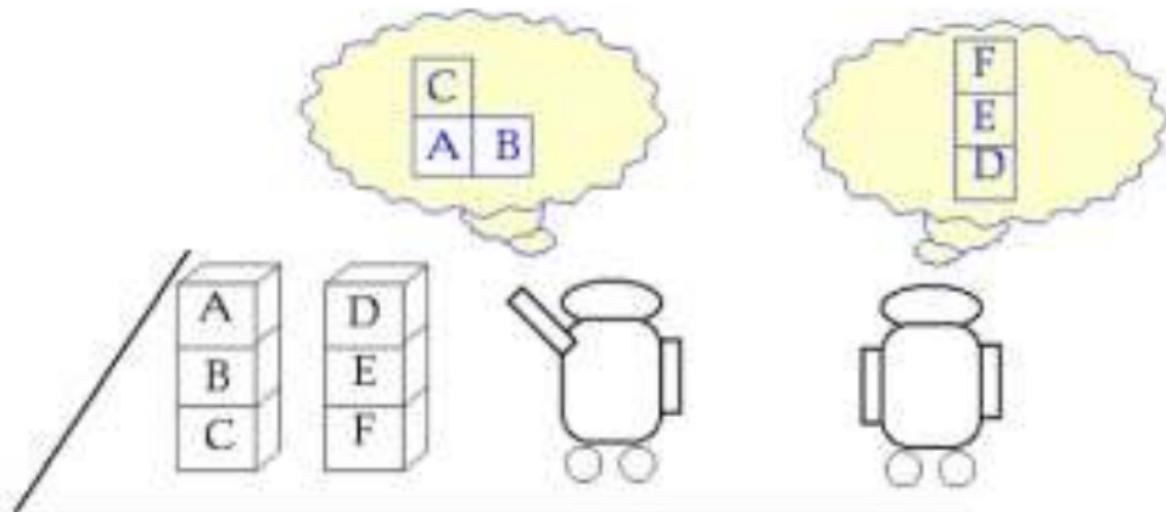
Algorithm 3: Algorithme général d'un agent planifieur

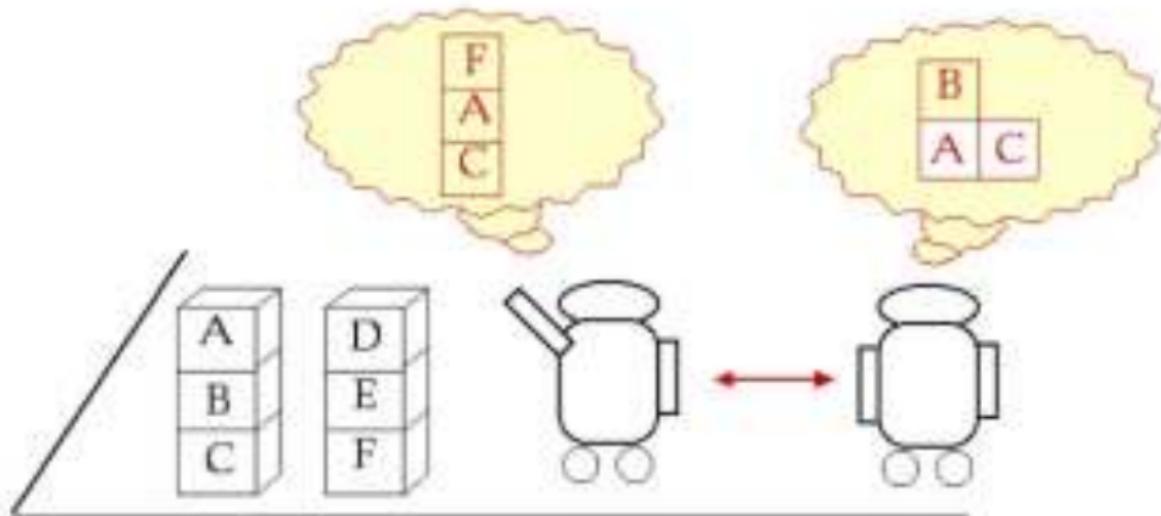
Subsection 2

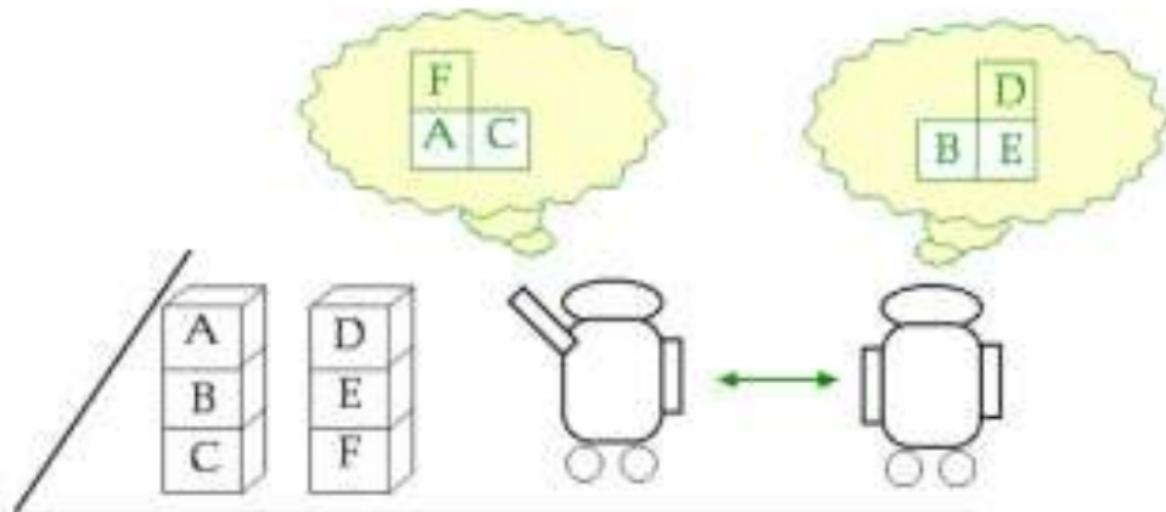
Le cas des systèmes multi-agents



- La décision des buts à accomplir peut prendre en compte les buts des autres agents
- Le calcul du plan peut prendre en compte les plans des autres agents
- L'environnement est dynamique et peut remettre en cause la planification

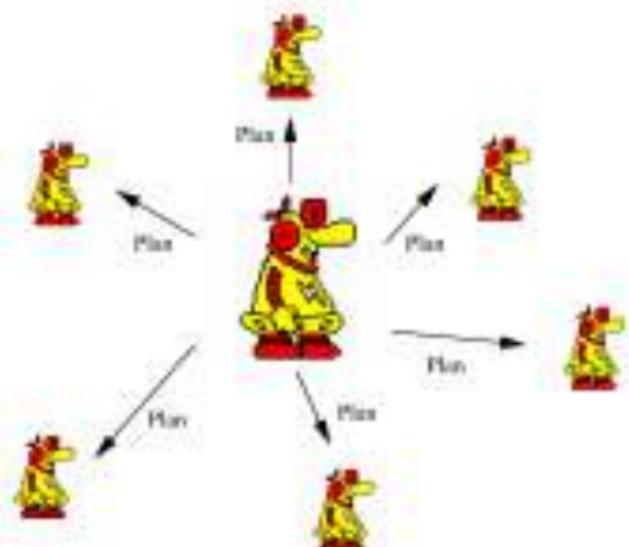






Subsection 3

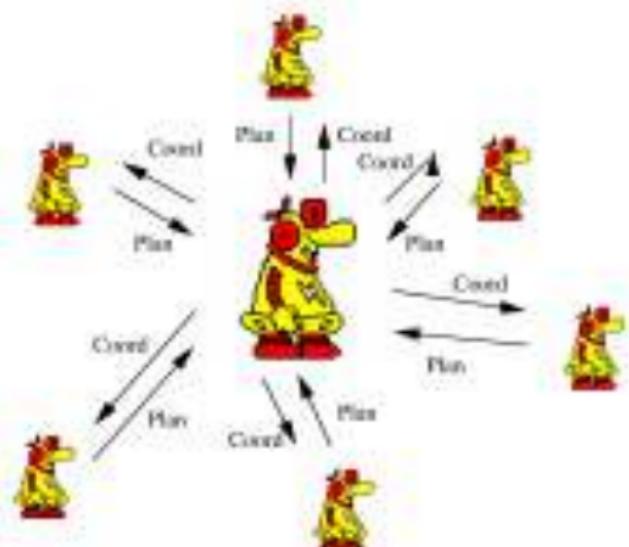
Differentes stratégies



- 1 Plan général partiel, exprimé sous forme d'un graphe acyclique.
- 2 Branches pouvant être en parallèle, points de synchronisation quand elles se joignent.
- 3 Allocation des tâches aux exécuteurs.

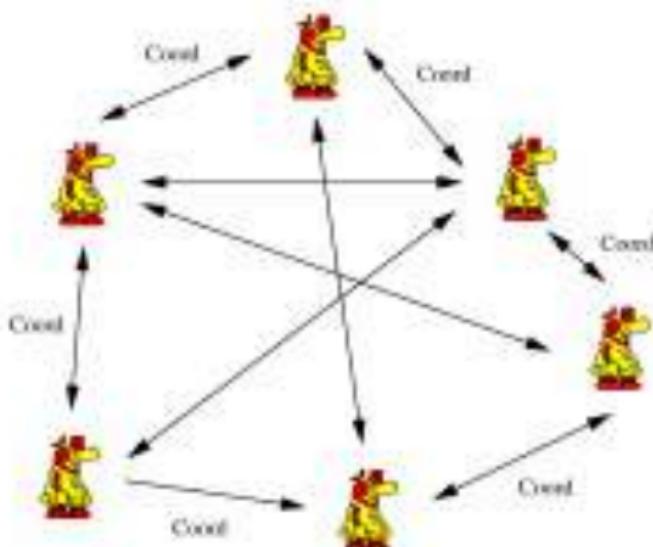
Coordination centralisée pour plans partiels

21



Seule la coordination est centralisée (fusion de plans partiels), trois cas possibles

- Indépendance des actions,
- Relations positives entre actions (égalité, faveur, ...),
- Relations négatives (conflits de ressources).



- Répartition complète sur les agents (Planification agent),
- Mécanisme de génération/révision de plan doit être multi-agentifié

- Complexité de la méthode
- Hypothèse d'homogénéité des agents
- Limitation du nombre d'agents
- Hypothèse de régularité du monde
- Problèmes temporels : durée des actions, simultanéité

Solutions envisageables : Hiérarchisation des plans, Engagement des agents.



- Solution bien adaptée pour certains problèmes
- Forte prédictivité



Section 2

Planification centralisée

Principes et méthodes de planification

1 Marchés de biens

2 Planification centralisée

- STRIPS
- Planification à ordre total
- Planification à ordre partiel

3 Architecture de planification avec coûts unifs temps réel : les SPP

4 Planification distribuée

Subsection 1

STRIPS

STRIPS

STRIPS



STRIPS: STanford Research Institute Problem Solver

- Les états du monde sont représentés par une conjonction de formules atomiques (logique des prédictats du premier ordre).
- Les opérations sont exprimées en termes de formules logiques.



Exemple : Clotaire [Ferber, 1995]

Clotaire est un robot, capable de se déplacer (nord, sud, est, ouest) de pièce en pièce pour aller chercher des objets.

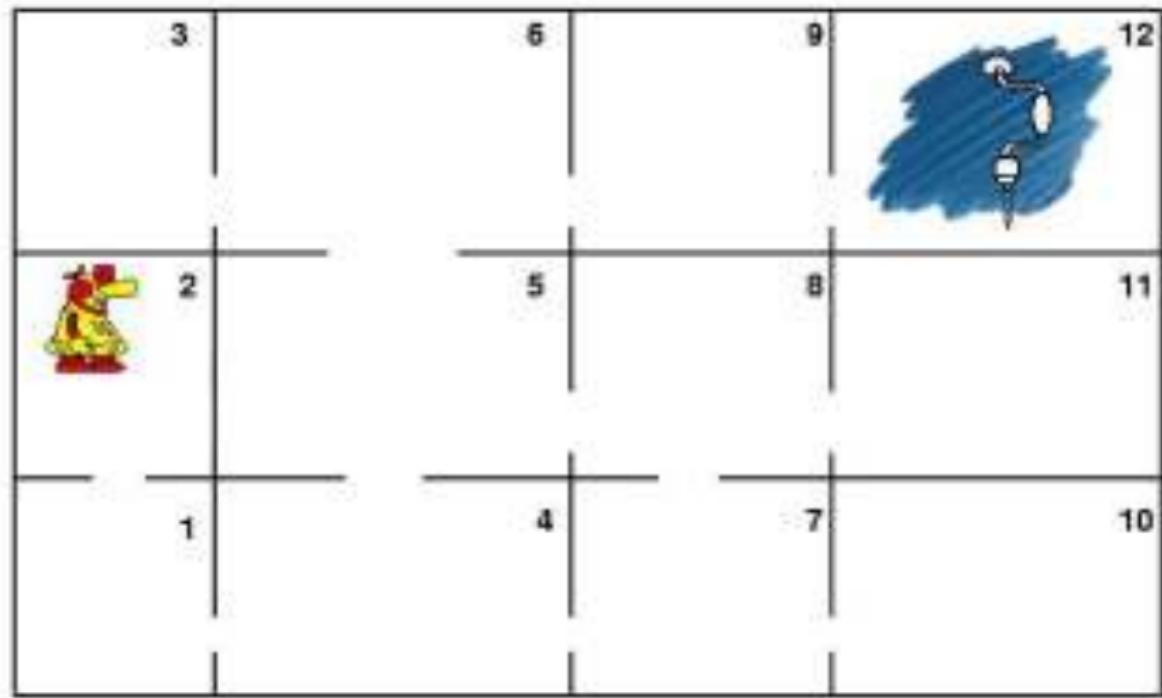
Un état du monde s_1 est par exemple

$$s_1 = \{ positionRobot(Clotaire, 2), \\ positionOutil(Perceuse, 12), \\ nord(1, 2), \\ sud(2, 1), \\ est(1, 4), \dots \}$$



Exemple : Clotaire

11





Une opération est définie par un triplet :

$$< \text{Pre}, \text{Add}, \text{Del} >$$

où Pre, Add et Del sont des formules atomiques.

Pour chaque opération on précise :

- 1 **Pre** : les pré-conditions portant sur l'état du monde,
- 2 **Add** : les formules à ajouter dans l'état suivant,
- 3 **Del** : les formules à supprimer dans l'état suivant.

Add et Del modélise les effets/post-conditions de l'opérateur

Opérateur allerVersSud(x)

Pre : positionRobot(x,l1),sud(l1,l2)

Add : lieu(x,l2)

Del : lieu(x,l1)

fin

Subsection 2

Planification à ordre total



Objectif

Trouver une séquence d'opérateurs (plan) depuis l'état initial ou courant jusqu'à l'état satisfaisant l'objectif courant.

Deux grands types de planification

- **Forward state space search:** goal progression, algo progressif
- **Backward state space search:** goal regression, algo régressif



Progression planning

- On part de l'état initial
- On détermine les opérateurs remplissant les préconditions
- On calcule l'état suivant en appliquant les clauses add/del



Regression planning

- On part du but jusqu'à obtenir l'état initial
- Description partielle des états car opérateurs inconnus

Fonction progressionPlan

Données: S un modèle du monde

Résultat: Π un plan

pour un opérateur possible a_i **faire**

$u := \text{unify}(s, \text{precond}(a_i));$

 // Croise l'état de l'environnement et les precond de a_i

si unifySucceeds **alors**

 apply($u, \text{dellist}(a_i), \text{addlist}(a_i))$;

 // remplace les variables dans del/add

$t := s;$

pour chaque $d_i \in \text{dellist}(a_i)$ **faire** delete(d_i, t);

pour chaque $d_i \in \text{addlist}(a_i)$ **faire** add(d_i, t);

si $\neg \text{goalTest}(t)$ **alors** $\Pi := \Pi \cup a_i \cup \text{progressionPlan}(t);$

fin



■ AllerVersSud(Clotaire)

■ AllerVersOuest(Clotaire)

■ AllerVersNord(Clotaire)

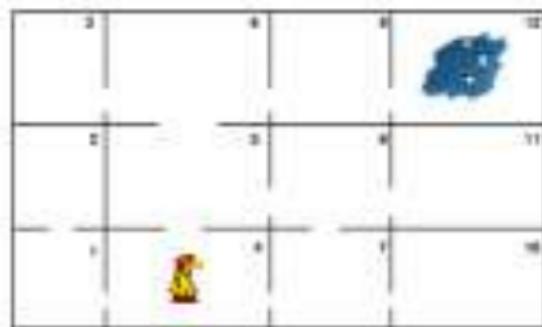
■ AllerVersEst(Clotaire)

■ Reste une case (Clotaire)

■ Reste une case (Clotaire)



- 1 AllerVersSud(Clotaire)
- 2 AllerVersEst(Clotaire)
- 3 AllerVersNord(Clotaire)
- 4 AllerVersOuest(Clotaire)
- 5 Reste au sud(Clotaire)
- 6 Reste au nord(Clotaire)

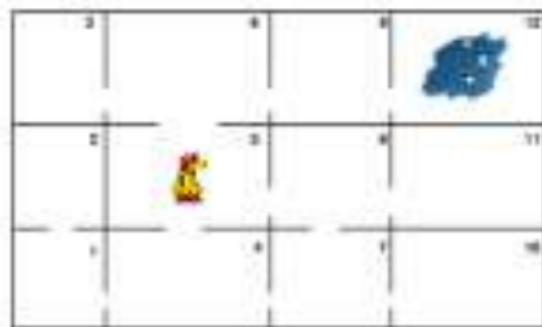


- AllerVersSud(Clotaire)
- AllerVersEst(Clotaire)
- AllerVersNord(Clotaire)

AllerVersOuest(Clotaire)

AllerVersEst(Couvent)

AllerVersOuest(Couvent)



- AllerVersSud(Clotaire)
- AllerVersEst(Clotaire)
- AllerVersNord(Clotaire)
- AllerVersNord(Clotaire)
- AllerVersEst(Claude)
- AllerVersNord(Claude)



- 1 AllerVersSud(Clotaire)
- 2 AllerVersEst(Clotaire)
- 3 AllerVersNord(Clotaire)
- 4 AllerVersNord(Clotaire)
- 5 AllerVersEst(Clotaire)

Clotaire va vers le sud puis vers l'est



- 1 AllerVersSud(Clotaire)
- 2 AllerVersEst(Clotaire)
- 3 AllerVersNord(Clotaire)
- 4 AllerVersNord(Clotaire)
- 5 AllerVersEst(Clotaire)
- 6 AllerVersEst(Clotaire)



- 1 AllerVersSud(Clotaire)
- 2 AllerVersEst(Clotaire)
- 3 AllerVersNord(Clotaire)
- 4 AllerVersNord(Clotaire)
- 5 AllerVersEst(Clotaire)
- 6 AllerVersEst(Clotaire)

Fonction REG-PLAN

Données: t: un état, plan: le plan en construction

choix de o parmi les opérateurs:

si état courant contient L unifiable avec $o_i \in \text{Add}(o)$ **alors**

 u := unify(L, o_i);

 p' := substitution de u dans Pre(o) ;

 t' := substitution de u dans t ;

pour chaque m de t' **faire**

si m \in Add(o) **alors** m est vraie;

sinon si m \in Del(o) **alors** m est faux;

 ;

fin

 s := UNION(p', t');

fin

si s \neq étatInitial **alors** plan:= REG-PLAN(s, plan);

- Représentation adaptée pour la planification
- Modélisation naturelle

- Les conséquences des opérateurs doivent être exhaustives
- Le codage du monde et des opérations sont exhaustifs
- Linéarité/Séquentialité: Pas de parallélisme entre opérateurs/actions



Subsection 3

Planification à ordre partiel

- Planification à ordre total:

- Recherche dans l'espace d'états par chainage avant ou arrière.
- Exploite seulement des séquences linéaires d'actions connectant l'état initial à l'objectif
- Pb majeur: permet pas de raisonner séparément sur des sous-pbs. Pas de décomposition/réutilisation du pb, seulement des solutions globale.

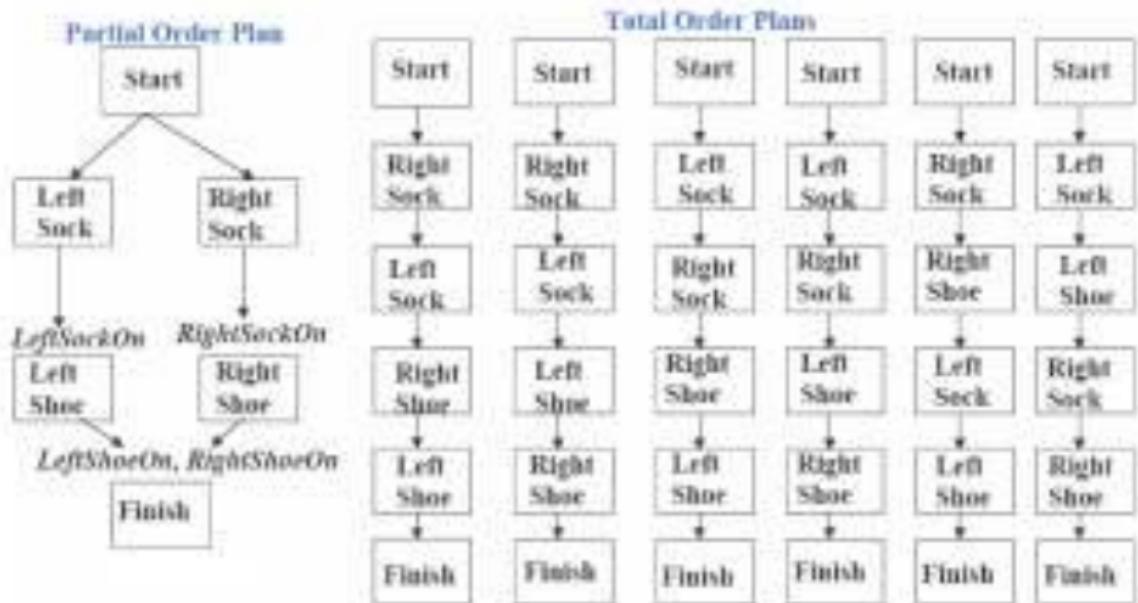
- Planification à ordre partiel:

- Recherche dans l'espace de plan.
- Résout les sous-problèmes indépendamment avec des sous-plans
- Combine les sous-plans pour déterminer un plan global

Exemple:

- Plan partiel: Se chausser = (MettreChaussettes, MettreChaussures)
- Un planificateur peut réaliser l'objectif en exécutant la séquence de deux actions: ChaussetteDroite, ChaussureDroite puis ChaussetteGauche, ChaussureGauche
- On combine ensuite les deux séquences pour produire le plan final
- Plan final: Se chausser =
(ChaussetteDroite, ChaussetteGauche, ChaussureGauche, ChaussureDroite)
- Le planificateur manipulera les deux sous-séquences **indépendamment**, sans s'engager sur le fait qu'une action dans une séquence sera avant ou après une action dans l'autre

- Tout algorithme de planification, qui peut placer deux actions dans un plan sans spécifier quelle action vient en premier, s'appelle **planificateur d'ordre partiel**
- Une solution de plans partiellement-ordonnés peut correspondre à plusieurs plans totalement-ordonnés. Chacun d'eux s'appelle **linéarisation de plans partiellement ordonnés**



- La planification partiellement-ordonnée peut être implémentée comme une recherche dans l'espace de plans partiellement ordonnés.
- On débute avec un plan vide et après on cherche comment raffiner le plan jusqu'à ce qu'on arrive à un plan complet qui résout le problème
- Les actions dans cette recherche ne sont pas des actions dans le monde mais des actions sur des plans: ajouter un pas dans un plan, imposer un ordre qui met une action avant une autre,...
- Différentes méthodes heuristiques de recherche peuvent être appliquées quand le problème de recherche est formulé
- Les états du problème de recherche seront des plans non finis.

Un plan est un quadruplet

$$\langle S, T, V, L \rangle$$

- **S:** ensemble des pas (actions) s_i
- **T:** contraintes temporelles sur les pas s , $s_i \prec s_j$
une contradiction existe si $s_i \prec s_j$ et $s_j \prec s_i$
- **V:** ensemble des contraintes sur les valeurs des variables de la forme $\text{var} = \text{x}$
une contradiction existe si $\text{var} = A$ et $\text{var} = B$ avec $A \neq B$
- **L:** ensemble des liens causaux de la forme

$$s_i \longrightarrow_c s_j$$

s_i réalise la précondition c de s_j

- A t_0 , Le plan vide contient juste les actions Début et Fin

Début n'a pas de préconditions et a comme effets tous les littéraux de l'état initial du problème de planification

Fin n'a pas d'effets et a comme préconditions les littéraux de l'objectif du problème de planification

Contrainte d'ordre:

$A \prec B \iff "A \text{ avant } B" \iff \text{Action } A \text{ doit être exécutée avant } B \text{ mais pas forcément juste avant.}$

Liens causaux:

$A \rightarrow_c B \iff A \text{ accomplit } c \text{ pour } B$

ChaussetteDroite \rightarrow *ChaussetteDroiteMise ChaussureDroite*

Planification complète et consistante

une précondition c de s_j est réalisée par s_i si :

- 1 $s_i \prec s_j$
- 2 $c \in \text{EFFETS}(s_i)$
- 3 $\nexists s_k$ tel que $\neg c$ est un élément de $\text{EFFETS}(s_k)$ et
 $s_i \prec s_k \prec s_j$ est une linéarisation du plan.

Autrement dit, pour chaque linéarisation du plan:

$$\forall k \text{ si } s_i \prec s_k \prec s_j \text{ alors } \neg c \notin \text{EFFETS}(s_k)$$

- La planification consiste à ajouter des pas qui réalisent les préconditions du but
- Le plan est complet quand chaque précondition de chaque pas est réalisée par un autre pas
- Le plan est consistant quand il n'y a pas de contradictions dans les contraintes d'ordre et de liens entre variables
- Un plan complet et consistant est une solution

- On cherche dans l'espace des plans plutôt que dans l'espace des situations
- Le plan initial contient l'état initial et l'état final
- A chaque itération on ajoute un pas
- Si on aboutit à un plan inconsistante on fait un retour en arrière et on essaye une autre branche
- On ajoute des pas qui réalisent une précondition qui n'a pas déjà été réalisée

Données: initial l'état initial, goal le but et operators les opérateurs

Résultat: plan

plan.S := {start, finish};

start contient les effets qui incluent les prédictats d'initial;

finish contient toutes les préconditions de l'état goal;

tant que Les préconditions de finish ne sont pas remplies faire

 | Ajout d'un opérateur à plan;

 | Traitement des incohérences;

fin

s_{need} := choix dans plan.S d'un sous-but qui a une précondition inachevée c;

s_{add} := choix d'un pas dans les opérateurs ou dans plan.S qui a c comme effet;

si $s_{add} = \emptyset$ **alors**

 | backtrack;

fin

plan.L := plan.L \cup $s_{add} \rightarrow_c s_{need}$;

plan.T := plan.T \cup $s_{add} \leftarrow s_{need}$;

si $s_{add} \notin plan.S$ **alors**

 | plan.S := UNION(plan.S, s_{add});

 | plan.T := plan.T \cup start $\leftarrow s_{add} \leftarrow finish$;

fin

pour chaque s_t qui menace un lien $s_i \rightarrow_c s_j$ **faire**

choisir;

$\text{plan.T} := \text{plan.T} \cup s_t \prec s_i;$

$\text{plan.T} := \text{plan.T} \cup s_j \prec s_t;$

fin

si incohérences ne peuvent être résolues **alors**

backtrack;

fin



Précond	Action choisie	Contrainte
Avoir un plat de tagliatelles prêt	casser un oeuf sur le plat de tagliatelles	casser oeuf \prec manger tagliatelles
Avoir mélangé tagliatelles et lard	mélanger tagliatelles et lard	mélanger tagliatelles-lard \prec casser oeuf
Tagliatelles cuites	cuire les tagliatelles	cuire tagliatelles \prec mélanger tagliatelles-lard
Avoir des tagliatelles	prendre des tagliatelles	prendre tagliatelles \prec cuire tagliatelles
Lardons cuits	prendre crème	prendre crème \prec mélanger tagliatelles-lard



Section 3

Architecture de planification avec contraintes temps réel : les STN



- Notions de base
- Planification contrainte
- Architecture de planification avec contraintes temps réel : les STN
 - Définitions
 - Exemples
 - Implémentations par agents
 - STN: Conclusion
- Planification discursive

Subsection 1

Définitions



STN: Simple Temporal Network

Définition [Dechter, 1991]

Un réseau temporel simple est un graphe dans lequel les arcs sont étiquetés par des bornes inférieures et supérieures. Les noeuds du graphe représentent des *points temporels*, alors que les arcs correspondent aux contraintes de temps entre événements.

Définition

Un STN est défini par un quadruplet $\langle N, E, l, u \rangle$ où N est un ensemble de noeuds, E un ensemble d'arcs,

$$l : E \rightarrow \mathcal{R}$$

et

$$u : E \rightarrow \mathcal{R}$$

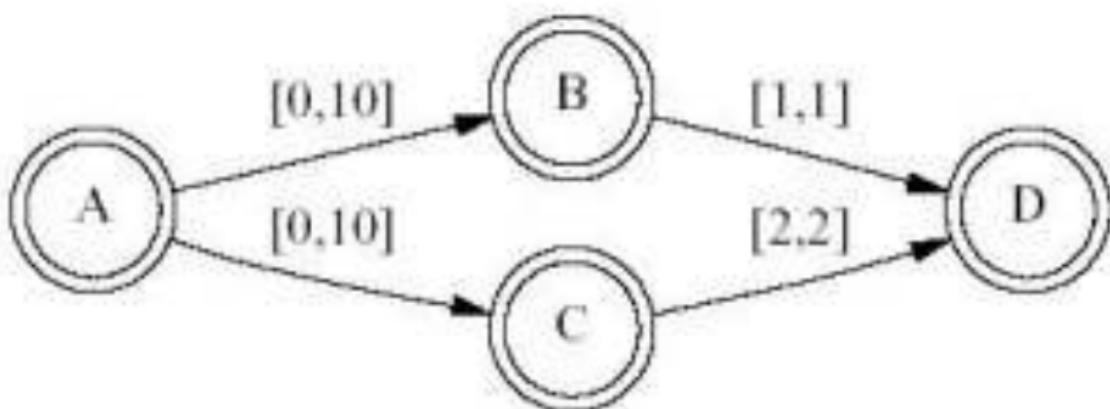
sont des fonctions qui donnent respectivement les bornes inférieures et supérieures d'un arc.

- Chaque STN est associé avec un *graphe de distances*
- Les distances sont calculées à partir des bornes inf et sup
- Un STN est consistant ssi le graphe de distance ne contient pas de cycle négatif
- On peut tester la consistance par un algorithme de plus court chemin

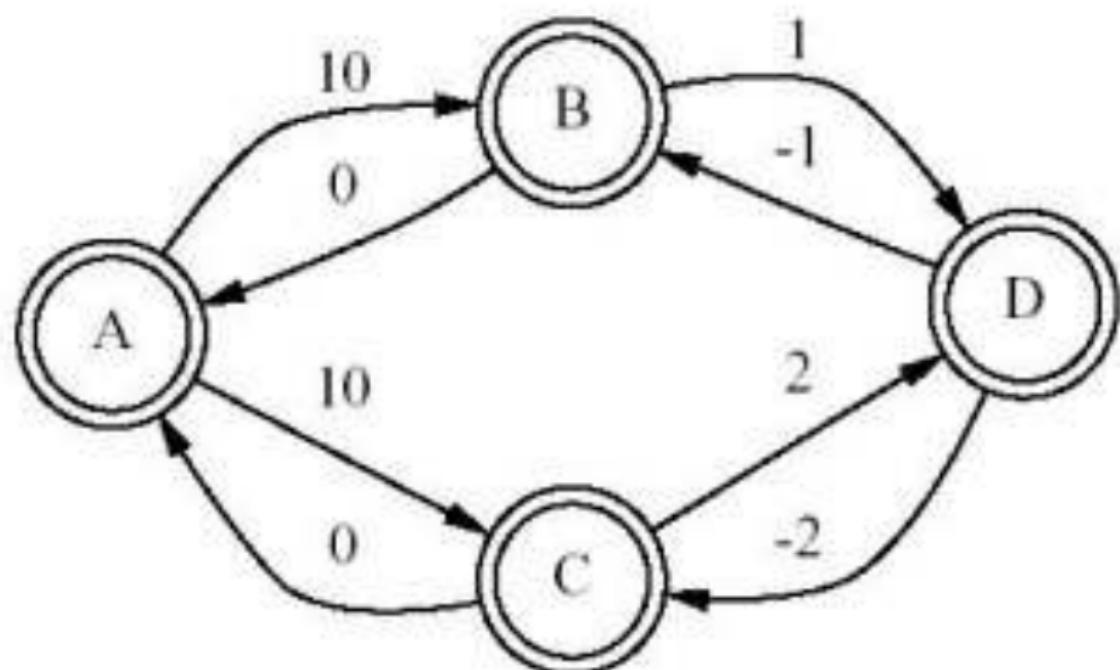


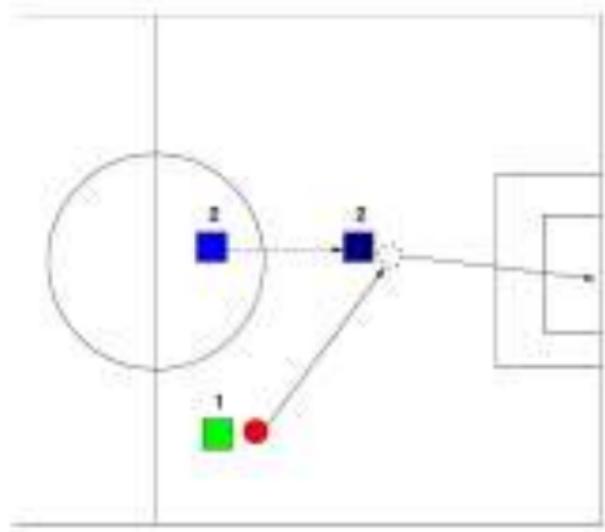
Subsection 2

Exemples

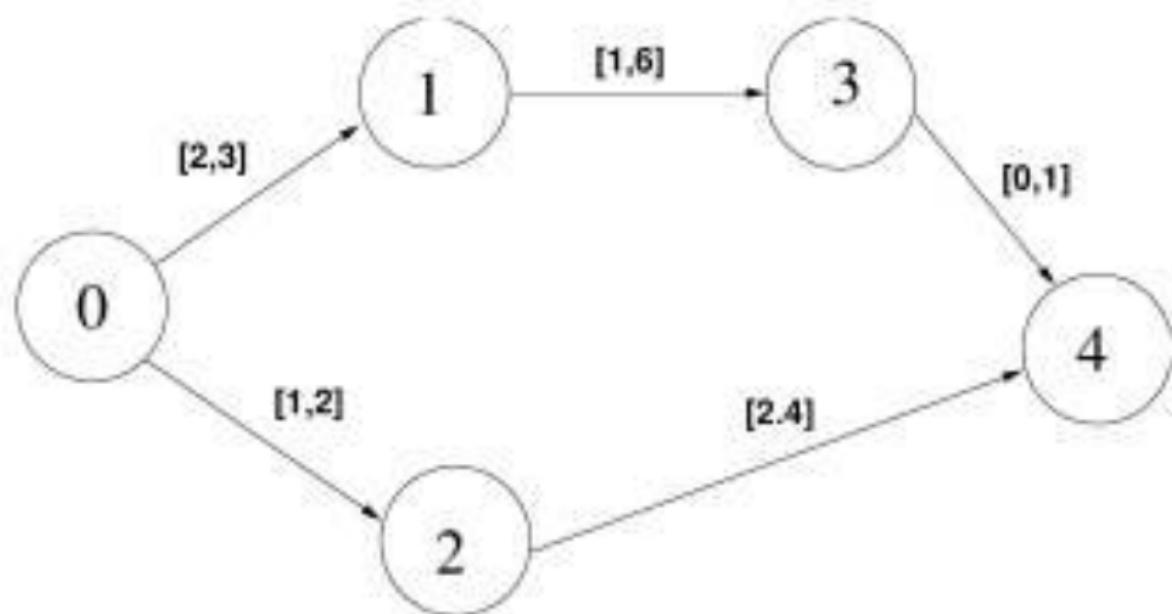


Exemple de graphe de distances





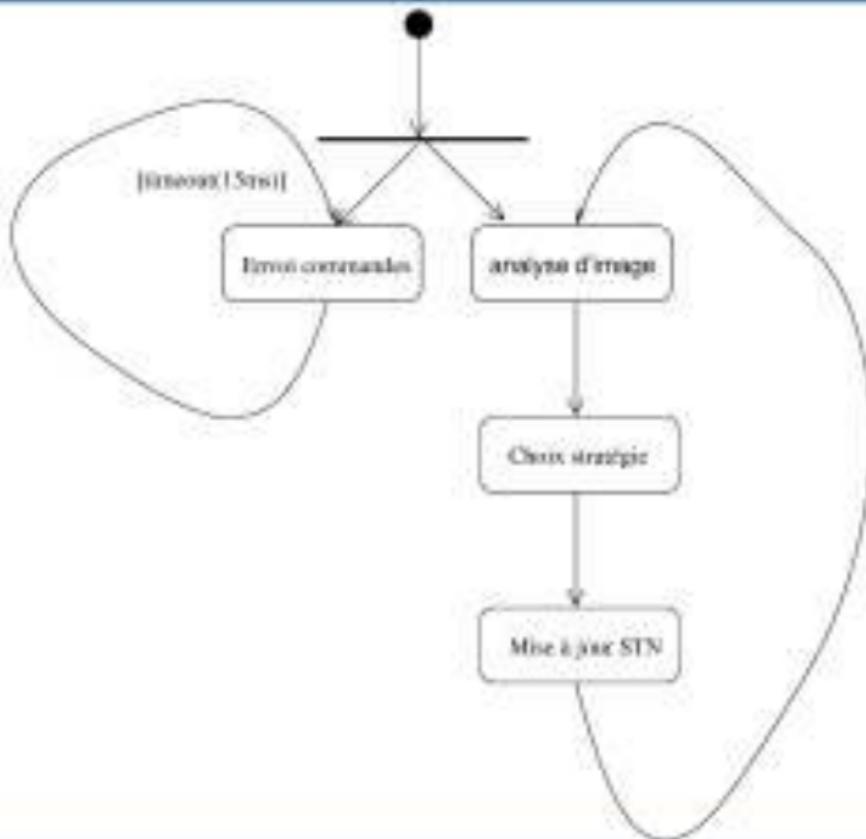
- (0, PASSE_GAUCHE, Robot_1)
- (0, SUPPORT_CENTRE, Robot_2)
- (1, RECEPTION, Robot_2)
- (2, SUPPORT_DROIT, Robot_1)
- (3, TIR_AU_BUT, Robot_2)



- Expertise (action stratégique),
- Apprentissage,
- Négociation.

- Bibliothèque de STN : 1 STN → une action, (construction d'un plan temporel valide composé d'actions horodatées)
- Un analyseur de situation,
- Un coordinateur qui sélectionne un nouveau STN ou laisse actif le STN courant.

- Chaque STN est associé à une valeur fitness
- Cette valeur fitness est remise à jour lors de l'exécution en fonction de la réussite/échec du STN





Subsection 3

Implémentations par agents

- Planification centralisée pour agents multiples:
Un agent planifie/coordonne les tâches et les affecte aux différents noeuds et demande l'exécution aux agents.
- Coordination centralisée pour plans partiels:
Chaque agent construit un STN qu'il envoie à un coordinateur qui vérifie la cohérence de l'ensemble des contraintes
- Coordination distribuée pour plans partiels:
Chaque agent construit un STN qu'il envoie aux autres agents pour vérifier la cohérence.

Subsection 4

STN: Conclusion

• STN is a planning system for the design of complex systems.

• STN is a planning system for the design of complex systems.

- Représentation figée (une seule stratégie par STN)
- Synchronisation "donnée" par les contraintes temporelles
- Propriétés temporelles sur l'ensemble des actions
- Apprentissage par renforcement possible



Section 4

Planification distribuée

Chapitre 10



- Planification centralisée pour plans distribués
- Planification distribuée pour plans centralisés
- Planification distribuée pour plans distribués



Données: S un modèle du monde, G un ensemble de buts, O un ensemble d'opérateurs, A un ensemble d'agents

$\Pi := \text{POP}(S, G, O);$

$(\pi_1, \dots, \pi_n) := \text{decompose}(\Pi);$

$\text{insertSynchro}(\pi_1, \dots, \pi_n);$

$\text{allocateSubPlans}(A, \pi_1, \dots, \pi_n);$

$\text{initiatePlanExec}(\pi_1, \dots, \pi_n);$



- Le processus de planification peut être complexe (expertise)
- La construction d'un plan peut se faire de manière collaborative (multi-expertise)
- Chaque agent construit une partie d'un plan global
- Chaque agent communique pour synchroniser et faire évoluer la solution

- Solution purement agent
- Plans et mécanismes de planification distribués au sein de chaque agent
- Exécutions distribuées



- 1 Chaque agent construit un plan totalement ordonné
 - 2 Collecte des plans pour former un plan global
 - 3 Examen des situations de conflit et résolution (commutation, synchronisation, pause, etc)
- ⚠ stratégie exponentielle dans l'examen des situations de conflit (entrelacement)



- Plutôt que générer un plan global on va procéder itérativement
- A chaque étape (opérateur) les agents examinent l'état du monde et se coordonnent pour arriver chacun à leur but



- Approches avec plans très détaillés à coordonner opérateurs détaillés exhaustivement, contraintes temporelles entièrement décrites
- Approches avec technique générique de coordination

Mais le problème c'est que

- Planification doit être révisée
- Coordination doit être remise en cause

⇒ **Partial Global Planning (PGP)** [Durfee, 1987].



- Les agents commencent par planifier des interactions coordonnées au mieux de leurs connaissances
- réagissent aux situations non prévues en modifiant leurs plans, ici les agents construisent des plans et partagent ces plans pour identifier des améliorations potentielles pour leur coordination.

1 Tâches décomposées agents → sous-tâche ou aucune vue globale

Il n'est pas nécessaire pour un agent d'avoir une vue globale pour accomplir ses tâches

2 Plans locaux incertitudes, redondances, plans partiels

Les différentes façons d'arriver à l'accomplissement des tâches

3 Abstraction de plans

pour se coordonner de manière efficace il faut pouvoir identifier les étapes qui nécessitent interaction

les étapes du plan sont décrites à différents niveaux d'abstraction

4 Communication nécessaire pour échanger des plans abstraits

⇒ identification activités communes

selon une météo-organisation (fLOT de contrôle, visibilité,) planifiées à partir des PGP



- 1 **Plans locaux** : organisation des activités futures de l'agent (court/moyen terme)
- 2 **Noeuds-plan** : résumé des plans locaux pour échange avec autres agents
- 3 **Plans globaux partiels** : information sur le déroulement global des activités (objectifs, résultats attendus, ...) concernant un sous-groupe d'agents

- 1 Identification des buts globaux partiels subsomption de buts à partir des plans locaux
- 2 Construction et modification de plans globaux optimisation des tâches, évitement redondance, ...
- 3 Planification des communications optimisation des envois de résultats
- 4 Raffinement du plan au niveau local

- 1 Seuil de déclenchement de modification réactif mais pas trop...
- 2 Redistribution de tâches pour les débordés...

$$\text{Plan local} = (n, t_{\text{create}}, G, S_{lt}, D_{st}, P_t, P_c, r)$$

- n : nom du plan
- t_{create} : date de création
- G : ensemble de buts à atteindre
- S_{lt} : stratégie à long terme
- D_{st} : détails à court terme
- P_t et P_c : estimation en temps et confiance du plan
- r : coefficient d'importance du plan

Plan global partiel = (n , t_{create} , $P_{component}$, G , S_{lt} , I_{it} , r)

- n : nom du plan
- t_{create} : date de création
- $P_{component}$: plans
- S_{lt} : stratégie à long terme
- I_{it} : interactions à réaliser
- r : coefficient d'importance du plan

$$S_{lt} = list < a_i >$$

Liste ordonnée de "planned-actions" a_i (étape majeure du plan):

$$(D_i, P_i, t_{est-start}, t_{est-end}, abres_{est-partial})$$

- D_i : l'ensemble des données à traiter
- P_i : les procédures à appliquer aux données
- $t_{est-start}$: date estimé de début
- $t_{est-end}$: date estimée de fin
- $abres_{est-partial}$: niveau de confiance dans le résultat partiel, dérivés de P_t et P_c

D_{st} : un ensemble d'opérations de base (étape atomique du plan)

- Plans locaux
- Noeuds-plan = plans locaux synthétisés
- PGP plans avec infos activités globales

- 1 Pour l'ordonnancement courant assigner une valeur aux actions individuelles et sommer;
- 2 Pour chaque action examiner les actions futures et trouver celle qui a la plus grande valeur;
- 3 Si le nouvel ordonnancement a une somme de valeurs supérieure à l'ancien échanger et recommencer à l'étape 2;
- 4 Renvoyer l'ordonnancement;

Algorithm 4: Algorithme PGP

- Plusieurs techniques pour planifier
- Problème d'optimisation
- Méthode rigide mais efficace pour des schémas de coordination "cablés"



Thank you for your attention...

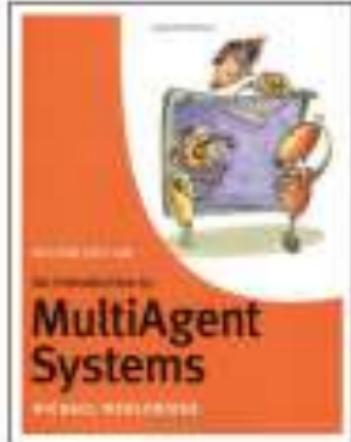
Appendix

Section 1

Books

Subsection 1

Multiagent Systems



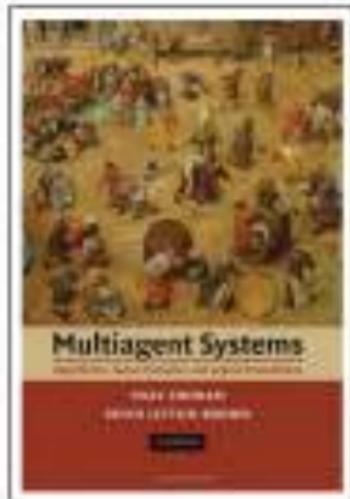
An Introduction to Multiagent Systems

2nd edition

Michael WOOLDRIDGE

Wiley, May 2009

ISBN 0-47-0519460

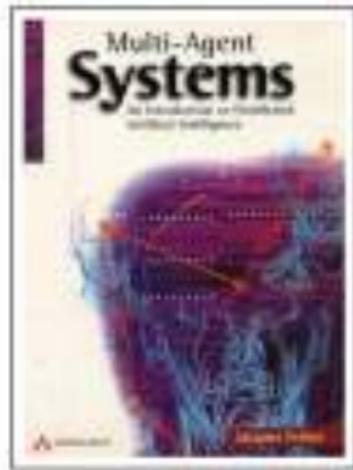


Multiagent Systems: algorithmic, game-theoretic, and logical foundations

Yoav SHOHAM and
Kevin LEYTON-BROWN

Cambridge University Press, 2008

ISBN 0-52-1899435

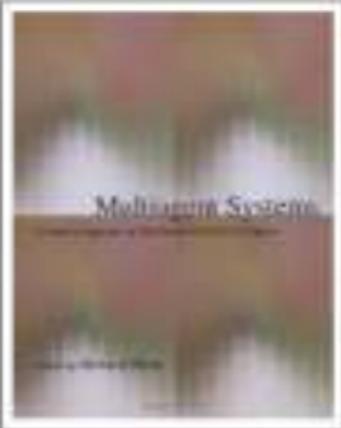


Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence

Jacques FERBER

Addison Wesley, 1999

ISBN 0-20-1360489



Multiagent Systems: a modern approach to distributed Artificial Intelligence

Gerhard WEISS

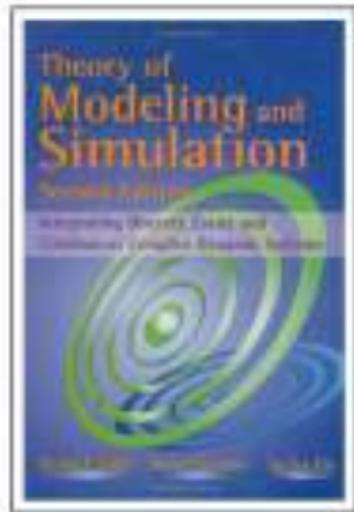
MIT Press, 2000

ISBN 0-26-2731312



Subsection 2

Simulation Theory



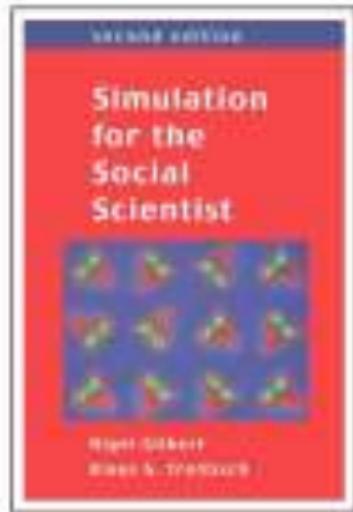
Theory of Modeling and Simulation

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 0-335-216005

Subsection 3

Games and Serious Games

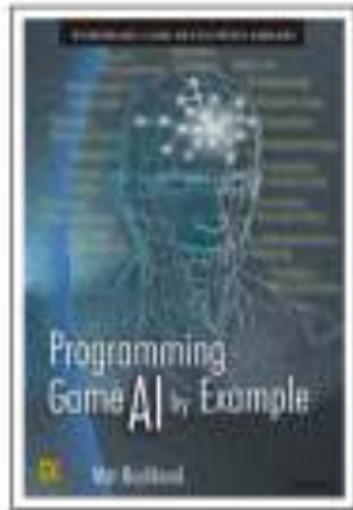


Artificial Intelligence for Games

Ian MILLINGTON

Morgan Kaufmann Publishers & Elsevier
Science, 2006

ISBN 0-12-4977820



Programming Game AI by Example

Mat Buckland

Morgan Kaufmann Publishers & Elsevier
Science, 2006

ISBN 9-78155622-078-4

Subsection 4

Transport



Multi-Agent Systems for Transportation Planning and Coordination

Hans MOONEN

Erasmus Research Institute of
Management, 2009

ISBN 978-90-5892-216-8



Subsection 5

Mathematics



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

2nd edition

James VAN VERTH and Lars BISHOP

Wordware Publishing Inc., 2005

ISBN 0-12-3742971



Calculabilité, Complexité et Approximation

Jean-François REY

Vuibert France, 2004

ISBN 2-71-1748081



Section 2

About the Authors



Professor

Université de Bourgogne Franche-Comté

Université de Technologie de Belfort-Montbéliard, France



**Topics: Multiagent systems,
Multiagent-based simulation, Agent-Oriented
Software Engineering**

Web page: http://www.multiagent.fr/People:Galland_stephane
Email: stephane.galland@utbm.fr

See also his open-source contributions:

- <http://www.sarl.io>
- <http://www.aspects.org>
- <http://www.janusproject.io>
- <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,
Multiagent-based simulation, Agent-Oriented
Software Engineering.**

Web page: http://multiagent.fr/People:Gaud_nicolas

Email: nicolas.gaud@utbm.fr

See also his open-source contributions:

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



Section 4

Bibliography

- Adam, E. (2000).
Modèle d'organisation multi-agents pour l'aide au travail coopératif dans les processus d'entreprise: application aux systèmes administratifs complexes.
PhD thesis, Univ. de Valenciennes et du Hainaut-Cambrésis.
- Baritza, M., Gahet, J., and Kowalew, A. (2004).
Adaptive service discovery and composition in ubiquitous computing
In *International Conference on Information and Communications Technologies: from Theory to Applications*.
- Bordat, B., Cardeca, C., and Zamfirescu, C. (2001).
Holons and Agents in Robotic Teams: A Synergistic approach.
In *Proceedings of ENAS'2001*, pages 634-661.
- Bauer, B., Müller, J. P., and Odeit, J. (2001).
Agent uml: A formalism for specifying multiagent interaction.
In Gancarski, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering*, pages 91–103. Springer.
- Brown, L. and Petrie, T. (1968).
Statistical inference for probabilistic functions of finite state markov chains.
Annals of Mathematical Statistics, 37(8):1254–1263.
- Bergeron, F. and Poggi, A. (2000).
Exploiting uml in the design of multi-agent systems.
In Gretton, A., Tolksdorf, R., and Zaretskii, P., editors, *Engineering Societies in the Agents' World. Lecture Notes in Artificial Intelligence*. Springer Verlag.
- Bézivin, J. (2005).
On the unification power of models.
Software and System Modeling (SoSyM), 4(2):171–186.

- Brazier, F., Jurdie, C., and Tiwart, J. (1998).
Principles of compositional multi-agent system development.
In Cozman, J., editor: Conference on Alternative Technologies and Knowledge Systems (VALTAK'98), page 14. Chaireau and Hall.
- Brecker, P., Giorgini, P., Gutzwiller, F., Mylopoulos, J., and Paletti, A. (2004).
Tropos: An agent-oriented software development methodology.
Journal of Autonomous Agents and Multi-Agent Systems, 8(1):203-226.
- Brooks, R. (1990).
Elephants don't play chess.
Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.
- Brooks, R. and Connell, J. W. (1986).
Asynchronous distributed control systems for a mobile robot.
SPIE, 727.
Mobile Robotics.
- Brooks, R. A. (1989).
A robot that walks: Emergent behaviors from a carefully evolved network.
Technical Report AI-MEMO-1001, MIT.
- Brassel, H. V., Wyns, J., Volckaert, P., Borgaert, L., and Peeters, P. (1996).
Reference architecture for Holonic Manufacturing Systems: PROSA.
Computers in Industry, 31:259-276.
- Burton, J., Galland, S., Good, N., Vysar, Goncalves, M., and Kockari, A. (2013).
Real-time collision avoidance for pedestrian and bicyclist simulation: a smooth and predictive approach.
In the 2nd International Workshops on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS), Halifax, Nova Scotia, Canada. Procedia Computer Science, Elsevier.

- Günther, H., Fischer, K., and G. Marke (1998).
Transportation Scheduling with Holonic MAS - The TeleTruck Approach.
In *Proc. on Practical Applications of Intelligent Agents and Multiagents*, pages 311-332.
- Cardea, C., Stăicu, M., and Barbu, B. (2002).
Holon - Like Approach for Robotic Soccer.
In *Proceedings of the RoboCup European Workshop 2002*, Arnhem, Holland.
- Cernuzzi, L., Conserviso, M., and Zambonelli, F. (2005).
Process models for agent-based development.
Journal of Engineering Applications of Artificial Intelligence (EAAI), 18(2).
- Chab-Draic, B. (2003).
Partie 4 : Raisonnement pratique.
<http://www.diensis.fr/oblivio/coursmbd05/>
- 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 I.
- Conserviso, M., Gasai, N., Haines, V., Galland, S., and Kaskars, 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, 21(2):200-230.
- Dasgupta, D. and Forrest, S. (1995).
Tool breakage detection in milling operations using a negative-selection algorithm.

- Davidson, P. (2000).
Multi-agent based simulation: Beyond social simulation:
Multi-Agent-Based-Simulation. LNCS series, 1979.
- Dachter, R., Meiri, I., and Pearl, J. (1991).
Temporal constraint networks.
Artificial Intelligence, #461-20.
- Dachter, R. and Pearl, J. (1985).
*Generalized best-first search strategies and the optimality of a**
J. ACM, 32(3):506-536.
- Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009).
Engineering route planning algorithms.
In Lerner, J., Wagner, D., and Zwick, R., editors, *Algorithmics of Large and Complex Networks*, volume 536 of *Lecture Notes in Computer Science*, pages 111-138. Springer Berlin Heidelberg.
- DeLoach, S. A., Wood, M. T., and Spachos, C. H. (2001).
Multiagent systems engineering.
The International Journal of Software Engineering and Knowledge Engineering, 11(3).
- Darfee, E. H. and Lesser, V. R. (1997).
Using partial global plans to coordinate distributed problem solvers.
In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 875-883. Milp, Italy.
(Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Hind and Laszlo Csato, editors, pages 289-309. Morgan Kaufmann, 1993.)
- Edwards, M. (2003).
A brief history of holons.



Bibliography (#5)

- Fahey, J. G., Packard, N. H., and Pearson, A. S. (1986).
The immune system, adaptation, and machine learning.
Physics D, 23:187–204.
- Ferber, J. (1995).
Les Systèmes Multi-Agents: Vers une Intelligence Collective.
D.R. Interditions.
- Ferber, J. (1999).
Multagent Systems: An Introduction to Distributed Artificial Intelligence.
Addison Wesley Professional.
- Ferber, J. and Müller, J. (2000).
Influences and reactions : a model of situated multiagent systems.
In *Second International Conference on Multi-Agent Systems (ICMAS-00)*, pages 72–79.
- Fischer, R., Schilo, M., and Siekmann, J. (2003).
Holonic multiagent systems: A foundation for the organization of multiagent systems.
In *Holonic and Multi-Agent Systems for Manufacturing*, volume 2744 of *LNCS*, pages 73–83. Springer Berlin, Heidelberg.
- Faloutsos, P. A. (1997).
Computer simulation: growth through extension.
Trans. Soc. Comput. Sci. Eng., 18(1):13–20.
- Faggetta, A. (2000).
Software process: a roadmap.
In Fens, A., editor, *IV Conference on Software Engineering (CSE): Future of Software Engineering Track*, pages 25–34, Limerick, Ireland.

- Galland, S., Goud, N., Demange, J., and Kouakam, A. (2009).
Environment model for multiagent-based simulation of 3D urban systems.
In the 2th European Meeting on Multiagent Systems (EUMAS2009), Riva Ridge, Canada.
Paper 36.
- Galland, S., Goud, N., Demange, J., and Kouakam, A. (2014).
Multi-level model of the 3D virtual environment for crowd simulation in buildings.
In 2nd International Workshop on Agent-based Modelling and Simulation of Cities (AgentCités'14), pages
823–827, Halifax, Noufieud, Désert.
Proceedings Computer Science, vol. 11.
- Galland, S., Goud, N., Yarie, A.-U.-H., Knappe, L., Jansema, O., and Larivière, O. (2013).
Simulation model of carpooling with the Janos 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.
- Goud, N. (2007).
Holonic Multi-Agent Systems: From the analysis to the implementation. Metamodel, Methodology and Multi-level simulation.
PhD thesis, University of Technology of Belfort-Montbéliard, Belfort, France.
- Gochue, F., Carter, J.-M., Galland, S., Larivière, O., and Kouakam, A. (2012).
Virtual intelligent vehicle urban simulator: Application to vehicle platoon evaluation.
Simulation Modeling Practice and Theory (SIMPAT), 24:103–114.
- Gerber, E., Sickmann, J., and Werke, G. (1999a).
Holonic multi-agent systems.
Research Report RR-99-03, OFRI.

Bibliography (#7)



- Gerber, C., Siekmann, J. H., and Vierke, G. (1998).
Holonic multi-agent systems.
Technical Report UERG-RR-98-03, Deutsches Forschungszentrum für Künstliche Intelligenz - DFKI, Postfach 28 00, 67660 Kaiserslautern, FRG.
- Ghezzi, C., Jatayev, M., and Montirosso, D. (2002).
Fundamentals of Software Engineering.
Prentice Hall, second edition.
isbn: 0133305600-8
- Giorgini, P. and Harrison-Selton, B. (2002).
Agent-Oriented Methodologies, volume ISBN1-59140-581-5, chapter Agent-Oriented Methodologies: an Introduction, pages 1-19.
Elsevier Publishing.
- Gret, A. (2006).
ANEMONA: A Multi-Agent Method for Holonic Manufacturing Systems.
PhD thesis, Universidad Politécnica de Valencia, Valencia, Spain.
- Grauer, P., Hildebrand, V., Rückert, A., and Röver, P. (2004).
Heterogeneous formal specification based on object-z and statecharts: semantics and verification.
Journal of Systems and Software, 78(2):103–105.
- Harel, D. (1987).
Statecharts: A visual formalism for complex systems.
Science of Computer Programming, 8(3):231–274.
Preliminary version: Technical Report CMU-CS-84-125, The Carnegie Institute of Science, Pittsburgh, Penn., February 1984.

- Huetting, D. and Mukar, P. (1997).
Self-organization phenomena in pedestrian crowds.
Self-organization of complex structures: from individual to collective dynamics, pages 560–577.
- Holland, J. H. (1995).
Hidden order: how adaptation builds complexity.
Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
- Hengstendoorn, S. P. and Bovy, P. H. (2001).
State-of-the-art of vehicular traffic flow modeling.
Special issue on Road Traffic Modelling and Control of the Journal of Systems and Control Engineering, 21(4):389–393.
- Hengstendoorn, S. P. and Bovy, P. H. (2004).
Pedestrian route-choice and activity scheduling theory and models.
Transportation Research Part B, 38:169–180.
- Jacobson, T., Christensen, M., Jørgensen, P., and Omsgaard, G. (1992).
Object-oriented software engineering: A use case driven approach.
København, MFA, Addison Wesley.
- Jerrum, M. R. (1974).
Towards a network theory of the immune system.
Annals of Immunology (Inst. Pasteur), 125C:373–389.
- Kendall, E. A. (2000).
Role modeling for agent system analysis, design, and implementation.
IEEE Transactions, 8(1):34–41.

Bibliography (#9)

- Kre, Y. H. (1996).
Micro-robot world cup soccer tournament.
KAIST.
- Koenig, D., Georgoff, M., and Rao, A. (1996).
A methodology and modelling technique for systems of BDI agents.
In van Hoorn, B., editor, *Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World*. Elektrotechniek, The Netherlands.
- Koenig, S. and Likhachev, M. (2005).
Fast replanning for navigation in unknown terrain.
Robotics: IEEE Transactions on, 21(3):394–398.
- Hofstadter, D. (1987).
The Ghost in the Machine.
Hutchinson.
- King, D. (1995).
Holonic Manufacturing Systems: Case Study of an IAMS Consortium.
<http://www.holonicmanufacturing.de/galilei/feasibility/study.htm>
- Lawson, B. and Park, S. (2002).
Asynchronous time evolution in an artificial society model.
Journal of Artificial Societies and Social Simulation, 3(1).
- Lack, M. and d'Istria, M. (1995).
A formal framework for agency and autonomy.
In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 294–299. San Francisco, CA, USA: AAAI Press.



Bibliography (#10)

xxi

- Maes, P. (1989).
How to do the right thing.
Technical Report AI-TR-118, MIT Artificial Intelligence Laboratory.
- Maturana, F., Shek, W., and Norm, D. (1999).
MetaMorph: An Adaptive Agent-based Architecture for Intelligent Manufacturing.
- Michel, F. (2001).
Le modèle influence/réaction pour la simulation multi-agents.
In *Proceedings Journées Francophones Modèles Parallèles et d'Intégration (JMPI)*, Toulouse, France.
- Michel, F. (2004).
Formalism, tools and methodological elements for the modeling and simulation of multi-agents systems.
PhD thesis, IUTIM, Montpellier, France.
- Michel, F. (2007).
The IRM4S model: the influence/reaction principle for multiagent based simulation.
In *ADVIS'07: Proceedings of the 8th international joint conference on Autonomous agents and multiagent systems*, pages 1–1. New York, NY, USA: ACM.
- Michel, J. (1983).
A critical view of driver behaviour models: What do we know, what should we do?
Human Behavior and Traffic Safety, pages 401–529.
- OCL (2006).
Object Constraint Language (OCL) Specification, v2.0, OMG Available Specification, formal/06-05-01.
Object Management Group (OMG).



Bibliography (#11)

6/61

- Odeh, J., Nodine, M., and Levy, R. (2005).
A micromodel for agents, roles, and groups.
In Odeh, J., Gengler, P., and Müller, J., editors, *Agent-Oriented Software Engineering (AOSE) IV: Lecture Notes in Computer Science*. Springer.
- Odeh, J., Parvath, H., and Bauer, B. (2000).
Extending uml-for agents.
In Wagner, E. F. C. and Leporato, V., editors, *Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17.
- Odeh, J., Van Dyke Parvath, H., and Bauer, B. (2001).
Representing agent interaction protocols in uml.
In Giacomo, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering: First International Workshop (AOSE 2000), Vienna, Austria, volume 1357 of LNCS*, pages 203–218, Leinweck, Ireland. Springer Berlin / Heidelberg.
- Rau, A. S. and Georgell, M. P. (1995a).
BDI agents: from theory to practice.
In Lesser, V., editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 113–118, San Francisco, CA. MIT Press.
- Rau, M. P. G. (1995b).
BDI-agents: From theory to practice.
In the First International Conference on Multiagent Systems (ICMAS'95).
- Reynolds, C. (1999).
Steering behaviors for autonomous characters.
In *Proceedings of the Game Developers Conference*, page 753–762.

Bibliography (#12)



- Rodrigues, 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.
- Rodrigues, S., Hjelm, V., and Koenig, A. (2003).
Towards a methodological framework for holonic multi-agent systems.
In *Workshop of Engineering Societies at the Agents World*, pages 179–188.
- Rolland, C., Prakash, R., and Berjaven, A. (1999).
A multi-model view of process modeling.
Requirements Engineering, 4:109–137.
- Rumbaugh, J., Jacobson, I., and Booch, G. (1999).
The Unified Modeling Language Reference Manual.
Addison-Wesley.
- Shanno, R. E. (1977).
Simulation modeling and methodology.
SIGSM Simul. Dig., 9(3):31–36.
- Simon, H. A. (1986).
The Science of Artificial.
MIT Press, Cambridge, Massachusetts, 3rd edition.
- Sommerville, I. (2004).
Software Engineering.
International Computer Science Series. Addison Wesley, Pearson Education, seventh edition.

Bibliography (#13)



- SPM (2007).
Software Process Engineering Metamodel Specification, v2.0, Final Adopted Specification, pcc/07-03-03.
Object Management Group (OMG)
- Suzuki, J. and Yamamoto, Y. (2000).
A decentralized policy coordination facility in openwebservice
In *Proceedings of ICACI2000*.
- Treiber, M., Helbing, D., and Hennecke, A. (2000).
Congested traffic states in empirical observations and microscopic simulations.
Phys. Rev. E 62:1805–1824.
- Trincaresky, I. and Comerio, R. (2005).
Agent Modeling Language (AML): A comprehensive approach to modeling MAS
Informatica, 29(4):391–406.
- Uleru, M. and Geras, A. (2002).
Emergent hierarchies for e-health applications: a case in glaucoma diagnosis.
In *IEEE ICCHWIE*, volume 6, pages 297– 291.
- Van Lathouwer, A. (2001).
Goal-oriented requirements engineering: A guided tour.
In *3rd IEEE International Symposium on Requirements Engineering (RE)*, page 249-265, Toronto, Canada. IEEE Press.
- Wagner, G. (2003).
The agent-object-relationship metamodel: towards a unified view of state and behavior
Information Systems, 28(5):479–504.

Bibliography (#14)

100

- Watashige, Y., Ichiguro, A., and Ueda, Y. (1999). Decentralized behavior arbitration mechanism for autonomous mobile robot using immune system. In *From Artificial Immune Systems and Their Applications*. Springer-Verlag, p. 189-218. 2000. 3-540-64390-7.
- Weiss, G. (1999). *Multagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA, USA.
- Weyns, O., Parvaz, H. V. D., Michel, F., Holczer, T., and Ferber, J. (2005). Environment for Multiagent Systems State-of-the-Art and Research Challenges. In *Environments for Multi-Agent Systems (EMAS)*, pages 1-47. Springer Berlin / Heidelberg.
- Wilder, K. (1995). *Sex, Ecology, Spirituality*. Shambhala.
- Wooldridge, M. (2003a). Lecture 2: Abstract agent architectures.
- Wooldridge, M. (2003b). Lecture 2: Practical reasoning.
- Wooldridge, M. and Giunchiglia, 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*, page 1-28. Springer-Verlag.
- Wooldridge, M., Jennings, N. R., and Koenig, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285-312.



Bibliography (#15)



Wynn, J. (1999).

Reference architecture for Holonic Manufacturing Systems - the key to support evolution and reconfiguration
PhD thesis, Katholieke Universiteit Leuven.

Zarebski, F., Jennings, N., and Wooldridge, M. (2003).

Developing multiagent systems: the gaia methodology

ACM Trans. on Software Engineering and Methodology 13(3)

Ziegler, B. P., Praehuber, H., and Kim, T. G. (2006).

Theory of Modeling and Simulation

Academic Press, 2nd edition edition.

Zilberman, S. (1996).

Using anytime algorithms in intelligent systems.

AI Magazine 17(3):73-83



Bibliography (#16)



