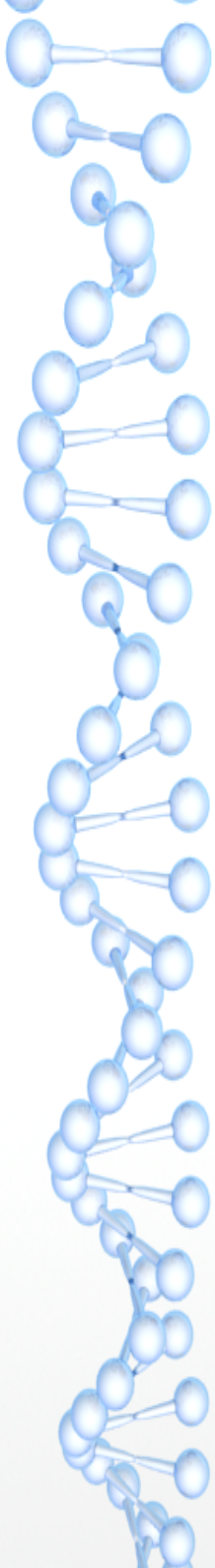
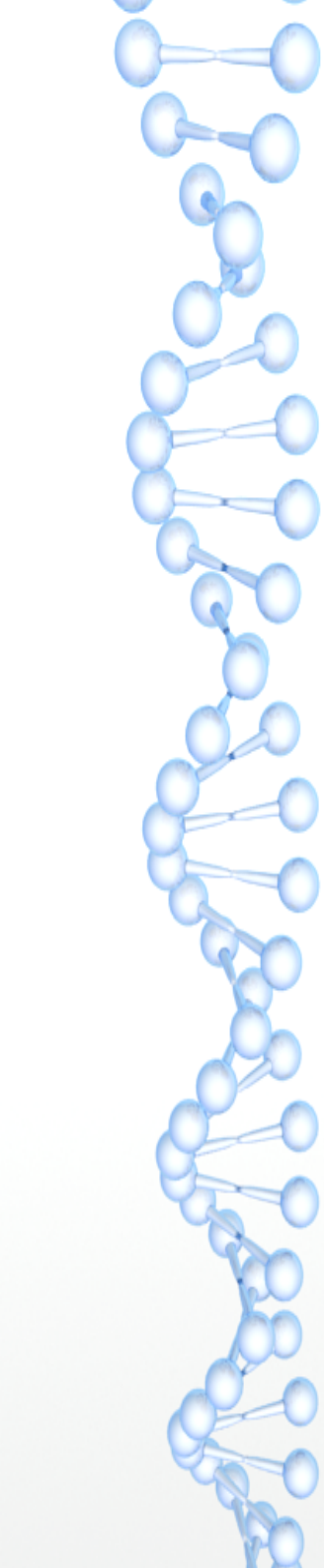


Palteformes multi-agents

Syma – cursus SCIA

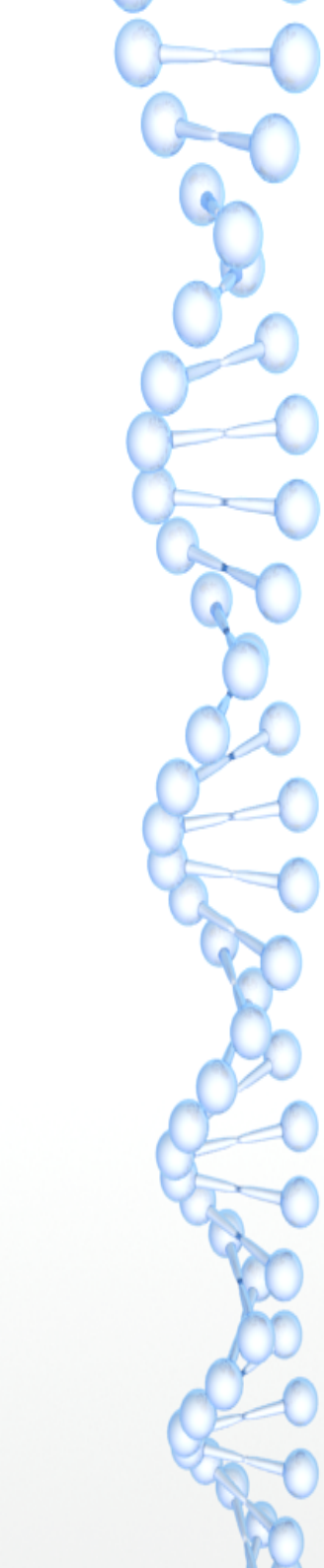
Julien Saunier - julien.saunier@ifsttar.fr

IFSTTAR



Plan

- Madkit/Magique: structure organisationnelle
- Jade: agents cognitifs communiquants
- Netlogo, Cormas: simulation



Madkit

- Plateforme multi-agents générique
- Deux types d'agents:
  - Threadés
  - Turtles
- Gestion des messages et des concepts AGR
- Gestion du cycle de vie des agents
- Ecrite en Java, architecture modulaire par plug-ins
- Distribuée

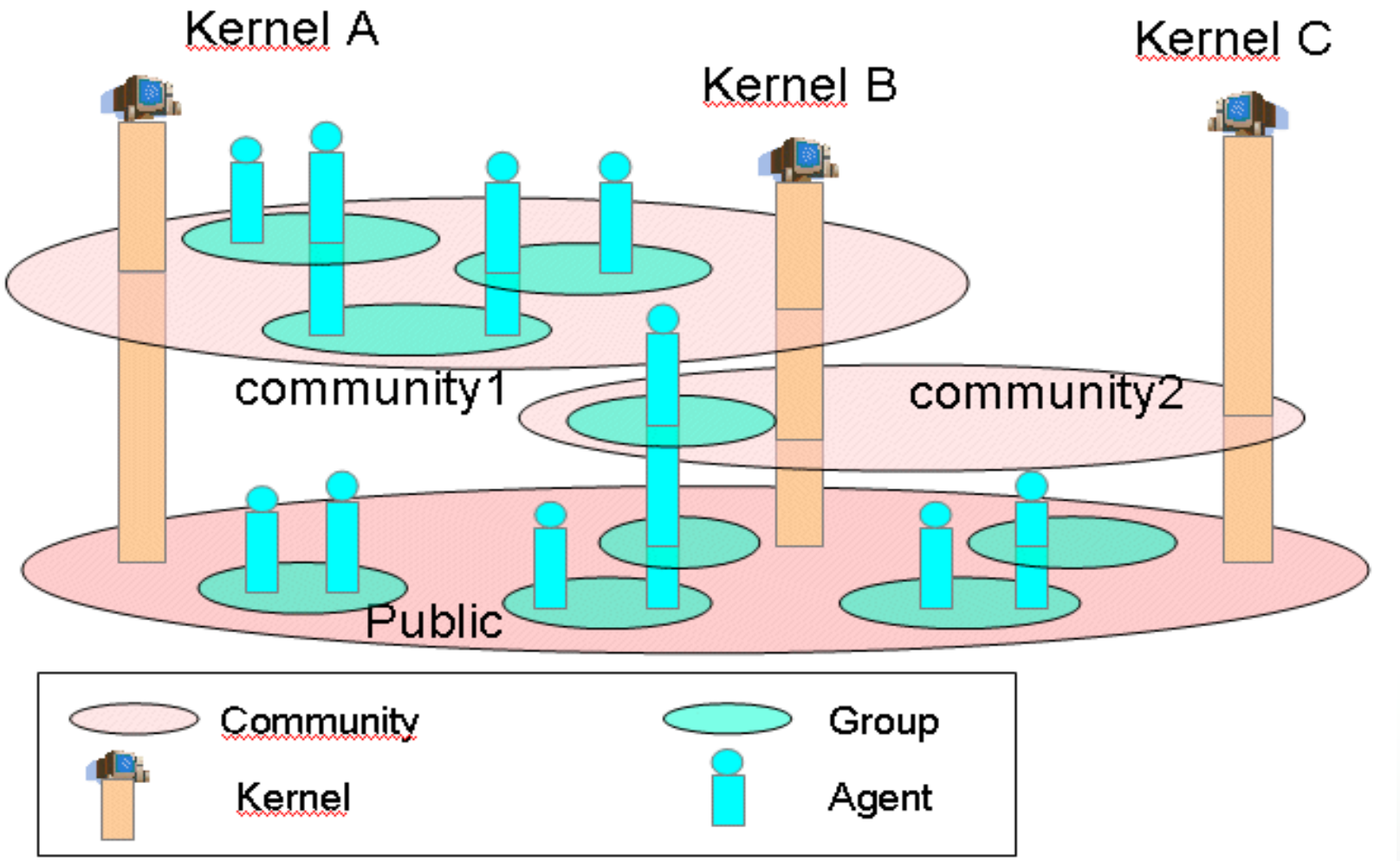
Le modèle Agent / Groupe / Rôle

- Agent:
  - *Une entité autonome et communicante*
    - Joue des ***rôles*** dans des ***groupes***. Un agent peut avoir *plusieurs rôles* et être membre de *plusieurs groupes*.

- Rôle:
  - *la représentation abstraite de la fonction d'un agent.*
    - Les rôles sont *locaux* aux groupesUn rôle peut être joué par *plusieurs agents*
- Groupe:
  - *un ensemble d'agents partageant une caractéristique*
    - Deux agents peuvent communiquer que s'ils sont membres du même groupe.

Madkit

Représentation du modèle AGR



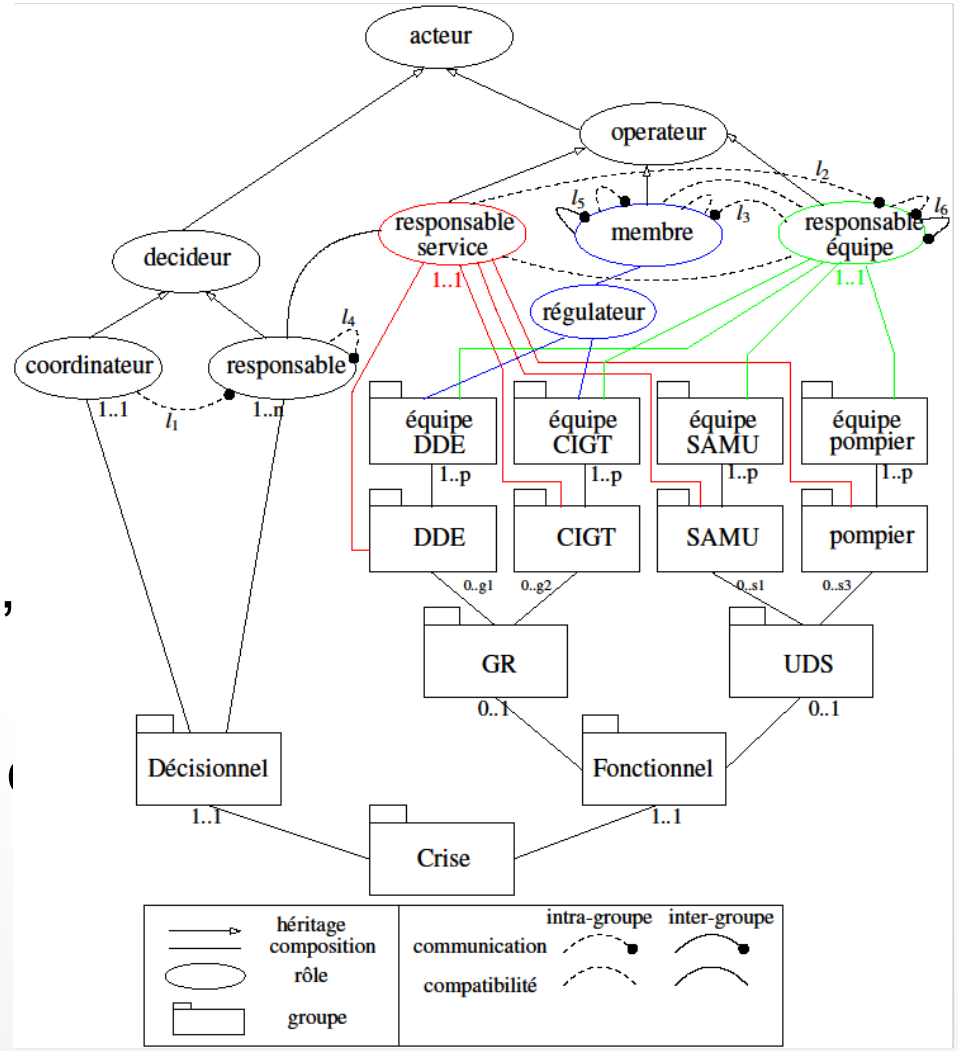
Modèle sous-jacent

$\mathcal{M}$oise

- Spécification Structurelle :

  $< R, \subset, rg >$

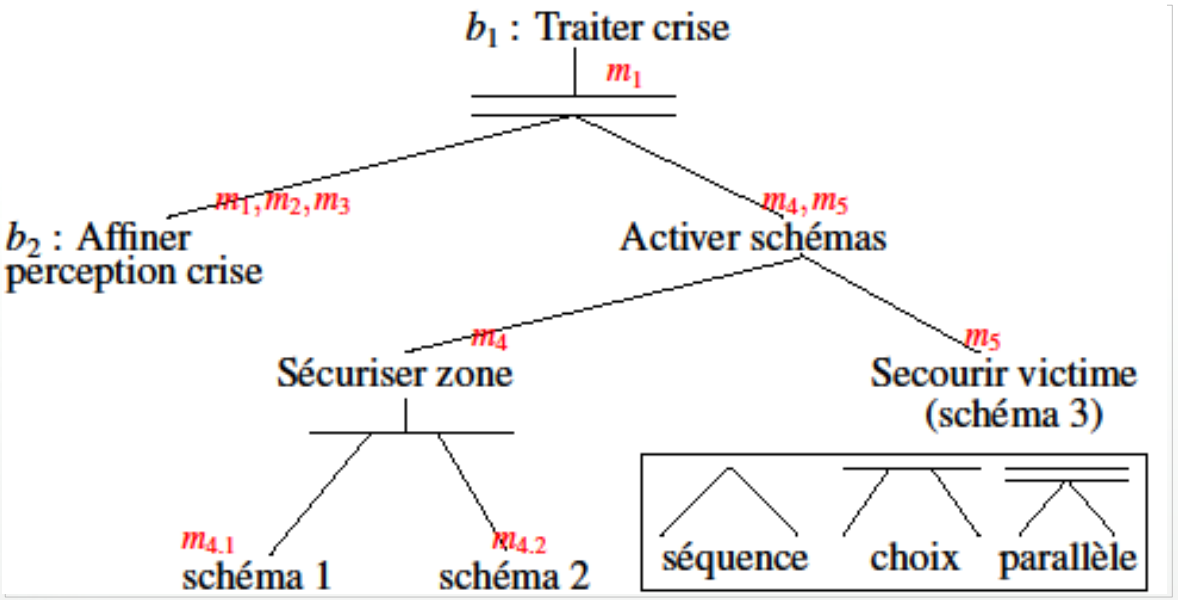
  - R : ensemble des identifiants des rôles,
  - $\subset$: relation d'héritage sur les rôles,
  - rg : spécification du groupe racine (l'organisation.



Modèle sous-jacent

$\mathcal{M}$oise

- Spécification Fonctionnelle : < *M, G, S* >
  - *M* : ensemble des missions,
  - *G* : ensemble des buts collectifs et individuels à satisfaire,
  - S : structurations arborescentes de ces buts en plans.



Exemple: PingPong

- 2 agents du groupe ping-pong ont le même rôle, joueur.
- Un des deux fonde le groupe
- Un agent appartient toujours à un groupe, même seul
- Les agents communique par messages

Exemple: PingPong

```
import madkit.kernel.*;
import madkit.lib.messages.*;

public class PingPong extends Agent
{
AgentAddress other = null;
boolean creator=false;
...
```

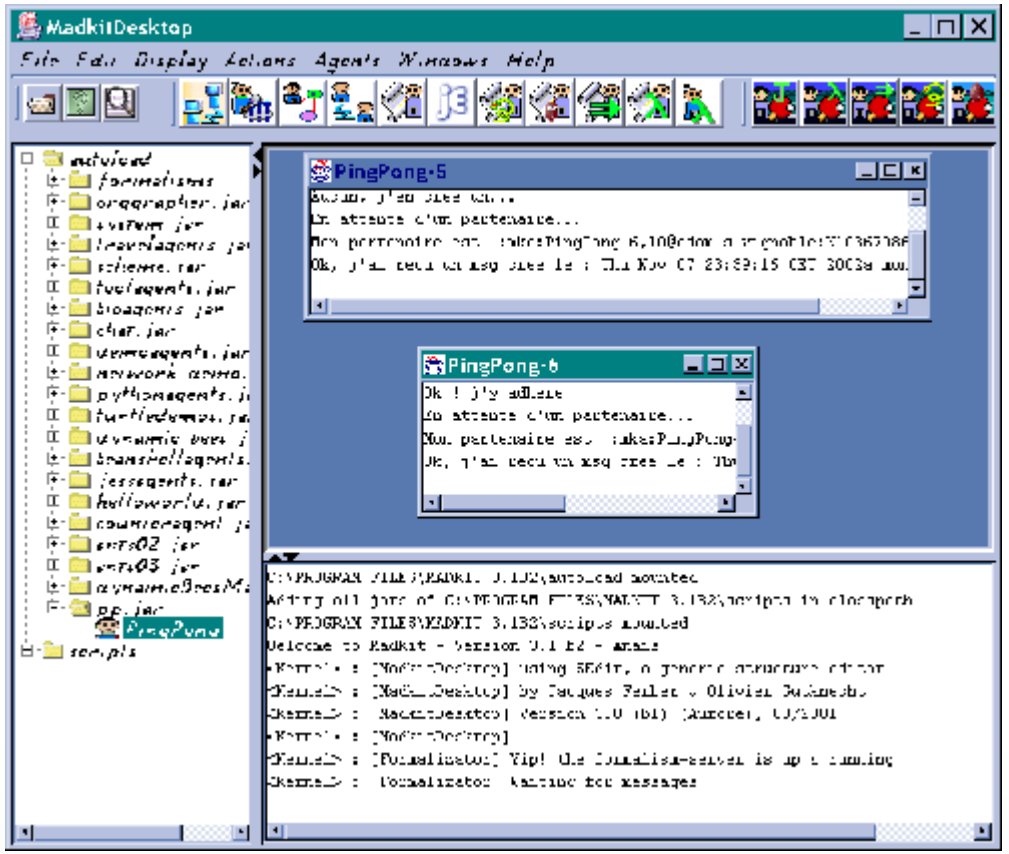
Exemple: PingPong

```
...
public void activate()
{
println("PingPong agent active");
println("recherche le groupe pingpong...");
if (isGroup("pingpong"))
{
println ("Ok ! j'y adhere");
creator=false;
}
else
{
println ("Aucun, j'en cree un...");
createGroup(true,"pingpong",null,null);
creator=true;
}
requestRole("pingpong", "player", null);
}
```

Exemple: PingPong

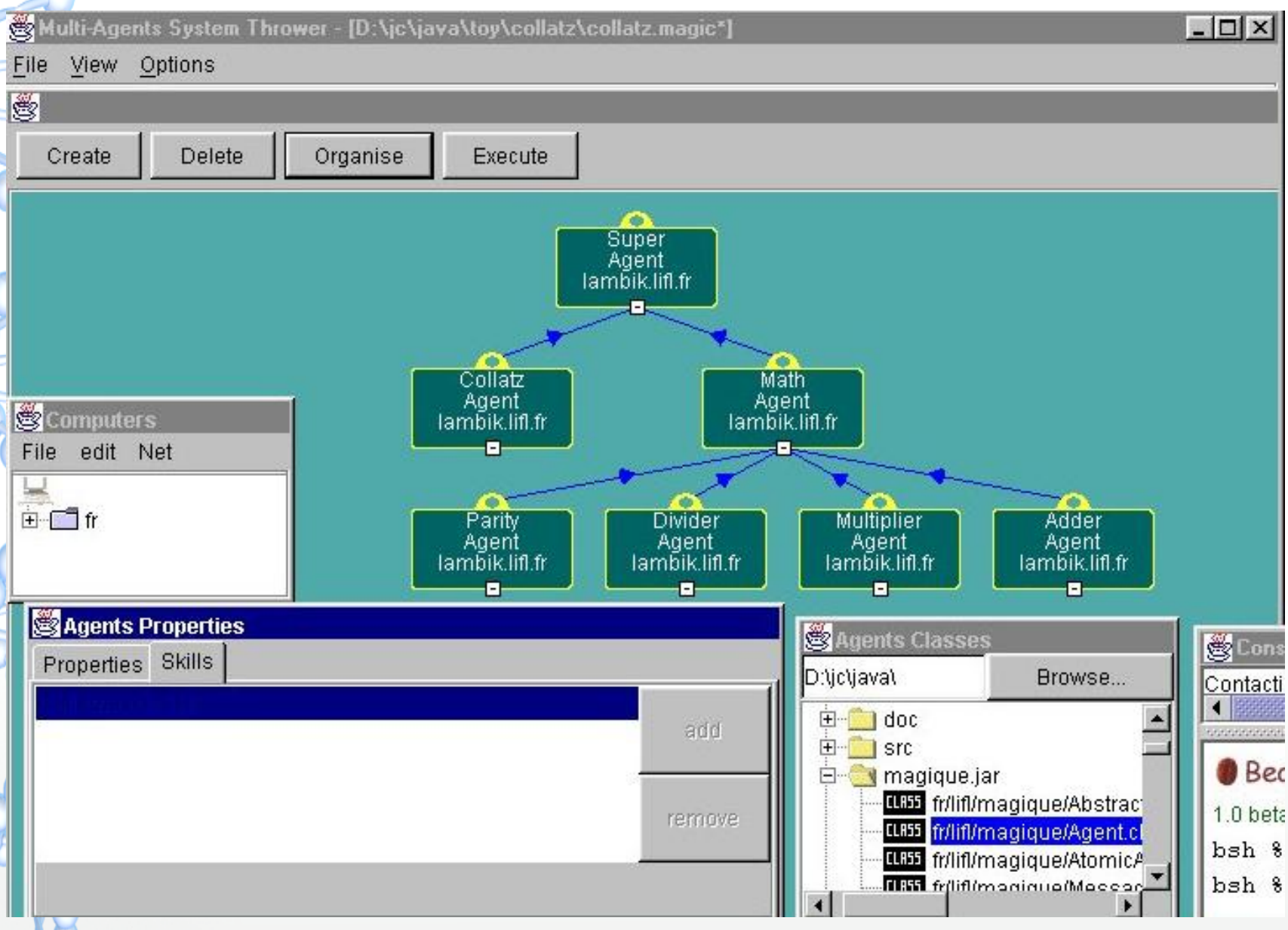
```
// Si je ne suis pas le createur du groupe, j'envoie la la balle en premier
if (! creator)
sendMessage(other, new StringMessage("Ball"));
for (int i = 5; i > 0; i--)
{
Message m = waitNextMessage();
StringMessage ans = (StringMessage) m;
println("Ok, j'ai recu un msg cree le : " +
m.getCreationDate() + "a mon tour ...");
pause(1000);
sendMessage(other, new StringMessage(ans.getString()));
}
}
public void end()
{
println("Bye Bye !!");
println("Fin de agent PingPong
");
}
```

Exemple: PingPong



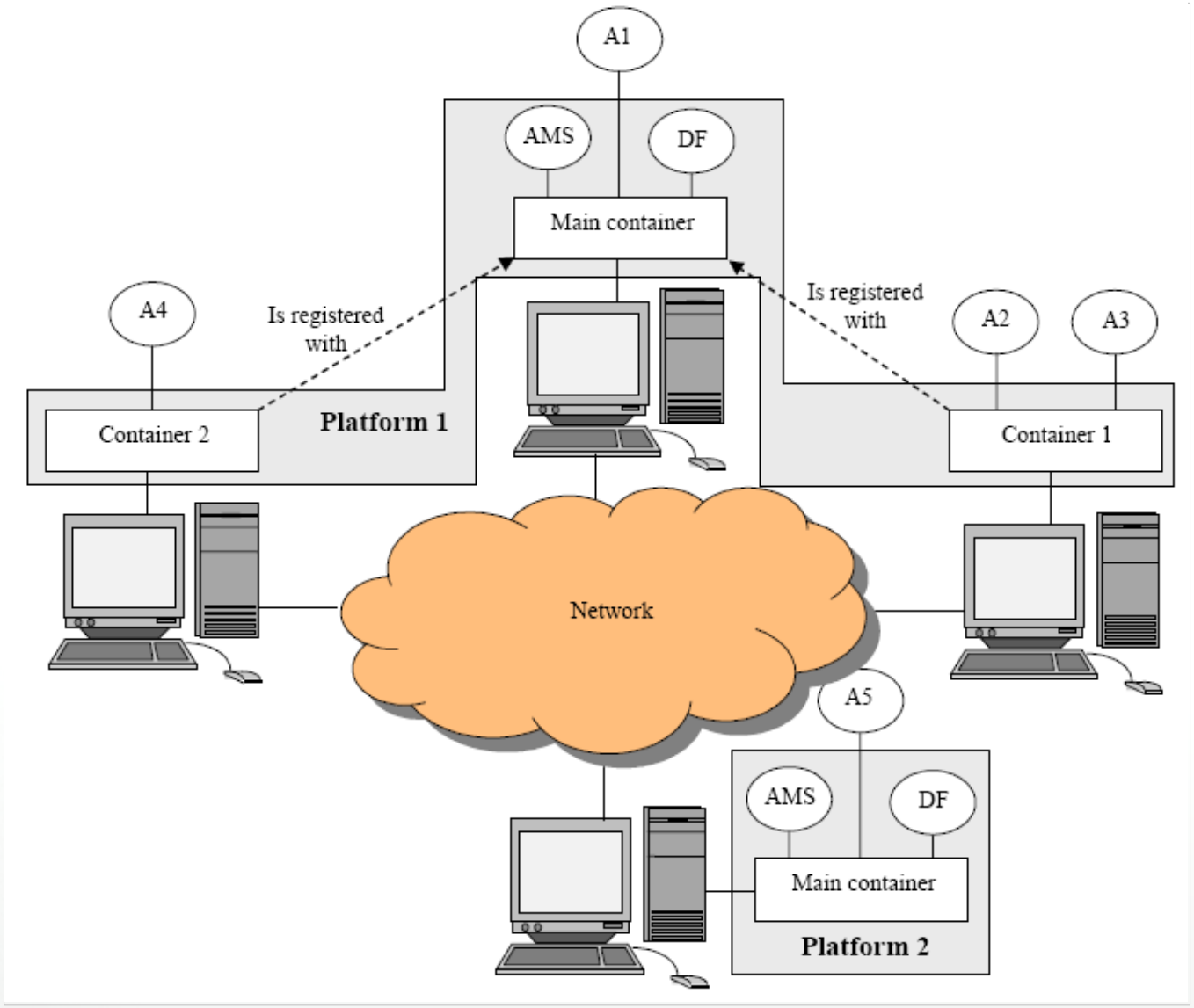
MAGIQUE

Multi-AGent hiérarchIQUE



Agent Cognitif

Plateforme Jade

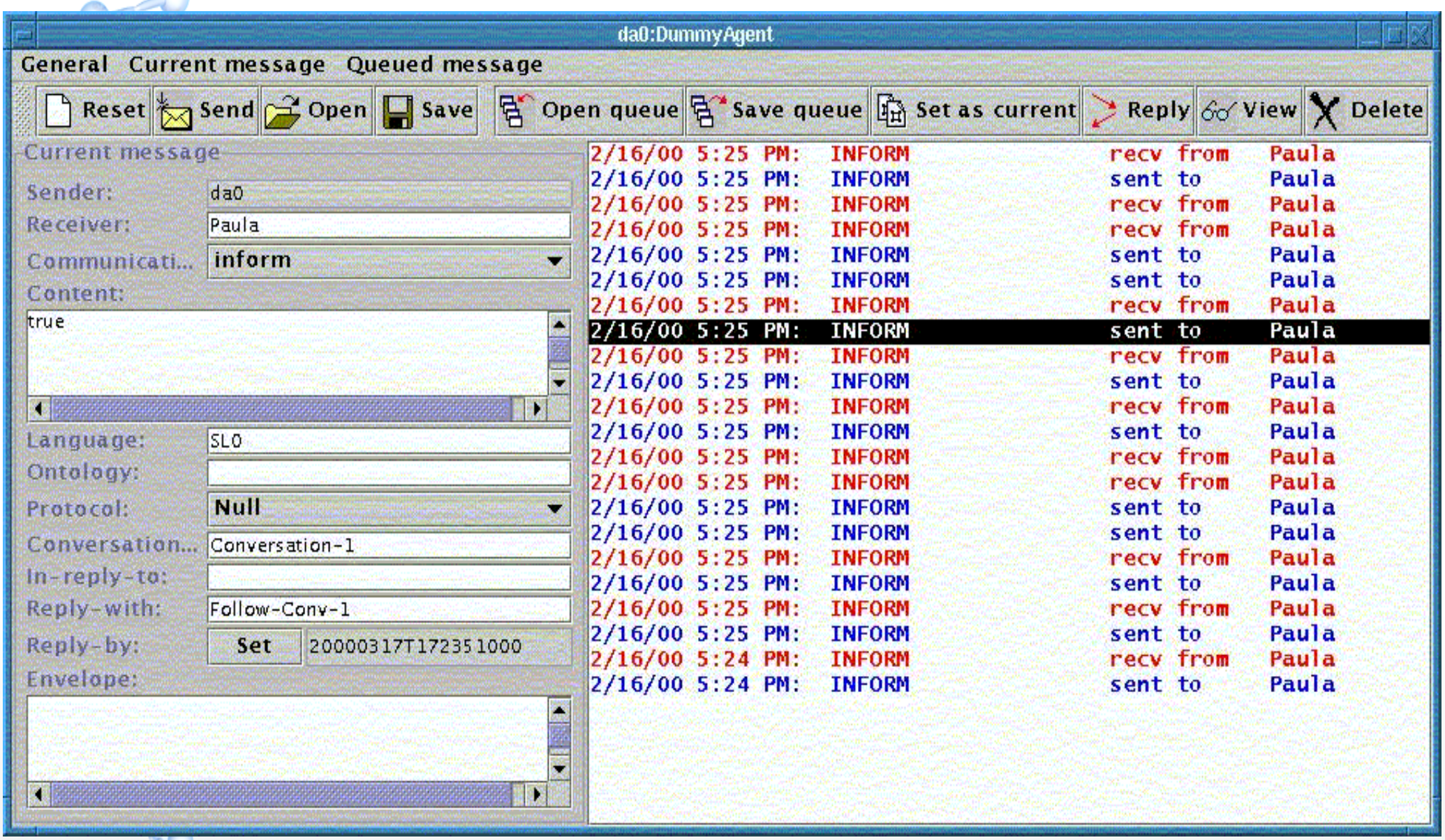


JADE

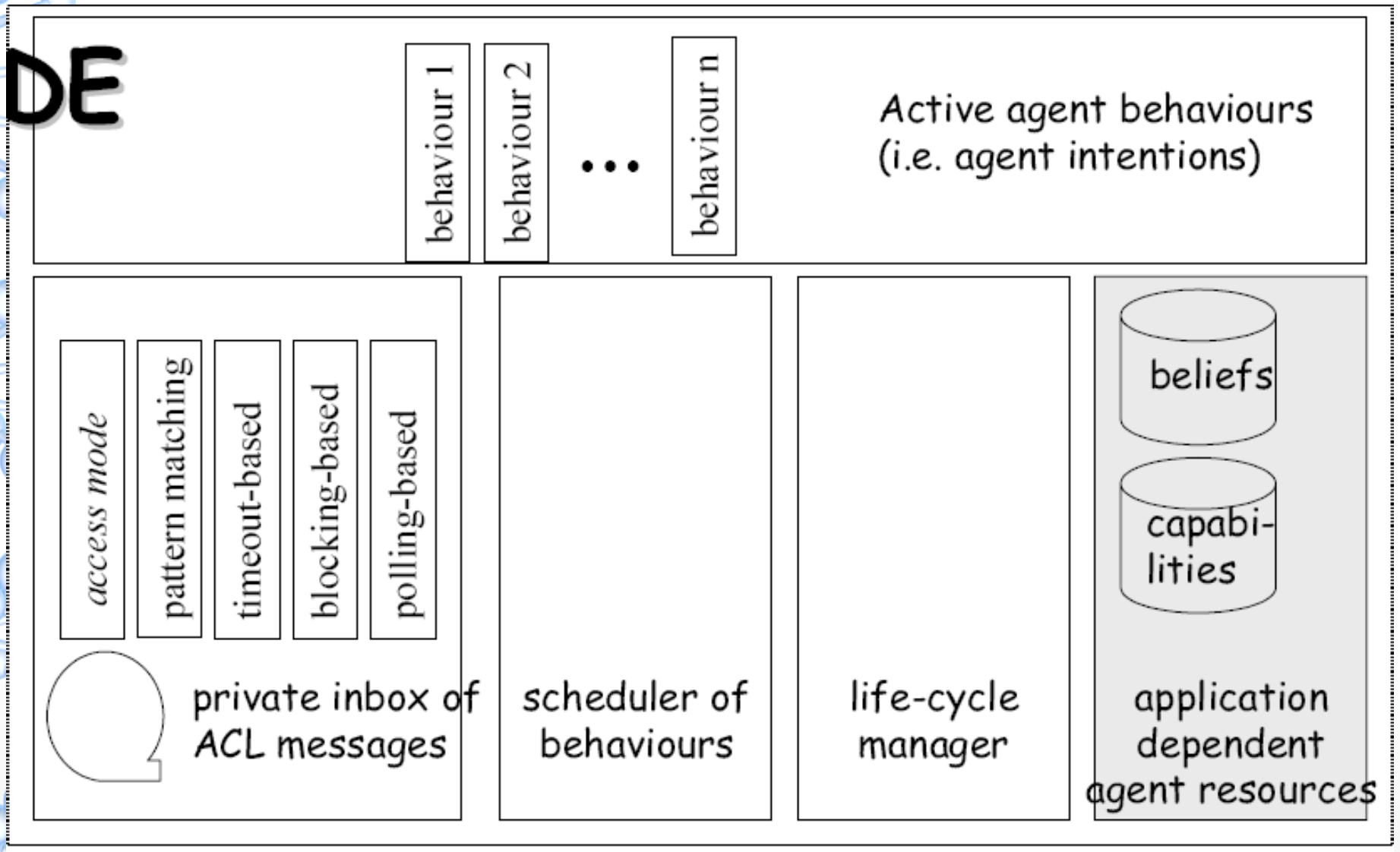
Description

- Services :
  - The Dummy Agent,
  - The Sniffer Agent,
  - The Introspector agent,
  - Remote Management Agent,
  - The DF GUI,
  - The LogManagerAgent,
  - The SocketProxyAgent agent

Jade Service

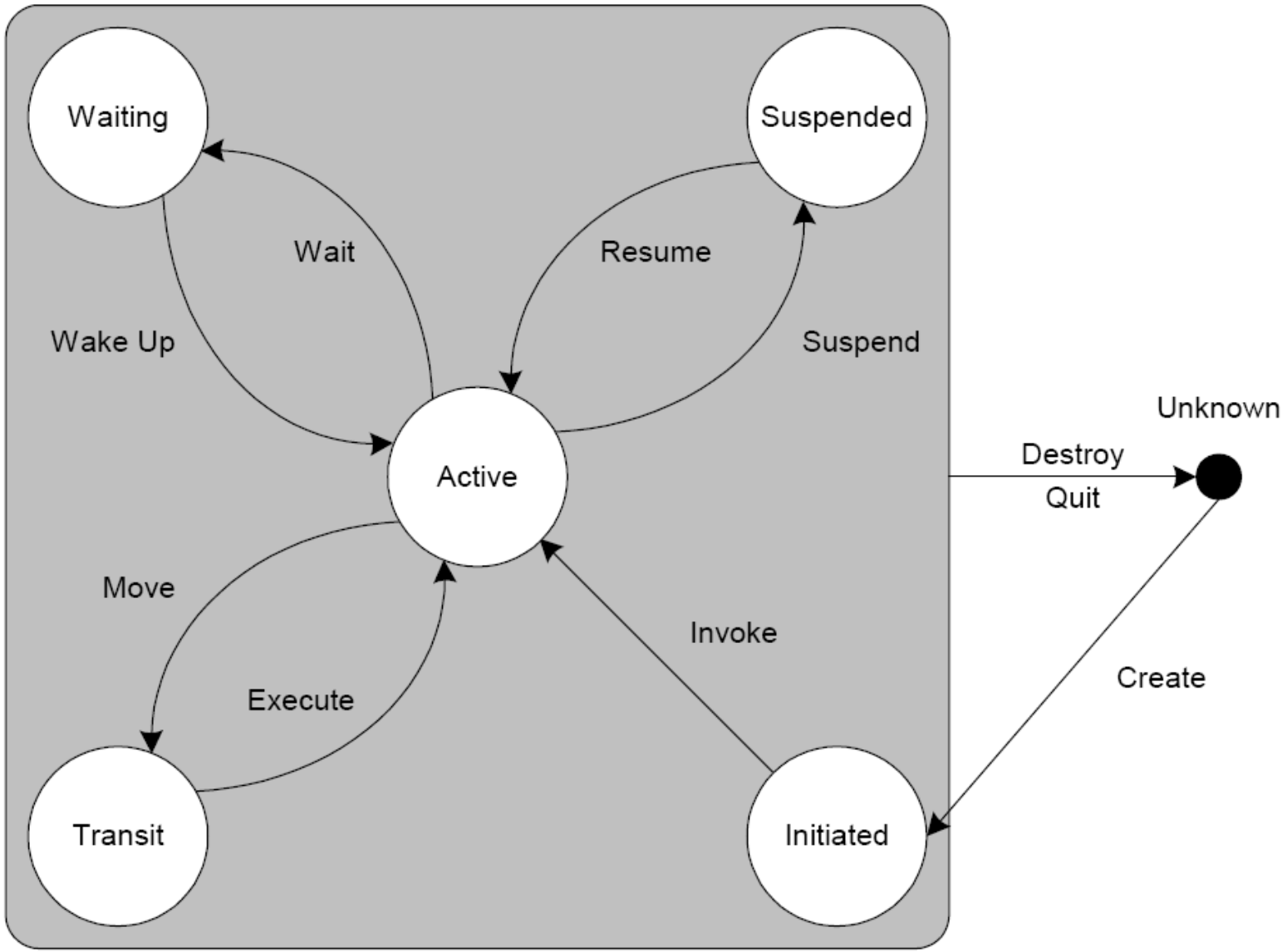


Agent architecture

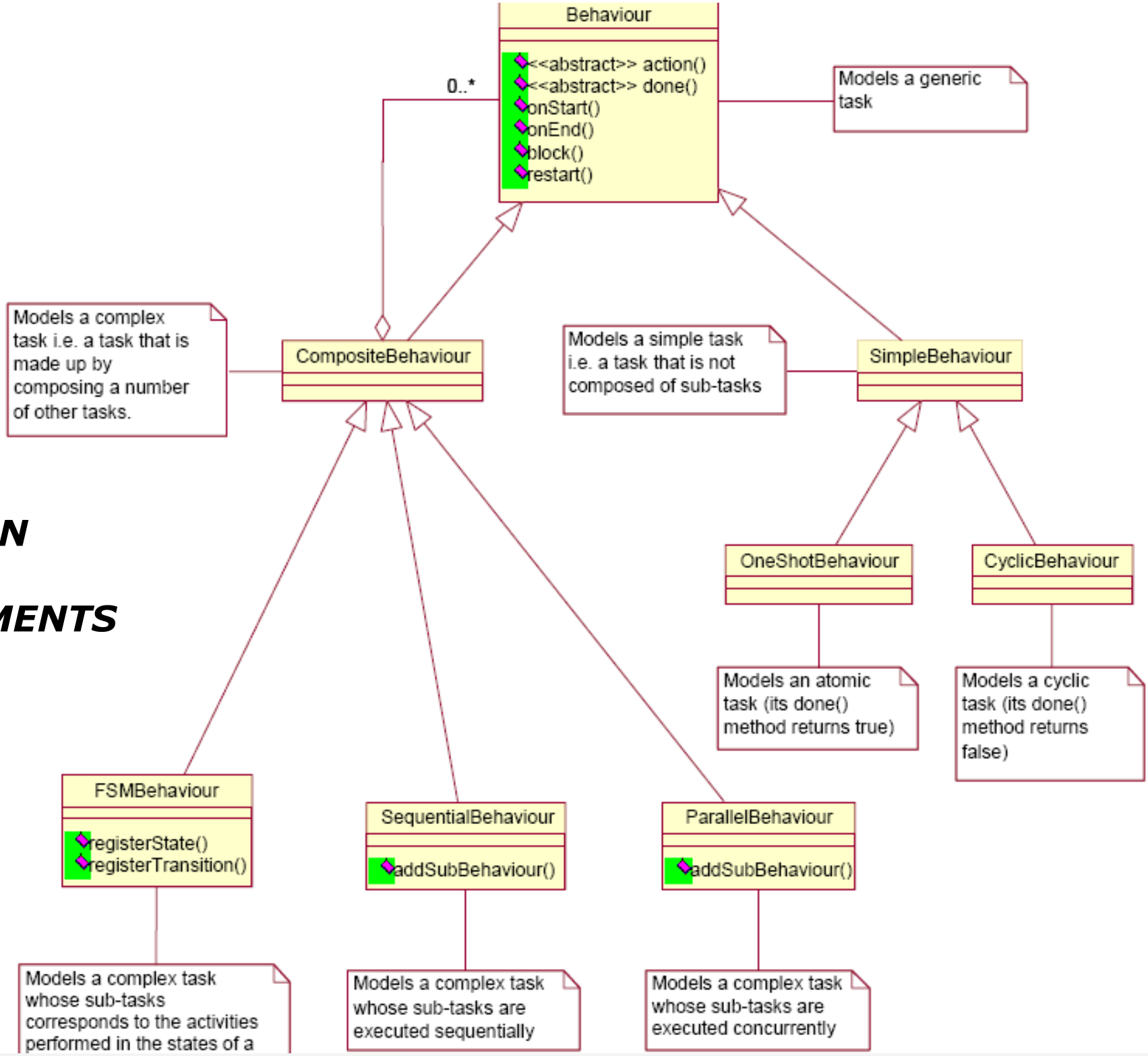


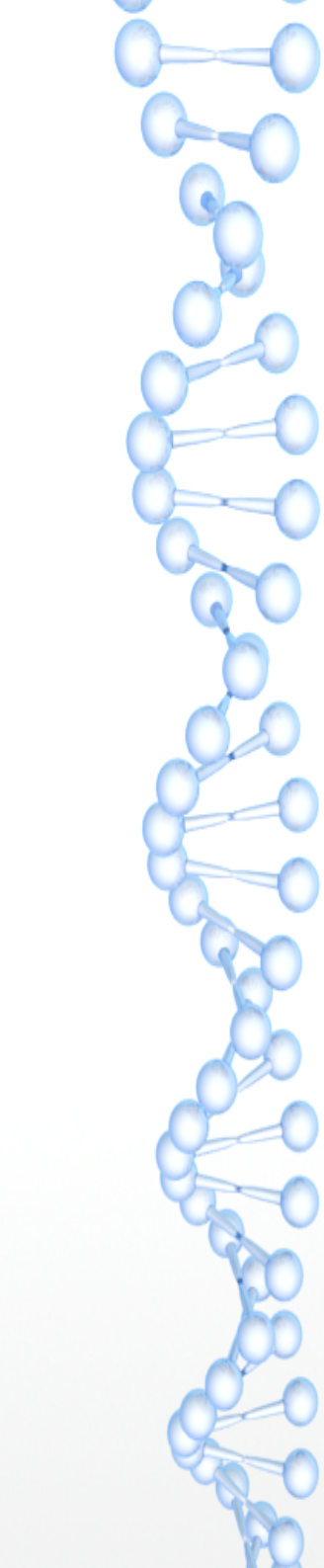
Agent

Cycle de vie



GESTION des COMPORTEMENTS





Simulation

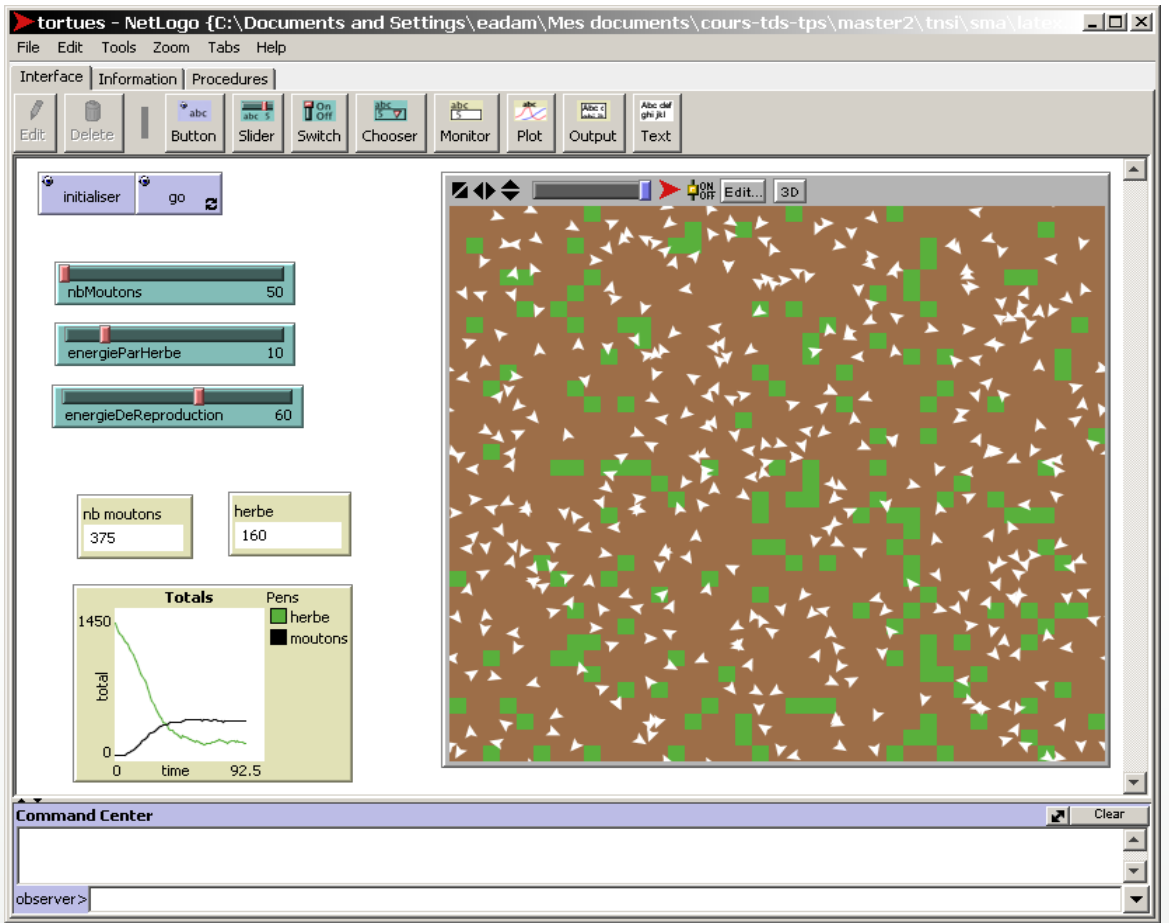
- Plateformes Netlogo et variantes langagières
- Plateformes CORMAS (agents situés), GAMA...

Netlogo

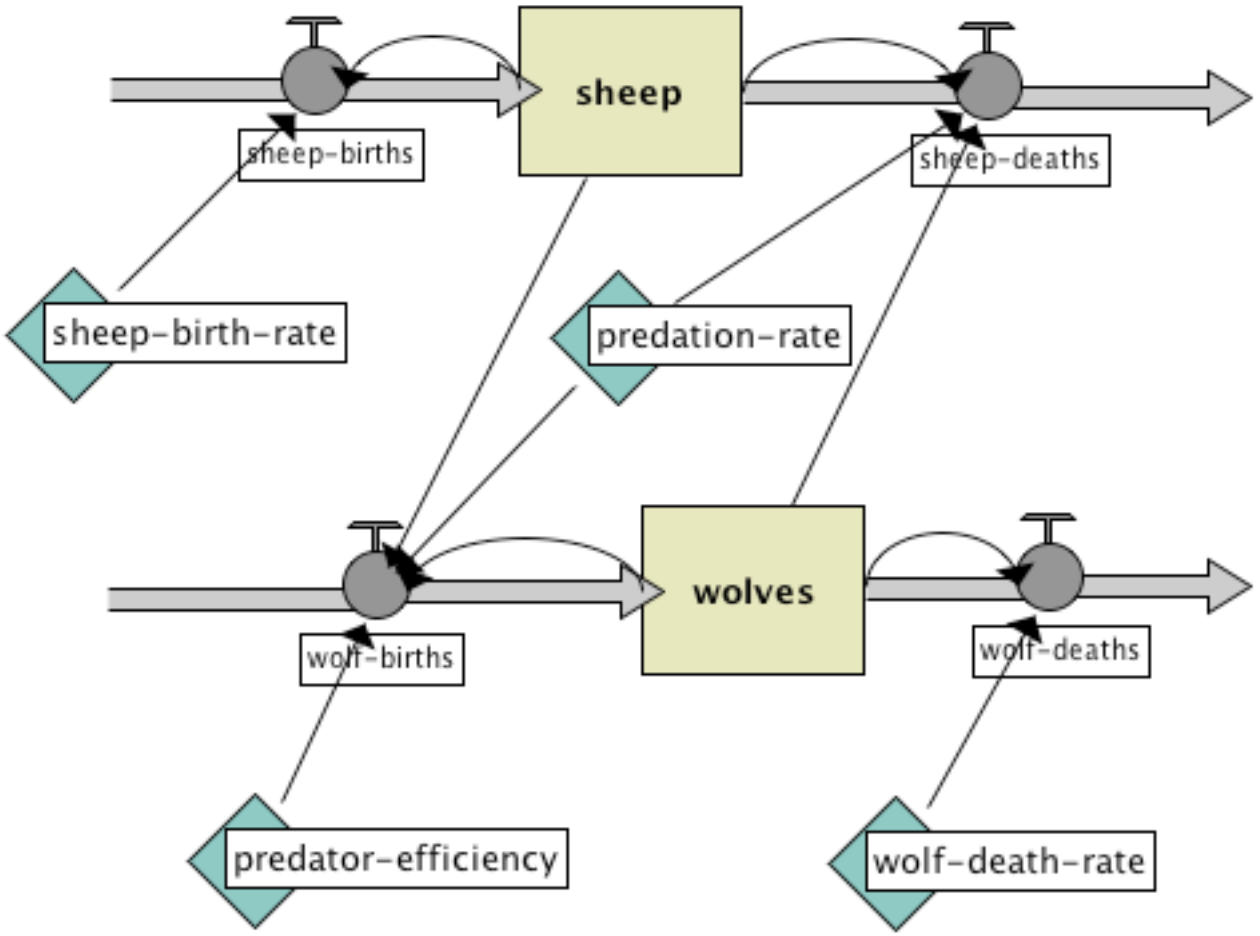
- Utilisation d'une plateforme multi-agent réactifs : NetLogo
- Utilisation de types cellules (patches), agents (turtles) et grille.
- Possibilité d'utiliser des indicateurs dédiés aux états des cellules,
- agents (compteurs, graphiques)
- Cas particulier de NetLogo : programmation "à la Logo" et affichage
- 3D possible. . .

Cas de l'éco-système

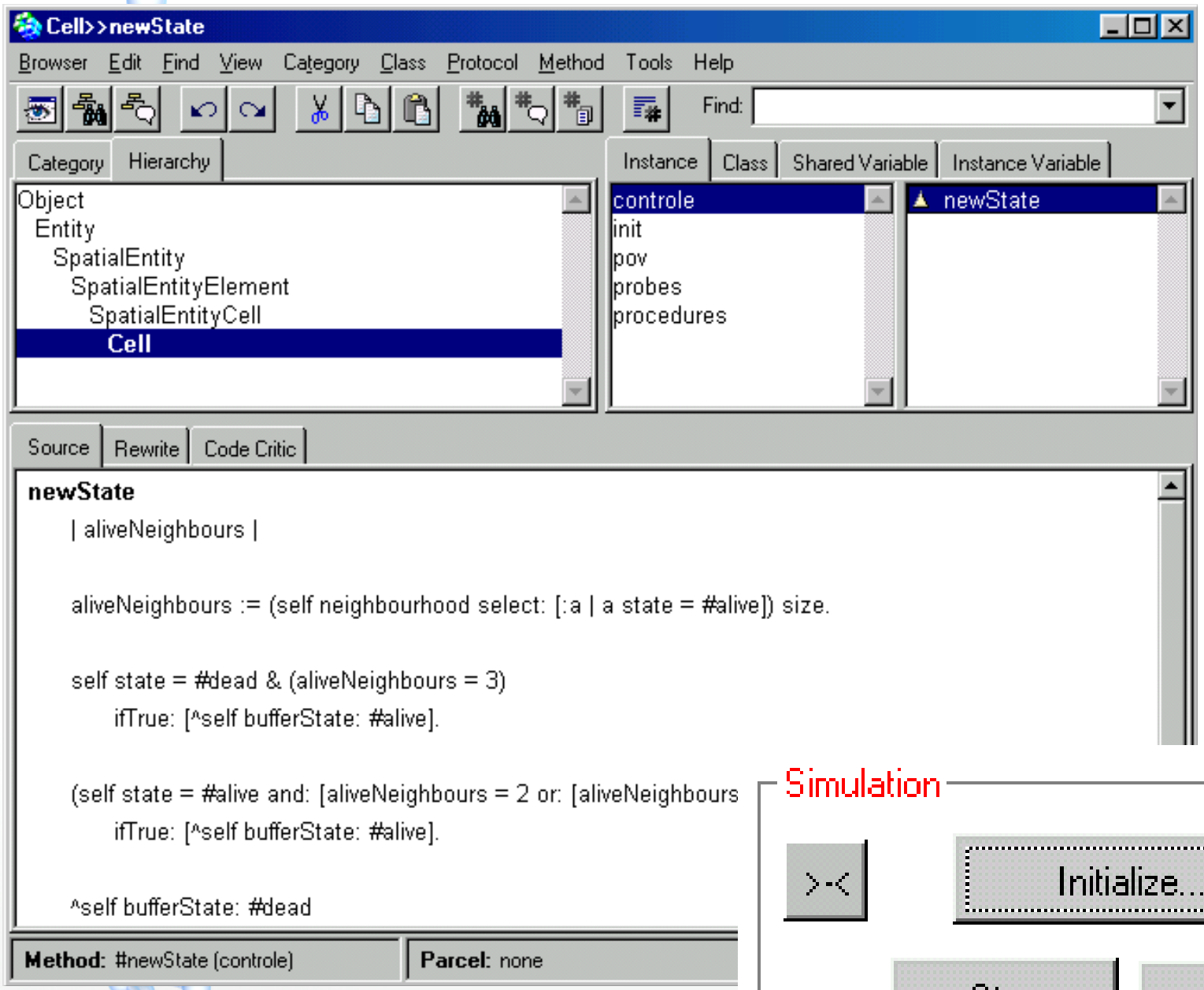
- Décomposition du problème: des agents représentent les moutons et des cellules représentent l'herbe.

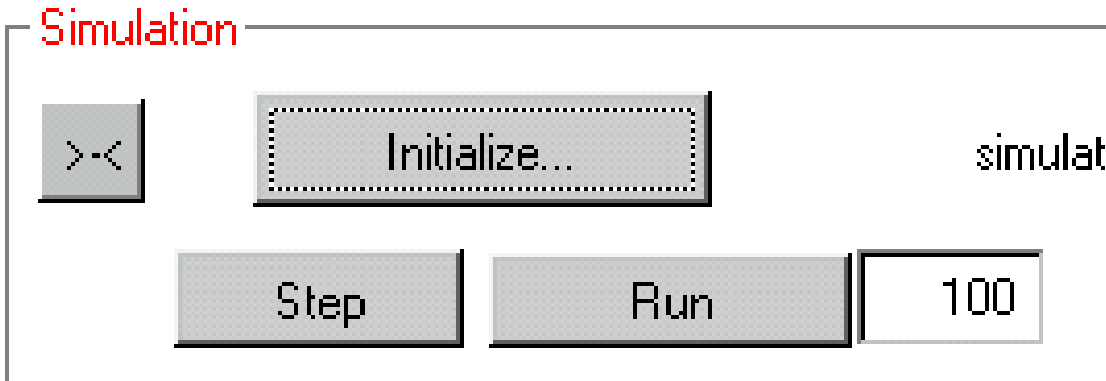


Cas de l'éco-système

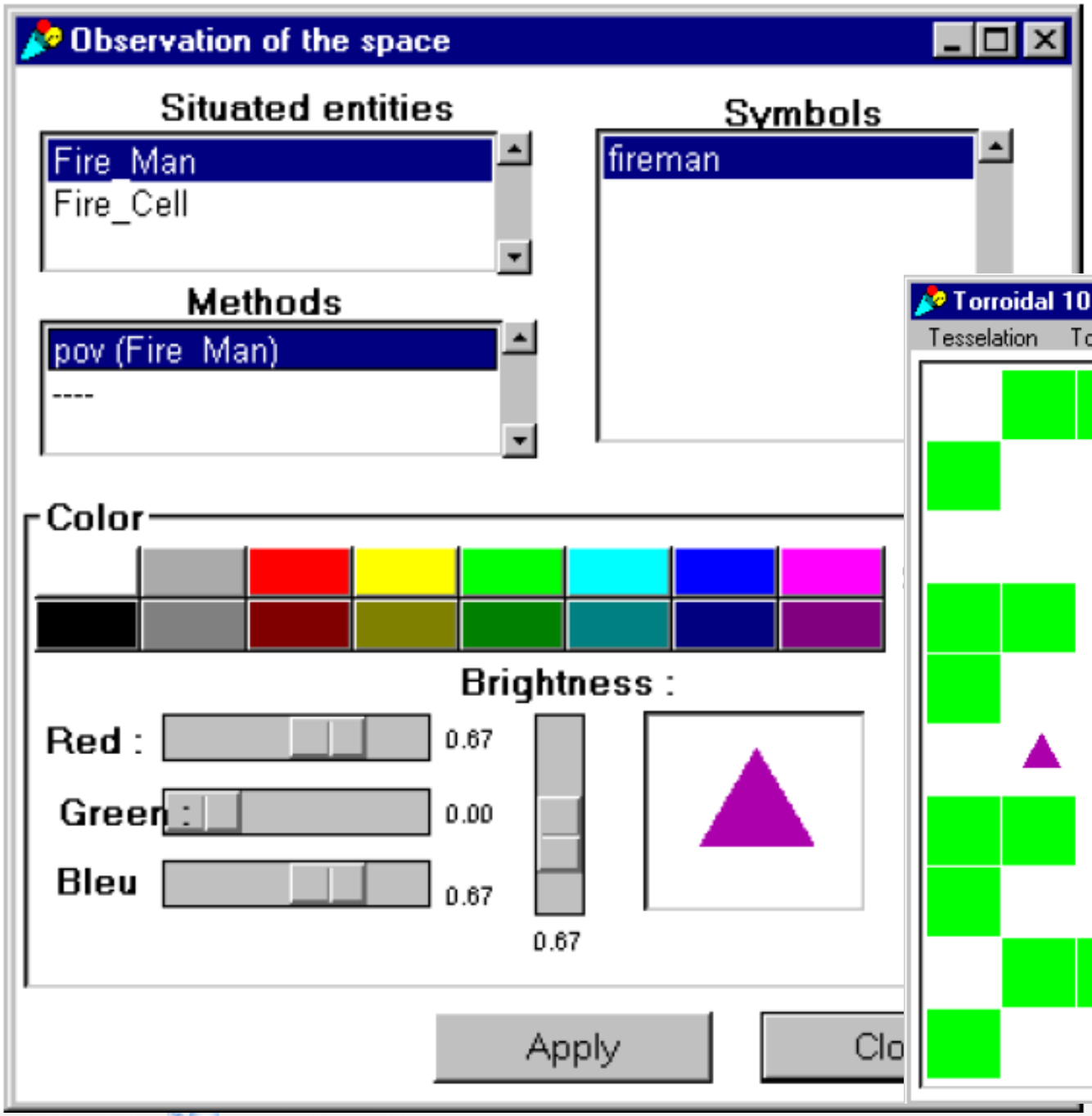


Cormas

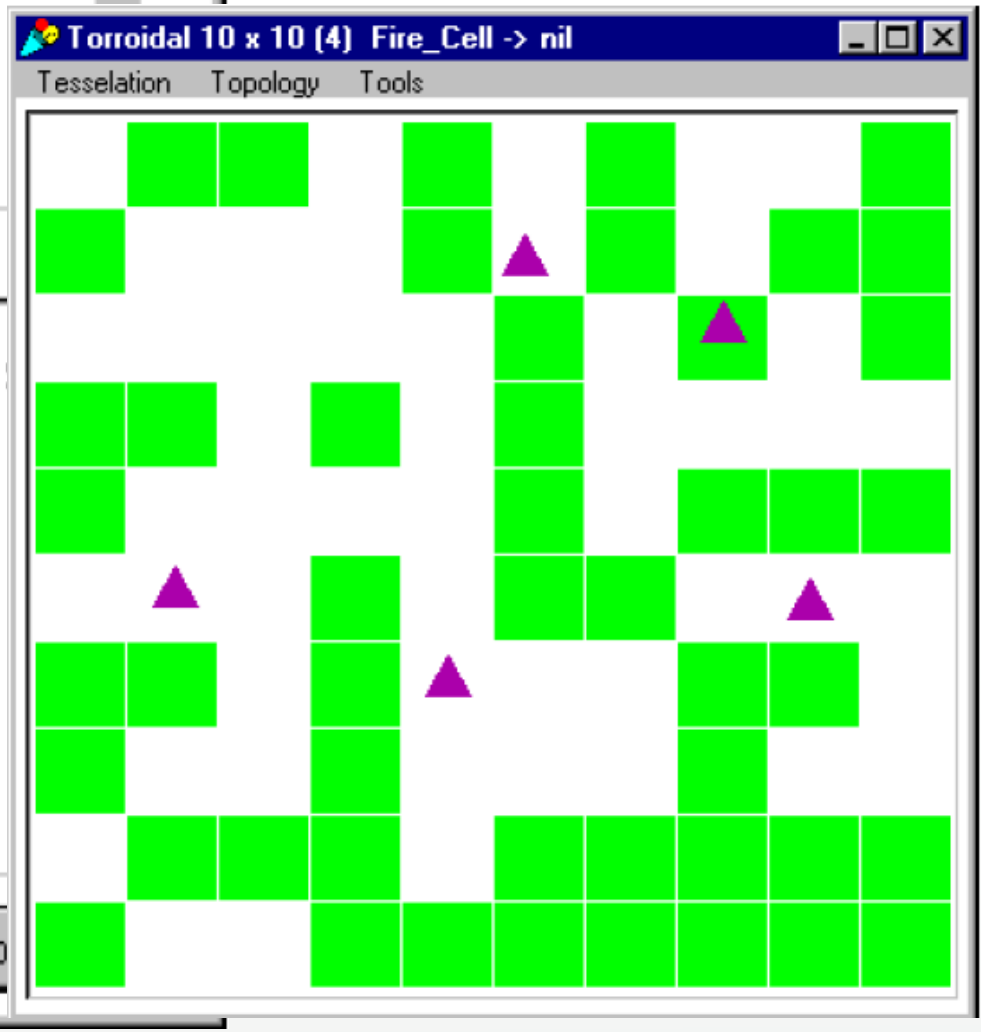

Cell>>newState
Browser Edit Find View Category Class Protocol Method Tools Help
Find:
Category Hierarchy
Object
Entity
SpatialEntity
SpatialEntityElement
SpatialEntityCell
Cell
Instance Class Shared Variable Instance Variable
controle
init
pov
probes
procedures
newState
Source Rewrite Code Critic
newState
| aliveNeighbours |
aliveNeighbours := (self neighbourhood select: [:a | a state = #alive]) size.
self state = #dead & (aliveNeighbours = 3)
ifTrue: [^self bufferState: #alive].
(self state = #alive and: [aliveNeighbours = 2 or: [aliveNeighbours
ifTrue: [^self bufferState: #alive].
^self bufferState: #dead
Method: #newState (controle)
Parcel: none


Simulation
>-<
Initialize...
simulat
Step
Run
100

Cormas


Observation of the space
Situated entities
Fire_Man
Fire_Cell
Symbols
fireman
Methods
pov (Fire_Man)

Color
Brightness :
Red :
0.67
Green :
0.00
Bleu
0.67
0.67
Apply
Clo


Torroidal 10 x 10 (4) Fire_Cell -> nil
Tesselation
Topology
Tools

Exercice

- Dans cet exemple, des agents vendent des livres et d'autres achetent des livres. . .
- Chaque agent acheteur recoit le titre du livre à acheter (le livre cible) en argument et invite periodiquement tous les agents vendeur connus a fournir une ore.
- Des qu'une offre est recue, l'agent acheteur l'accepte et publie un ordre d'achat.
- Si plus d'un agent vendeur fournit une offre, l'agent acheteur accepte la meilleure (le plus bas prix).
- L'agent acheteur stoppe apres avoir achete le livre cible.
- Chaque agent vendeur a une GUI minimale permettant a l'utilisateur d'inserer de nouveaux titres (et les prix associes) dans le catalogue local des livres a vendre.
- Les agents vendeur attendent sans interruption des demandes des agents acheteur.
- Lorsque les agents acheteur sont invités à fournir une offre pour un livre, ils verifient si le livre demandé est dans leur catalogue et repondent avec le prix.

  Autrement ils refusent.
- Quand ils recoivent un ordre d'achat, ils l'exécutent et enlèvent le livre demandé de leur liste