Les outils de simulation de la plate-forme MadKit

Fabien MICHEL

LIRMM / SMILE

Master 2 - Moteurs de simulation

Plan

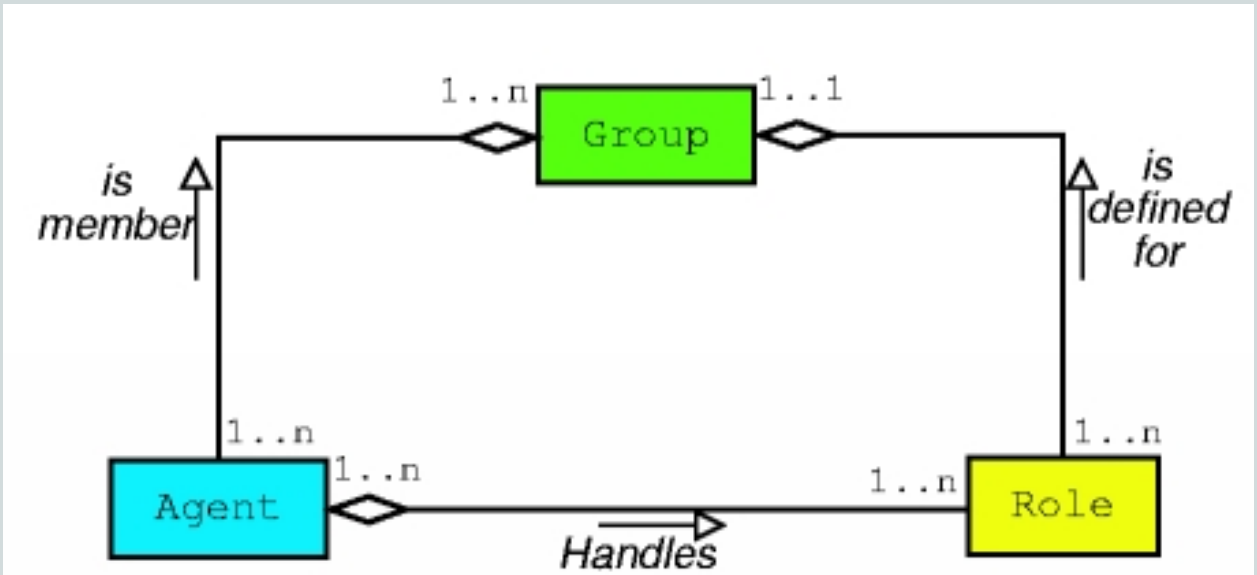
Plan

MadKit ▸ www.madkit.org

Caractéristiques

- Plate-forme pour le développement/déploiement de SMA
- Micro-noyau écrit en Java (distribuable P2P)
- Basée sur le modèle AGR :

Le modèle Agent/Groupe/Rôle

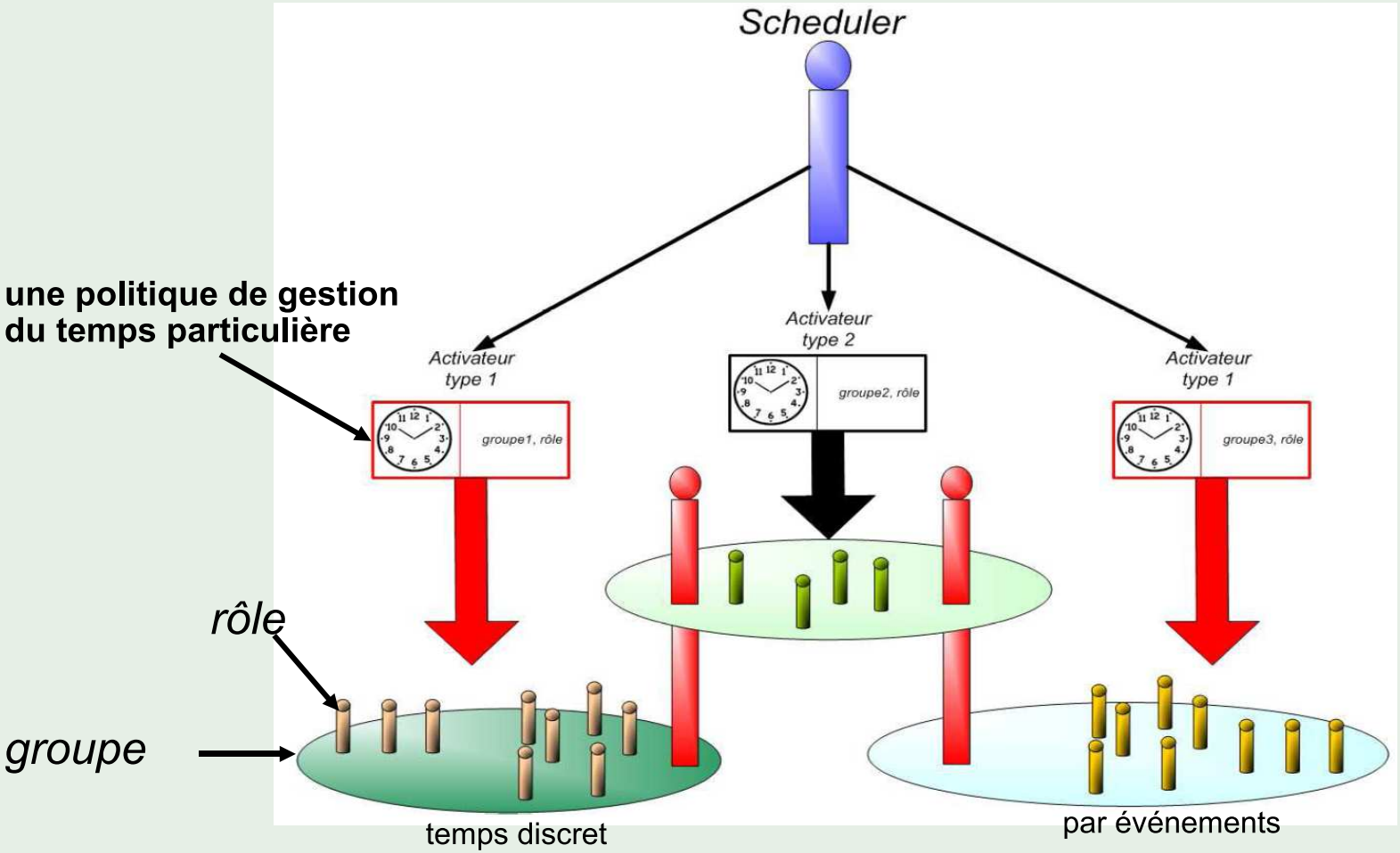


Plan

Les outils de simulation de la plate-forme MadKit

intégration dans MadKit d'un noyau de gestion du temps [Michel 04]

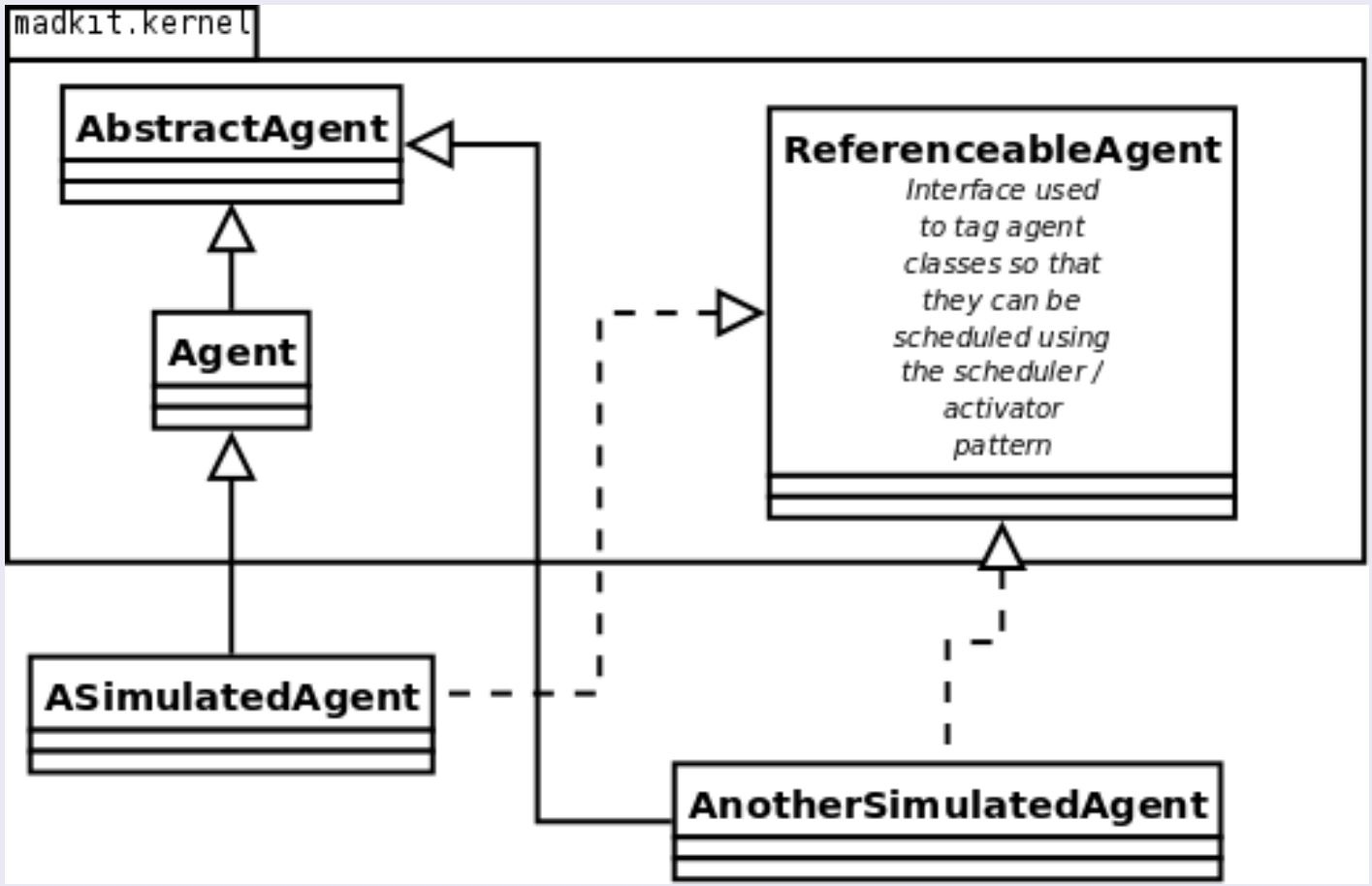
Agent/Groupe/Rôle pour la simulation



Plan

L'interface ReferenceableAgent du kernel de MadKit

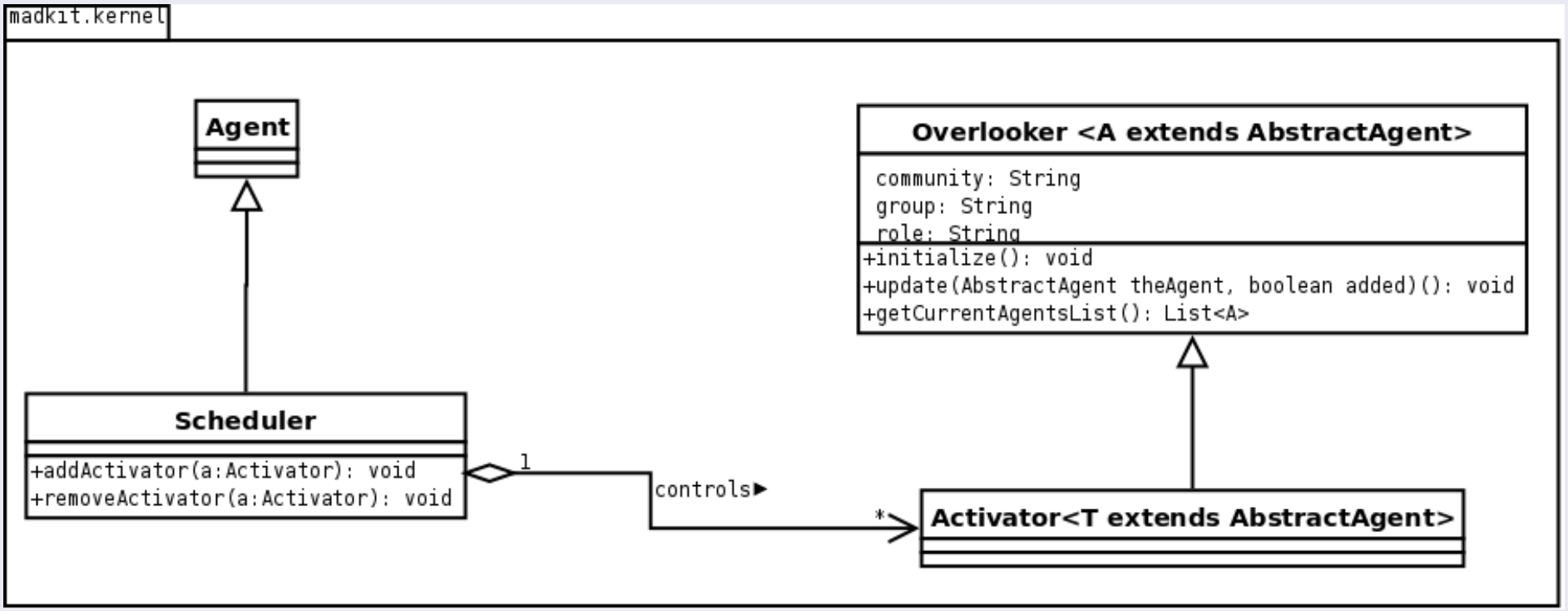
interface vide (sécurité + optimisation)



Plan

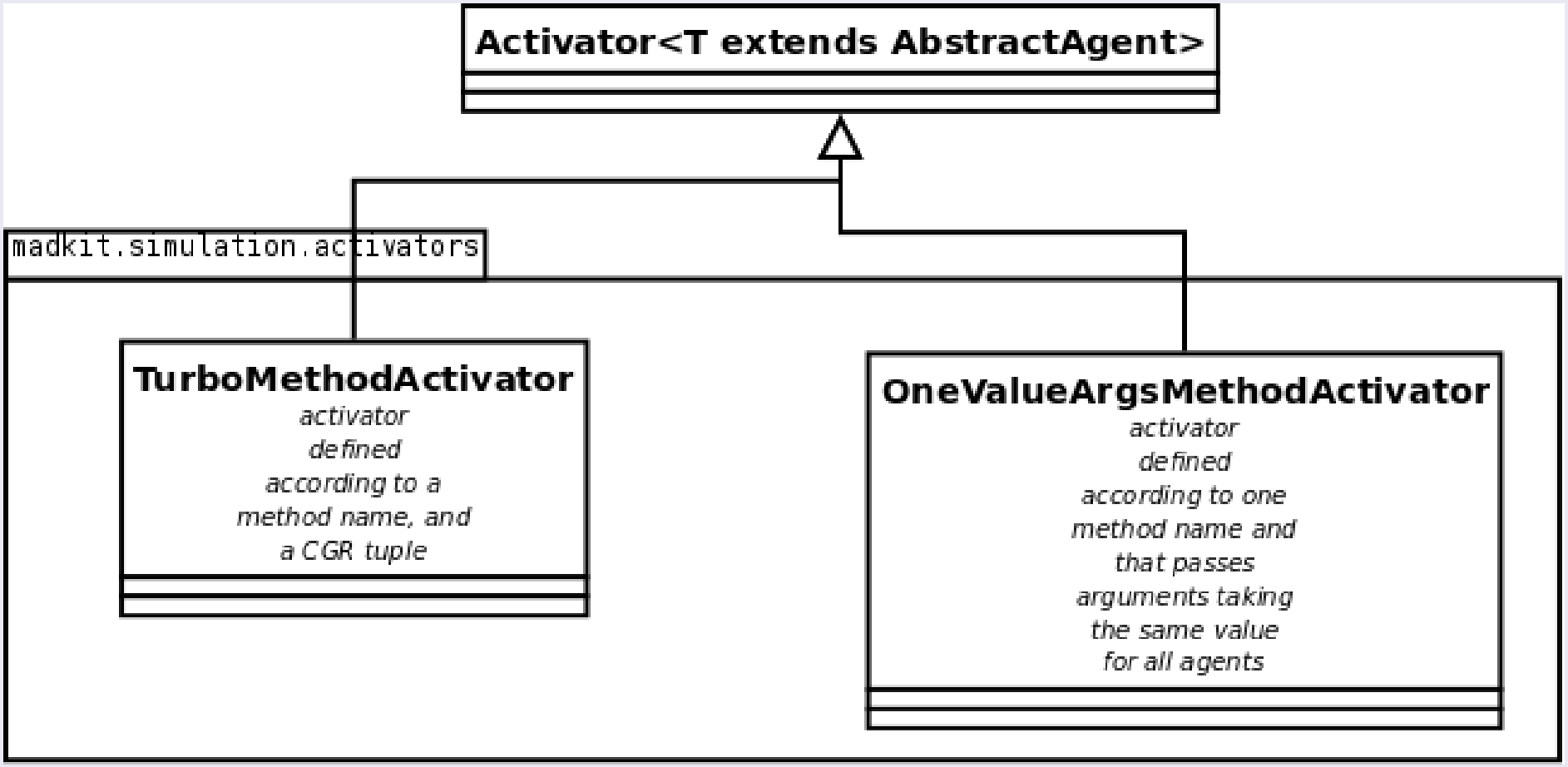
Le couple Scheduler / Activator

En UML



Le couple Scheduler / Activator

Les deux activators du package madkit.simulation.activators



Le couple Scheduler / Activator : En pratique

Soit un agent simulé très simple :

MyAgent.java

```
package toySimu;

import madkit.kernel.AbstractAgent;
import madkit.kernel.ReferenceableAgent;

public class MyAgent extends AbstractAgent implements ReferenceableAgent {

  @Override
  public void activate() {
    requestRole("testing", "simulation", "agent", null);
  }

  public void doIt() {
    println("activated");
  }
}
```

Le couple Scheduler / Activator : En pratique

1 : avoir au moins un activator

MyScheduler.java

```
package toySimu;

import madkit.kernel.Scheduler;
import madkit.simulation.activators.TurboMethodActivator;

public class MyScheduler extends Scheduler {

  private TurboMethodActivator tma;
```

Le couple Scheduler / Activator : En pratique

2 : un agent scheduler est aussi un agent MadKit comme un autre.
Utilisation d'activate pour l'initialisation :

MyScheduler.java

```
@Override
public void activate() {
  createGroup(false, "testing", "simulation", null, null);
  requestRole("testing", "simulation", "scheduler", null);
  tma = new TurboMethodActivator("doIt","testing", "simulation","agent");
  addActivator(tma);

  //launching simulated agents
  for (int i = 0; i < 100000 ; i++) {
    launchAgent(new MyAgent(), ""+i, false);
  }
}
```

Le couple Scheduler / Activator : En pratique

3 : la boucle de simulation (dans la méthode live)

MyScheduler.java

```
@Override
public void live() {
  while (true) {
    pause(100);
    exitImmediatlyOnKill();
    tma.execute();
  }
}
```

Le couple Scheduler / Activator : En pratique

Soit un deuxième type d'agent simulé de classe différente :

MyAgent2.java

```
package toySimu;

import madkit.kernel.AbstractAgent;
import madkit.kernel.ReferenceableAgent;

public class MyAgent2 extends AbstractAgent implements ReferenceableAgent {

  @Override
  public void activate() {
    requestRole("testing", "simulation", "agent", null);
  }

  public void doIt() {
    println("agent type2 activated");
  }
}
```

Le couple Scheduler / Activator : En pratique

Pas besoin de changer quoi que ce soit dans le scheduler : la classe de l'agent n'a pas dimportance pour un activator, seule la position dans l'organisation compte :

MyScheduler.java

```
@Override
public void activate() {
  createGroup(false, "testing", "simulation", null, null);
  requestRole("testing", "simulation", "scheduler", null);
  tma = new TurboMethodActivator("doIt","testing", "simulation","agent");
  addActivator(tma);

  //launching simulated agents
  for (int i = 0; i < 100000 ; i++) {
    launchAgent(new MyAgent(), ""+i, false);
    launchAgent(new MyAgent2(), ""+i, false);
  }
}
```

On peut donc écrire des choses comme ça :

```
public class MyAgent extends AbstractAgent implements ReferenceableAgent {

  String type;

  @Override
  public void activate() {
    if (Math.random()<.5) {
      requestRole("testing", "simulation", "agent", null);
      type = "agent";
    }
    else {
      requestRole("testing", "simulation", "ant", null);
      type = "ant";
    }
  }

  public void doIt() {
    println(type+" activated");
  }
}
```

Le couple Scheduler / Activator : En pratique

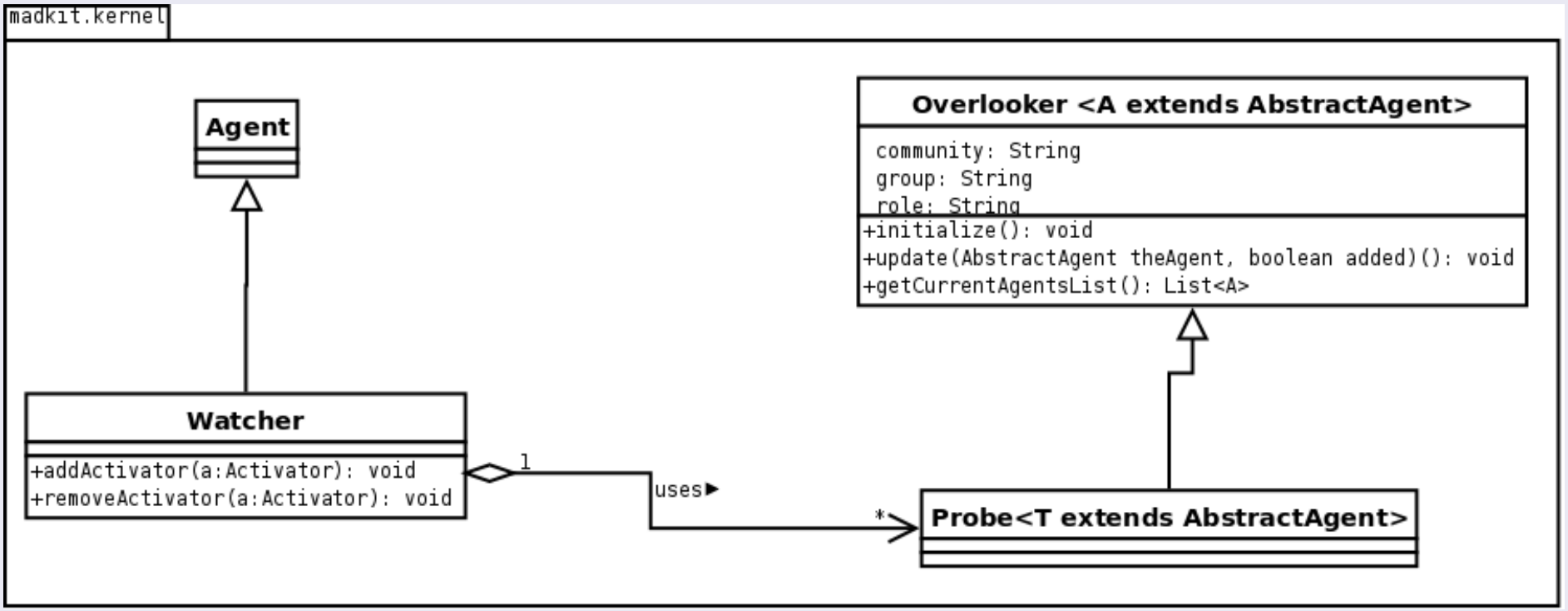
MyScheduler.java version 2

```
public class MyScheduler extends Scheduler {

  private TurboMethodActivator tmaForAgent;
  private TurboMethodActivator tmaForAnt;

  @Override
  public void activate() {
    createGroup(false, "testing", "simulation", null, null);
    requestRole("testing", "simulation", "scheduler", null);
    tmaForAgent = new TurboMethodActivator("doIt","testing",
      "simulation","agent");
    addActivator(tmaForAgent);
    tmaForAnt = new TurboMethodActivator("doIt","testing",
      "simulation","ant");
    addActivator(tmaForAnt);
    ...
```

Plan

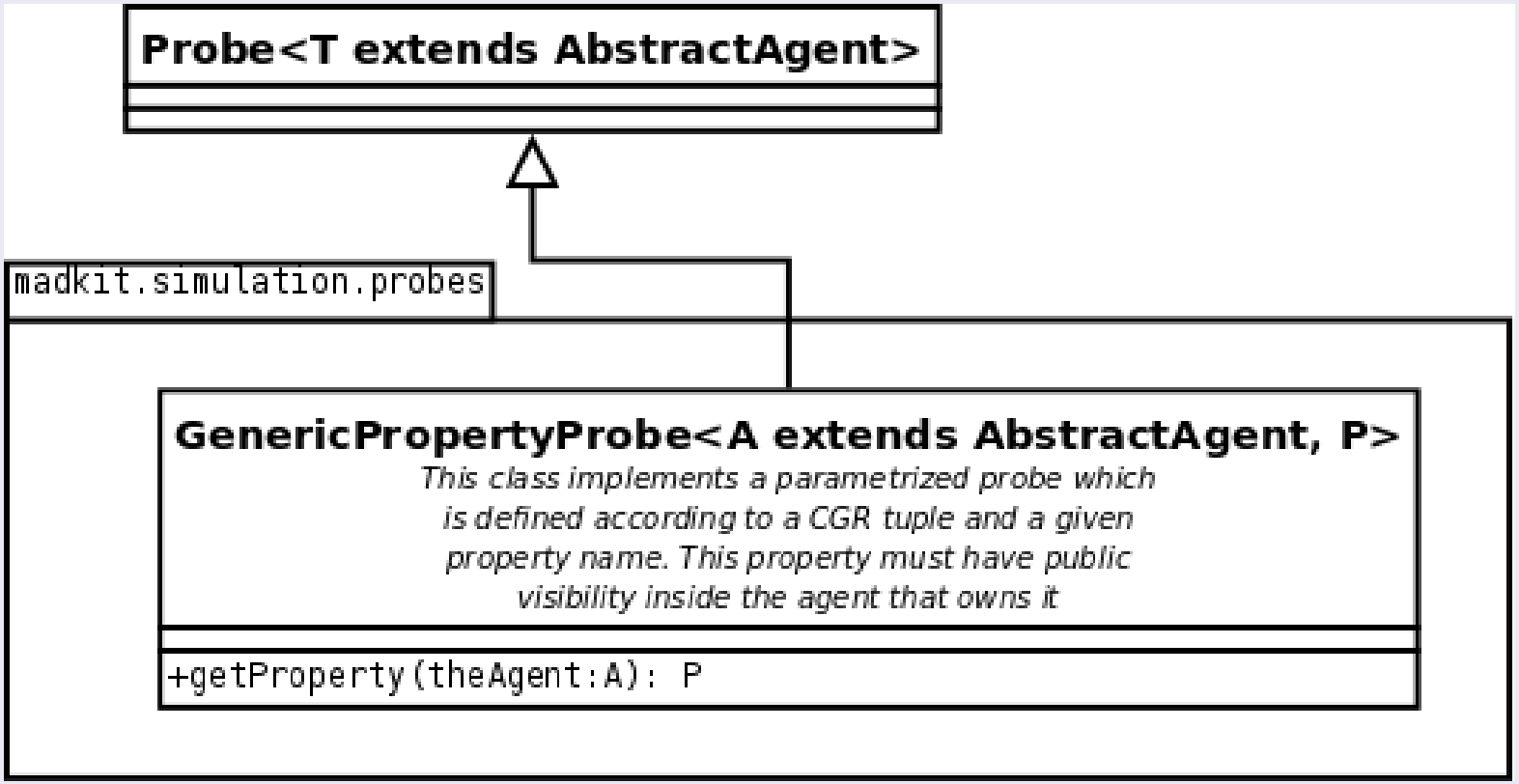
Le couple Watcher / Probe

En UML



Le couple Watcher / Probe

Une probe du package madkit.simulation.probes



Le couple Watcher / Probe : En pratique

Soit notre agent simulé précédent :

MyAgent.java

```
public class MyAgent extends AbstractAgent implements ReferenceableAgent {

  String type;
  public double x;

  . . .

  public void doIt() {
    println(type+" activated");
    x = Math.random()*20;
  }
}
```

Le couple Watcher / Probe : En pratique

1 : avoir au moins défini une probe

MyWatcher.java

```
public class MyWatcher extends Watcher implements ReferenceableAgent{

  private GenericPropertyProbe<AbstractAgent,Double> probe;

  ...
```

Le couple Watcher / Probe : En pratique

2 : un agent Watcher est aussi un agent MadKit comme un autre.
Utilisation d'activate pour l'initialisation :

MyWatcher.java

```
@Override
public void activate() {
  requestRole("testing", "simulation","observer",null);
  probe = new GenericPropertyProbe<AbstractAgent,Double>("testing",
      "simulation","agent","x");
  addProbe(probe);
}
```

Le couple Watcher / Probe : En pratique

3 : Utiliser la probe pour récupérer la propriété souhaitée

MyWatcher.java

```
public void observe(){
  for (AbstractAgent agent : probe.getCurrentAgentsList()) {
    System.out.println("x = "+probe.getProperty(agent));
  }
}
```

Le couple Watcher / Probe : En pratique

Soit un deuxième type d'agent simulé de classe différente :

MyAgent2.java

```
public class MyAgent2 extends AbstractAgent implements ReferenceableAgent {

  public double x;

  @Override
  public void activate() {
    requestRole("testing", "simulation", "agent", null);
  }

  public void doIt() {
    println("my agent 2 activated");
  }
}
```

Plan

Les agents simulés

MyAgent.java

```
public class MyAgent extends AbstractAgent implements ReferenceableAgent {
  public int x=200,y=200;

  @Override
  public void activate() {
    if (Math.random()<.5) {
      requestRole("testing", "simulation", "agent", null);
    }
    else {
      requestRole("testing", "simulation", "ant", null);
    }
  }

  public void doIt() {
    if(Math.random()<.5)
      x+=Math.random()*5;
    else
      x-=Math.random()*3;
    if(Math.random()<.5)
      y+=Math.random()*5;
    else
      y-=Math.random()*3;
  }
}
```

L'agent chargé de l'affichage : un Watcher

MyViewer.java

```
public class MyViewer extends Watcher implements ReferenceableAgent{

  private GenericPropertyProbe<AbstractAgent,Integer> probeX,probeY;
  private GridCanvas sheet;

  @Override
  public void activate() {
    requestRole("testing", "simulation","observer",null);
    probeX = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","agent","x");
    probeY = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","agent","y");
    addProbe(probeX);
    addProbe(probeY);
  }


  . . .
```

L'agent chargé de l'affichage : un Watcher

MyViewer.java

```
public void initGUI()
{
  setGUIObject(sheet = new GridCanvas(400,400,this));
}

public void observe(){
  sheet.repaint();
}

public void paintSimu(Graphics g) {
  for (AbstractAgent agent : probeX.getCurrentAgentsList()) {
    int x = probeX.getProperty(agent);
    int y = probeY.getProperty(agent);
    g.fill3DRect(x, y, 10, 10, false);
  }
}
}
```

L'agent chargé de l'affichage : un Watcher

MyViewer.java

```
class GridCanvas extends JPanel
{
private MyViewer viewer;

public GridCanvas(int width,int height,MyViewer l)
{
  setBackground(Color.green);
  setSize(new Dimension(width,height));
  viewer=l;
}
public Dimension getPreferredSize() {return getSize();}

public void paintComponent(Graphics g)
{
  super.paintComponent(g);
  viewer.paintSimu(g);
}
}
```

Le Scheculer

MyScheduler.java

```
public void activate() {
  createGroup(false, "testing", "simulation", null, null);
  requestRole("testing", "simulation", "scheduler", null);
  tmaForAgent = new TurboMethodActivator("doIt","testing", "simulation","agent");
  addActivator(tmaForAgent);
  tmaForAnt = new TurboMethodActivator("doIt","testing", "simulation","ant");
  addActivator(tmaForAnt);
  tmaForObs = new TurboMethodActivator("observe","testing", "simulation","observer");
  addActivator(tmaForObs);

  //launching simulated agents
  for (int i = 0; i < 50 ; i++) {
    launchAgent(new MyAgent(), ""+i, false);
  //  launchAgent(new MyAgent2(), ""+i, false);
  }
  launchAgent(new MyWatcher(), "obs", false); launchAgent(new MyViewer(),
  "viewer", true); }

@Override
public void live() {
  while (true) {
    pause(100);   exitImmediatlyOnKill();
    tmaForAgent.execute();
    tmaForAnt.execute();
    tmaForObs.execute();  } }
```

Un peu plus compliqué

MyViewer.java

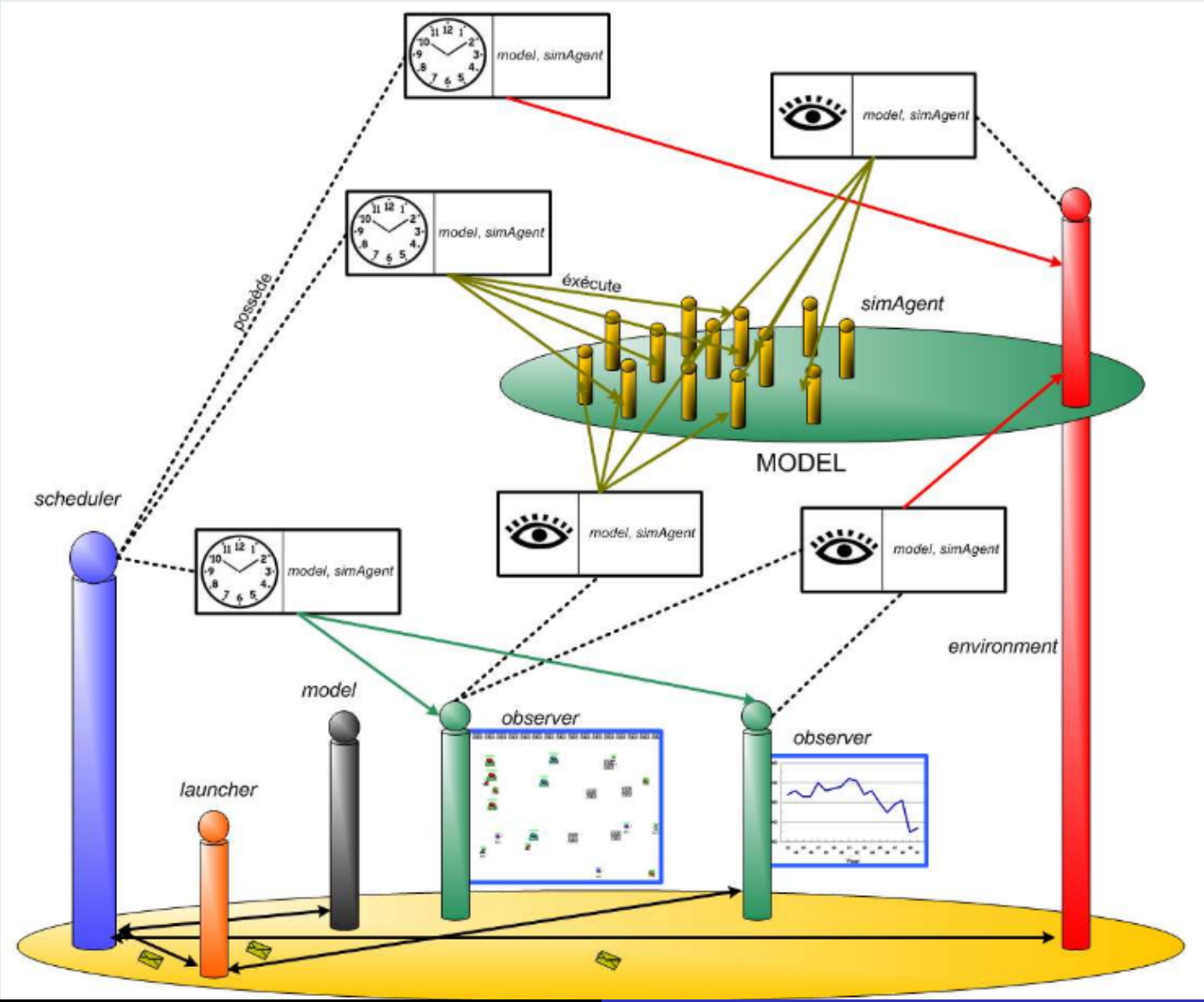
```
public class MyViewer extends Watcher implements ReferenceableAgent{

  private GenericPropertyProbe<AbstractAgent,Integer> probeX,probeY,antX,antY;
  private GridCanvas sheet;

  @Override
  public void activate() {
    requestRole("testing", "simulation","observer",null);
    probeX = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","agent","x");
    probeY = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","agent","y");
    addProbe(probeX);    addProbe(probeY);
    antX = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","ant","x");
    antY = new GenericPropertyProbe<AbstractAgent,Integer>("testing","simulation","ant","y");
    addProbe(antX);    addProbe(antY);
  }
```

Un peu plus compliqué

MyViewer.java

```
public void paintSimu(Graphics g) {
  g.setColor(Color.black);
  for (AbstractAgent agent : probeX.getCurrentAgentsList()) {
    int x = probeX.getProperty(agent);
    int y = probeY.getProperty(agent);
    g.fill3DRect(x, y, 10, 10, false);
  }
  g.setColor(Color.yellow);
  for (AbstractAgent agent : antX.getCurrentAgentsList()) {
    int x = antX.getProperty(agent);
    int y = antY.getProperty(agent);
    g.fill3DRect(x, y, 10, 10, true);
  }
}
}
```

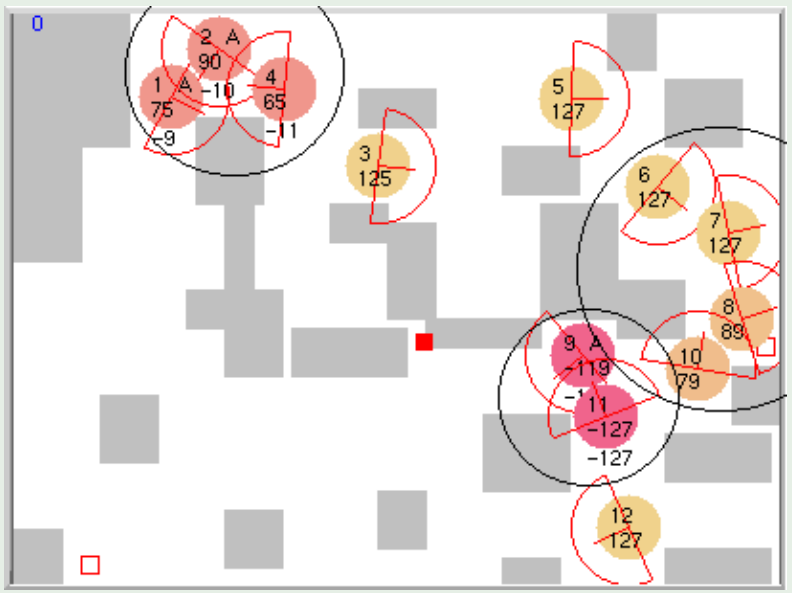
Patron "moteur de simulation multi-agent AGR"



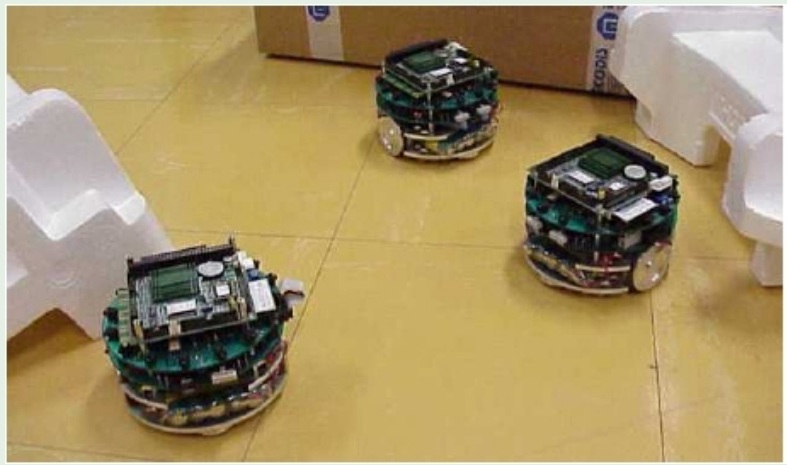
Thèses de doctorat : Simonin 2001, Chapelle 2006 (LIRMM)

Robotique mobile collective

De la simulation . . .



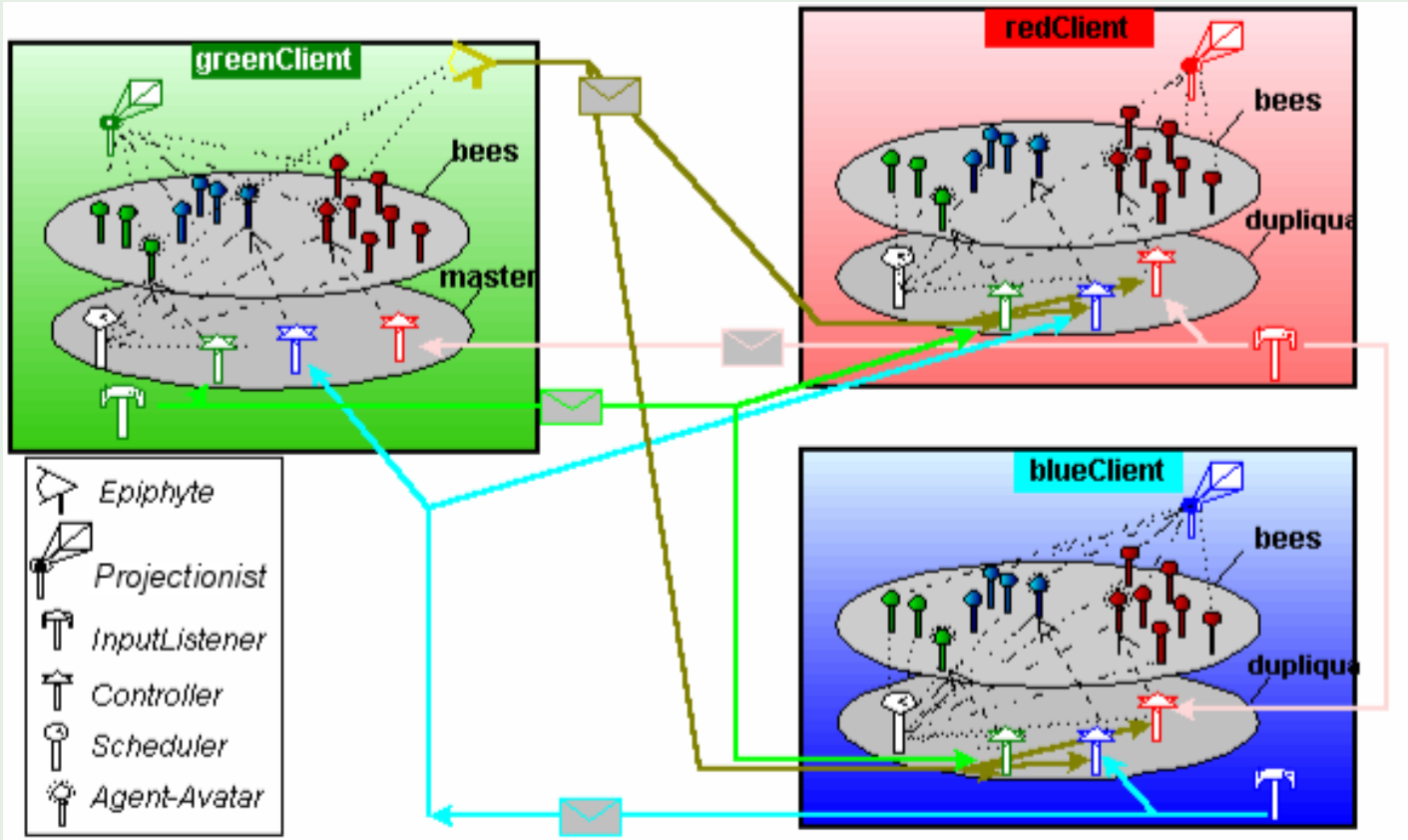
. . . À l'expérimentation



Simulation distribuée interactive

[Michel, Bommel, Ferber 02], Astronoid

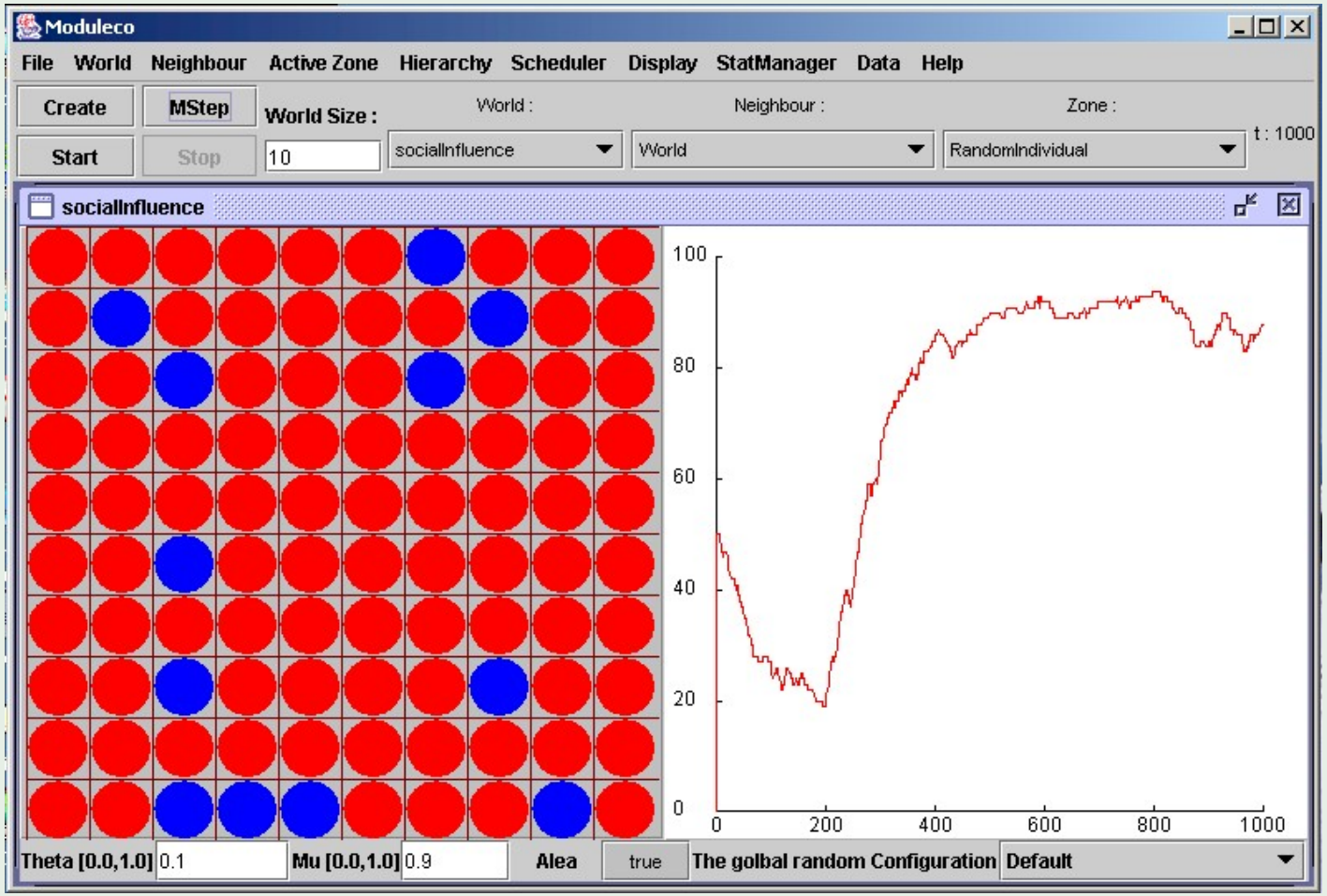
Simulation distribuée interactive



Intégration de la plate-forme Moduleco dans MadKit

Moduleco : simulation pour les sciences sociales [Michel et al. 05]

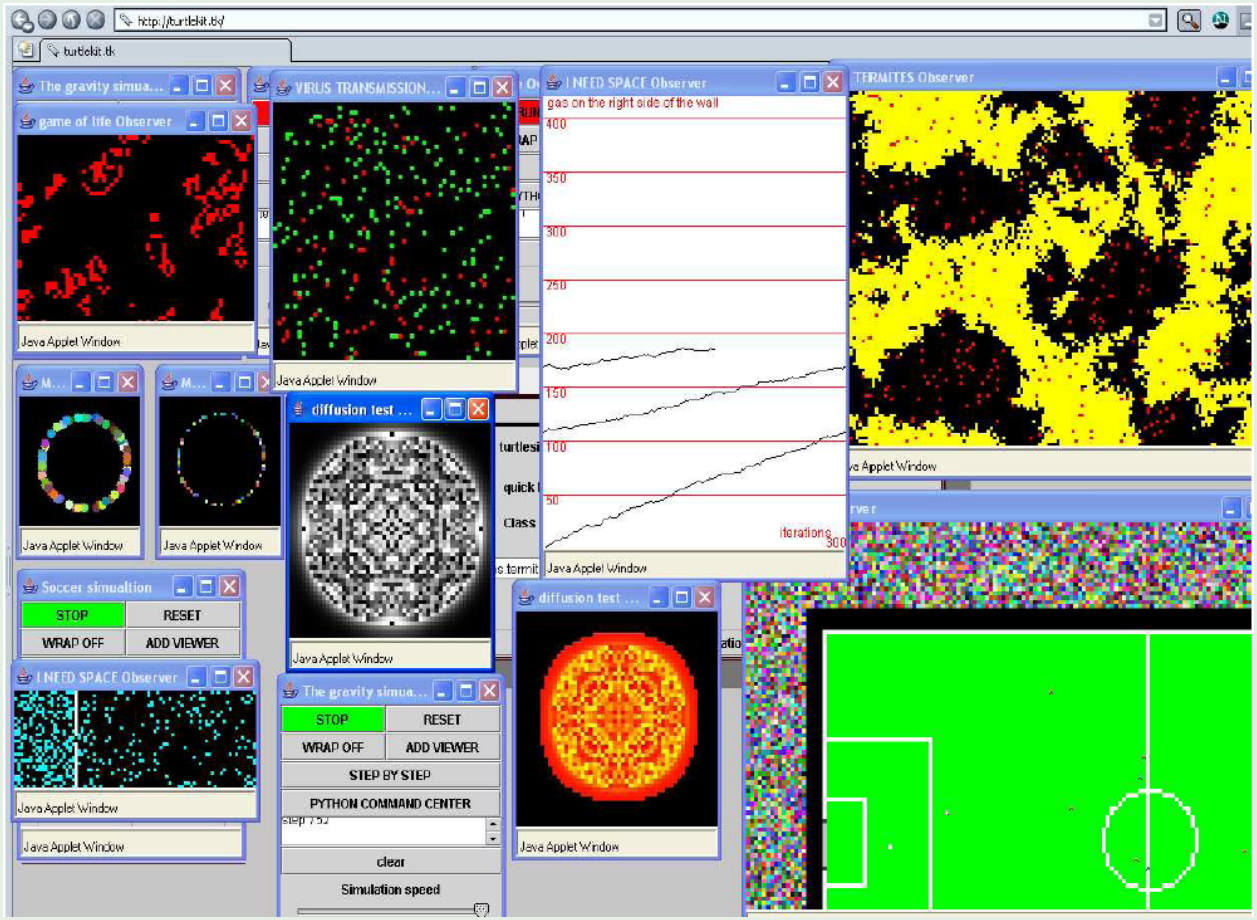
Moduleco



La plate-forme de simulation TurtleKit

[Michel 02, Michel et al. 05], LORIA (simonin)

TurtleKit



Beurier, thèse de doctorat 2007 (LIRMM)

Émergence multi-niveaux, codage indirect de la forme dans les SMA

Émergence multi-niveaux