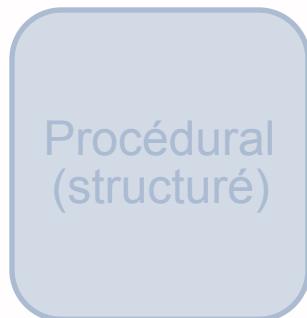


INTELLIGENCE COLLABORATIVE AVEC LES SYSTÈMES MULTI-AGENTS

S. HAMMADI, H. ZGAYA BIAU, Sarah BEN OTHMAN, Pascal YIM

SMA

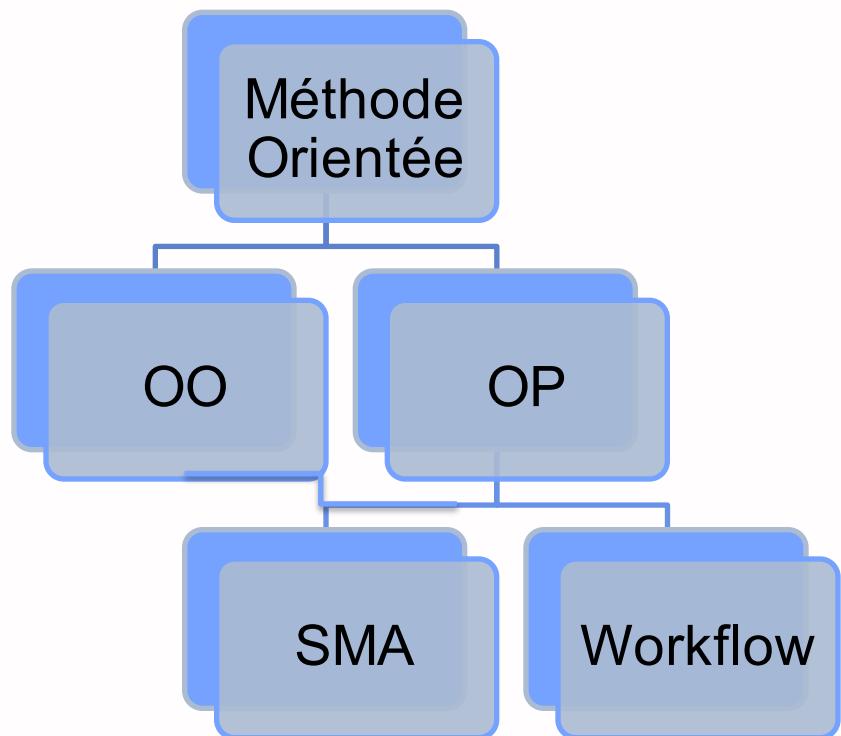
Approche de modélisation



- **Séparation** : données & traitements

- **Encapsulation** dans le même objet : données & traitements.
- **Utilisation** : Architecture logicielle
- **Standard** : Unified Modeling Language (UML)

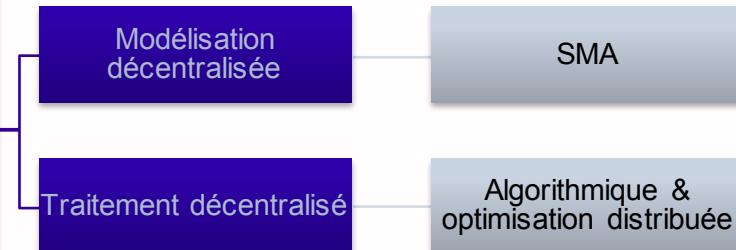
- **SMA** : représentation d'un système sous forme d'entités autonomes en interaction
- **Workflow** : représentation des processus (activités et gestion de flux)



Influences

Informatique

Représentation et traitement
distribués des données



Sciences Cognitives

Intelligence et autonomie des SI

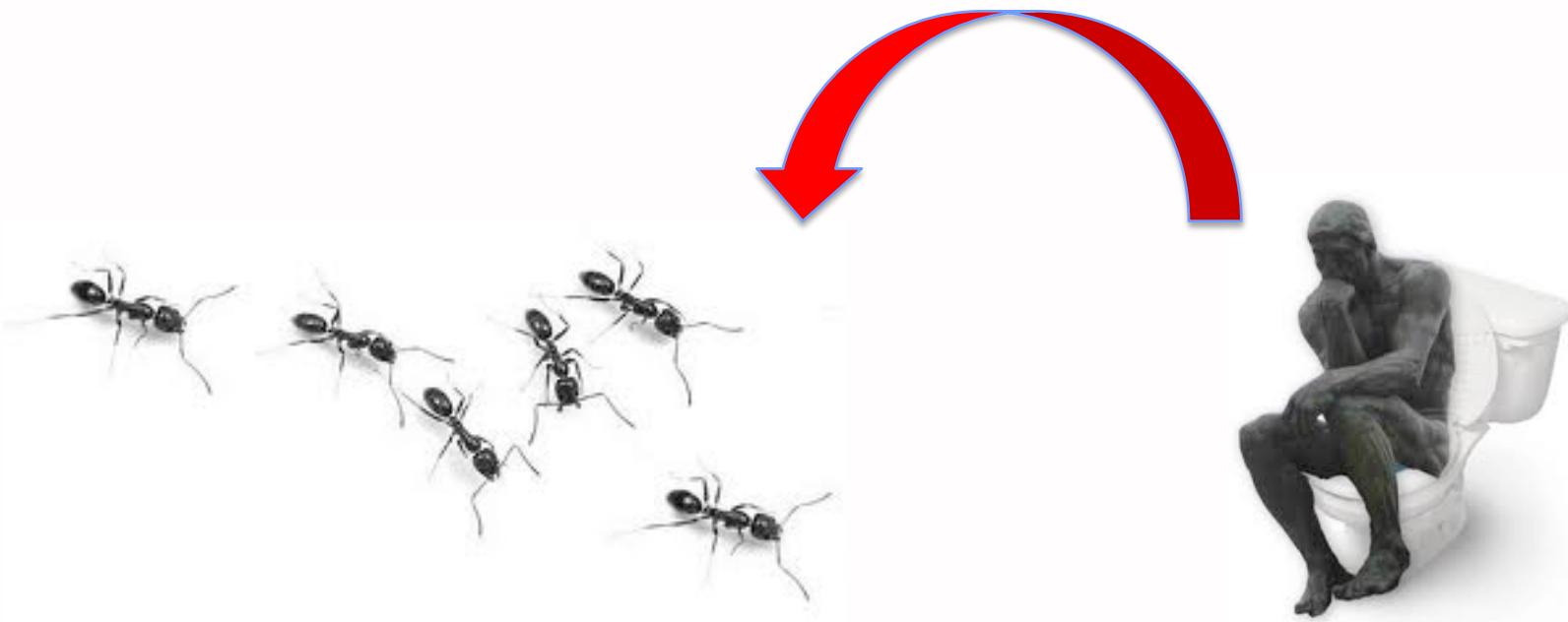
Vers l'Humain

Science de l'Homme
et de la société

Biologie & Physique

IAD : Intelligence Artificielle Distribuée
SMA : Système Muti-Agent

Influences



Communauté de penseurs
(SMA)

Penseur isolé
(agent)

SMA & IAD

Agent : Entité logicielle ou matérielle

Jeux vidéo

- Autonome,
- Situé dans un environnement (réel ou virtuel),
- Joue un ou plusieurs rôles dans une organisation,
- Motivé par ses tendances internes : buts, recherche de satisfaction...

- Capable de:
 - Communiquer avec d'autres agents,
 - Agir sur la conduite d'un utilisateur ou d'une autre entité,
 - Anticiper,
 - Apprendre de ses expériences et **s'adapte à son environnement**,
 - ...



Systèmes complexes

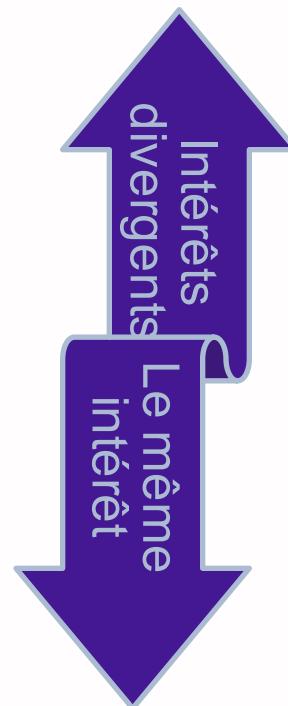
Robotique

Système Multi-Agent

- **Quoi?** Système composé d'agents qui interagissent
 - Entités ± autonomes et interactives
 - Représentation réaliste du monde
- **Pourquoi?**
 - Simuler les systèmes complexes
 - Résoudre les problèmes (SAD)
- **Comment?**
 - en coordonnant leurs actions pour éviter les conflits de ressources
 - d'une manière décentralisée.



les agents **négocient**
(agents antagonistes)



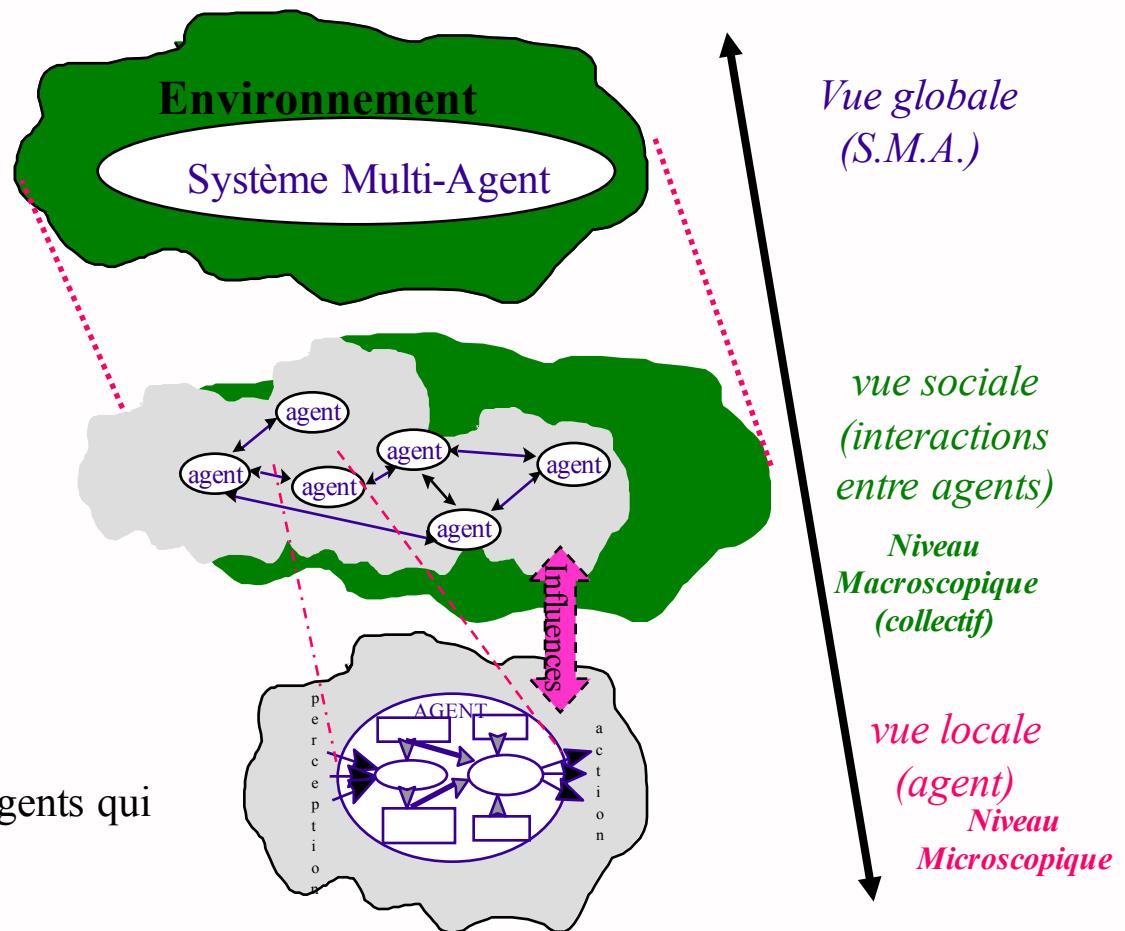
les agents
coopèrent/collaborent
(agents non antagonistes)

Conception d'Agent/SMA

→ définir leur environnement

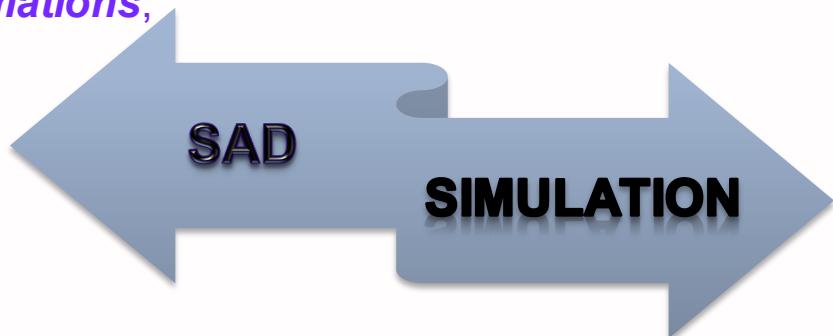
→ définir les organisations sociales et les interactions

→ définir le modèle de chacun des agents qui va entrer en action



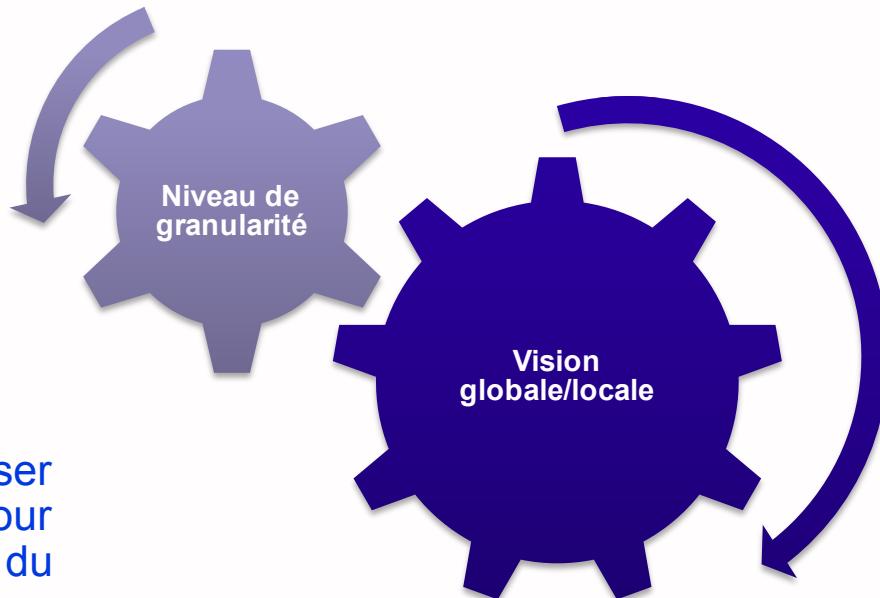
Quand utiliser un SMA? Pièges à éviter

Simplifier (a priori) la description d'un problème en distribuant les **tâches** et les **informations**;



Doter des agents de et les laisser évoluer dans un environnement pour étudier le comportement global du système (convergence).

Combien d'agents?
Compromis entre le nombre d'agents et la taille/nombre d'informations



A quel point distribuer le problème?
Compromis entre un problème global et une distribution locale aux agents

Agent vs Objet

Propriétés	Agent	Objet
Encapsulation de l'état interne (données)	Oui	Oui
Autonomie: capacité de réagir seul sans intervention	Oui	Non Une méthode doit être invoquée par un autre objet pour pouvoir accomplir ses effets.
Comportement	Pro-actif : Cherche à atteindre ses buts en saisissant les opportunité qui s'offrent à lui.	Passif : ne réagit que lorsqu'il reçoit un message.
Exécution de l'action spécifiée dans un message	Liberté de décision	toujours
Sociabilité	Capable de s'engager dans des interactions plus ou moins complexes : coopération, négociation, collaboration...	Non

Propriétés des agents

Persistence

- un agent cherche à satisfaire de manière persistante ses buts tant qu'il :
 - pense que c'est encore possible (précond. logique)
 - possède les ressources pour le faire (précond. physique)

Adaptation

- face à un environnement dynamique, un agent modifie constamment ses plans pour atteindre ses buts

Apprentissage

- les agents améliorent leurs performances à mesure que le temps passe

Bénévolat

- les agents ne doivent pas entrer en conflit avec les autres,
- ils doivent toujours essayer de faire ce qu'on leur demande (au risque de délaisser leurs propres tâches)

Sincérité

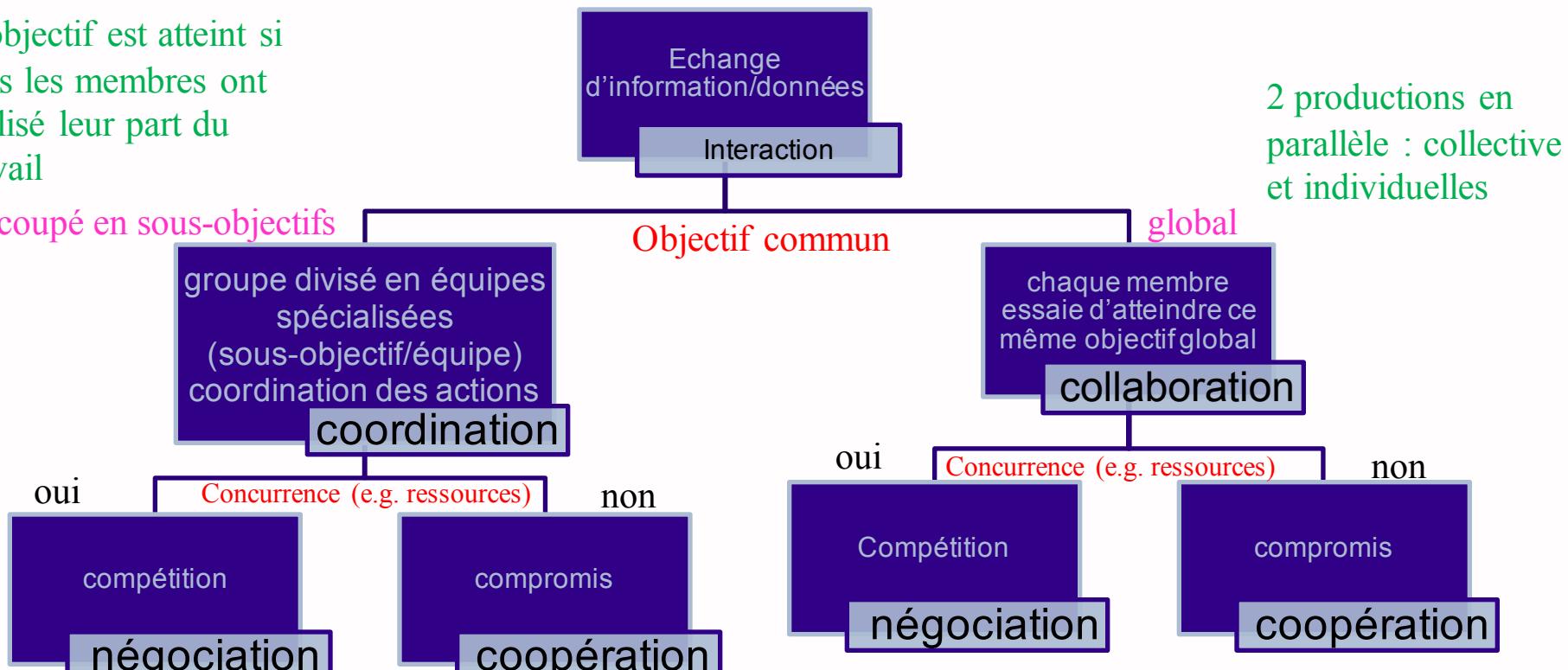
- un agent ne pourrait «en connaissance de cause» communiquer de fausses informations

...

Formes d'interaction

L'objectif est atteint si tous les membres ont réalisé leur part du travail

Découpé en sous-objectifs

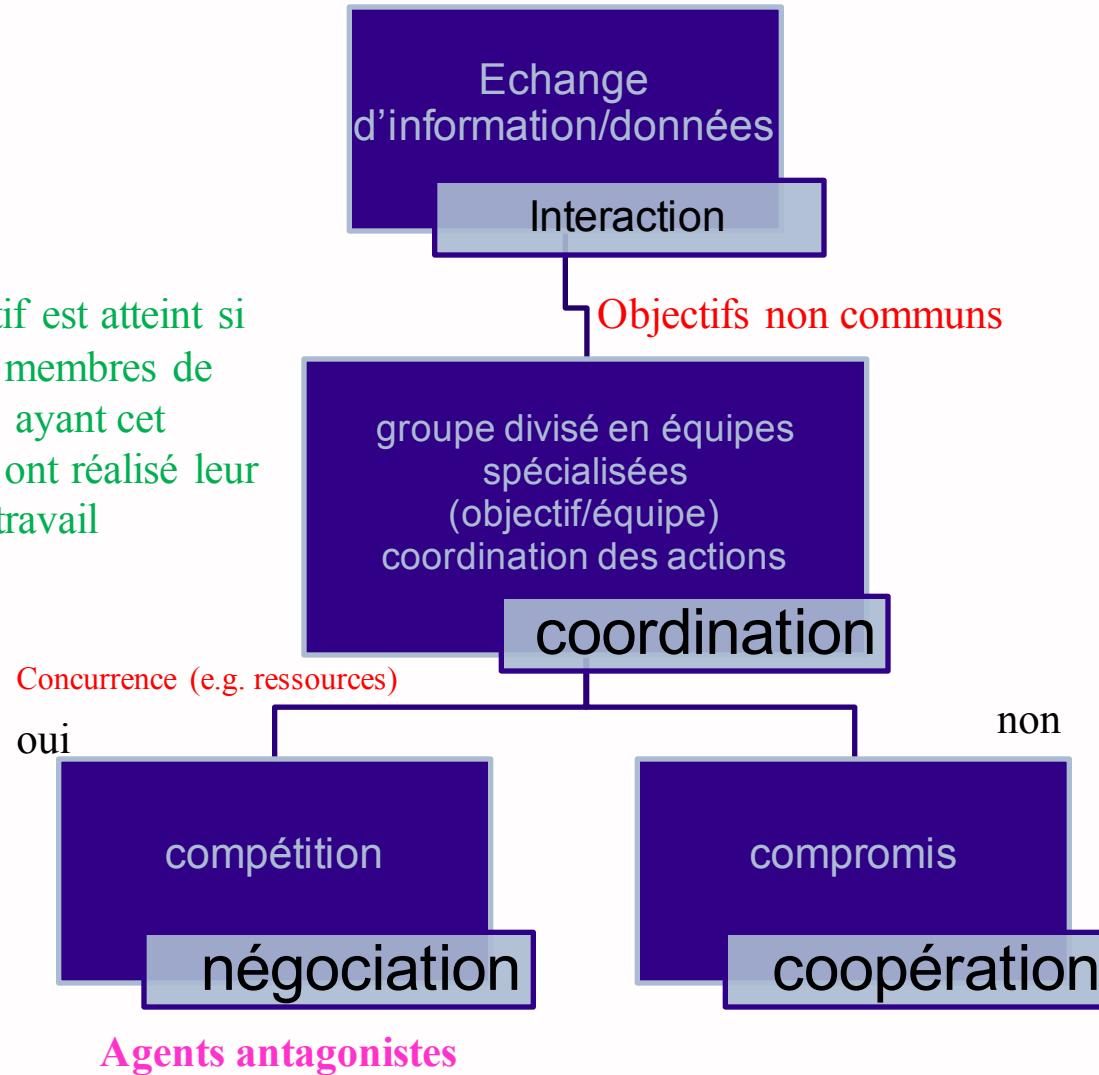


Agents antagonistes

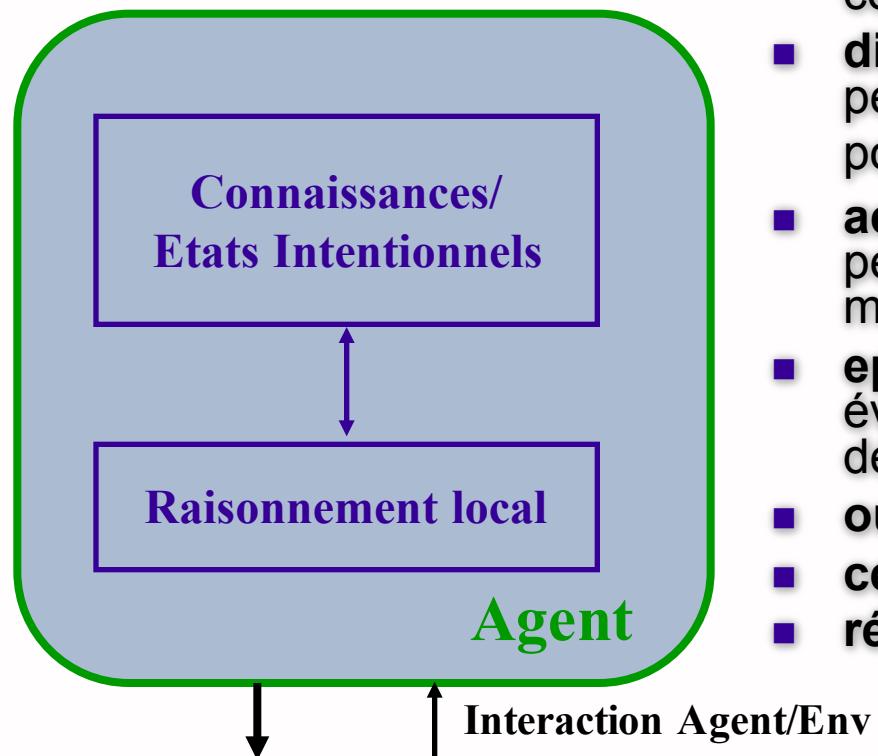
2 productions en parallèle : collective et individuelles

Formes d'interaction

L'objectif est atteint si tous les membres de l'équipe ayant cet objectif ont réalisé leur part du travail



Environnement: caractéristiques

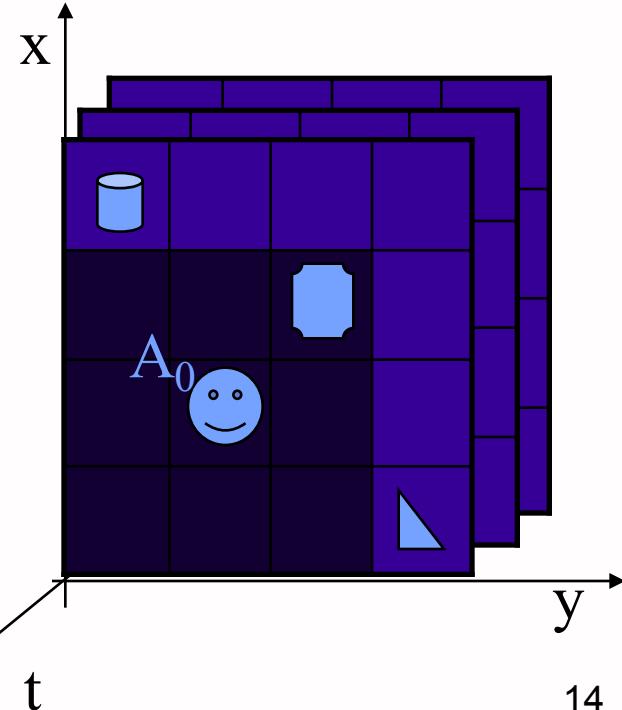
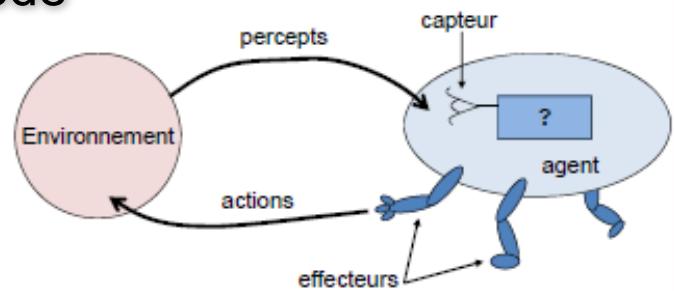


- **statique/dynamique;**
- **déterministe/non déterministe:** si une action du système sur \mathcal{E} a un effet unique et certain;
- **discret/continu:** si l'ensemble des perceptions et l'ensemble des actions possibles sur \mathcal{E} sont finis;
- **accessible/non accessible:** si le système peut être perçu à chaque instant d'une manière complète et précise;
- **épisodique/non-épisodique :** les prochaines évolutions ne dépendent pas des actions déjà réalisées;
- **ouvert/fermé,**
- **centralisé/distribué,**
- **réel/simulé.**

Agent: agir en contexte

Un agent A_0 est situé dans un espace où il possède un voisinage, il peut:

- Percevoir son environnement (**capteurs**);
- Agir sur l'environnement (**effecteurs**)
 - **Relativement**: en se déplaçant...
 - **Absolument**: par modification de l'état physique ou mental des choses du voisinage (objets, d'autres agents, humains...).

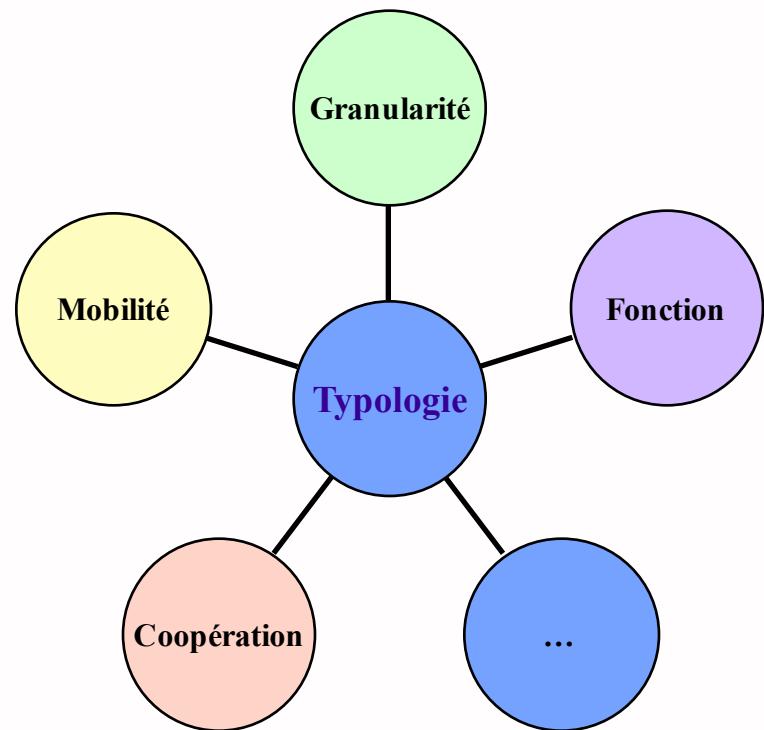


Le changement d'état de l'agent et/ou des choses modifie l'environnement (dimension temporelle).

Typologie des Agents

- Classification selon plusieurs critères:

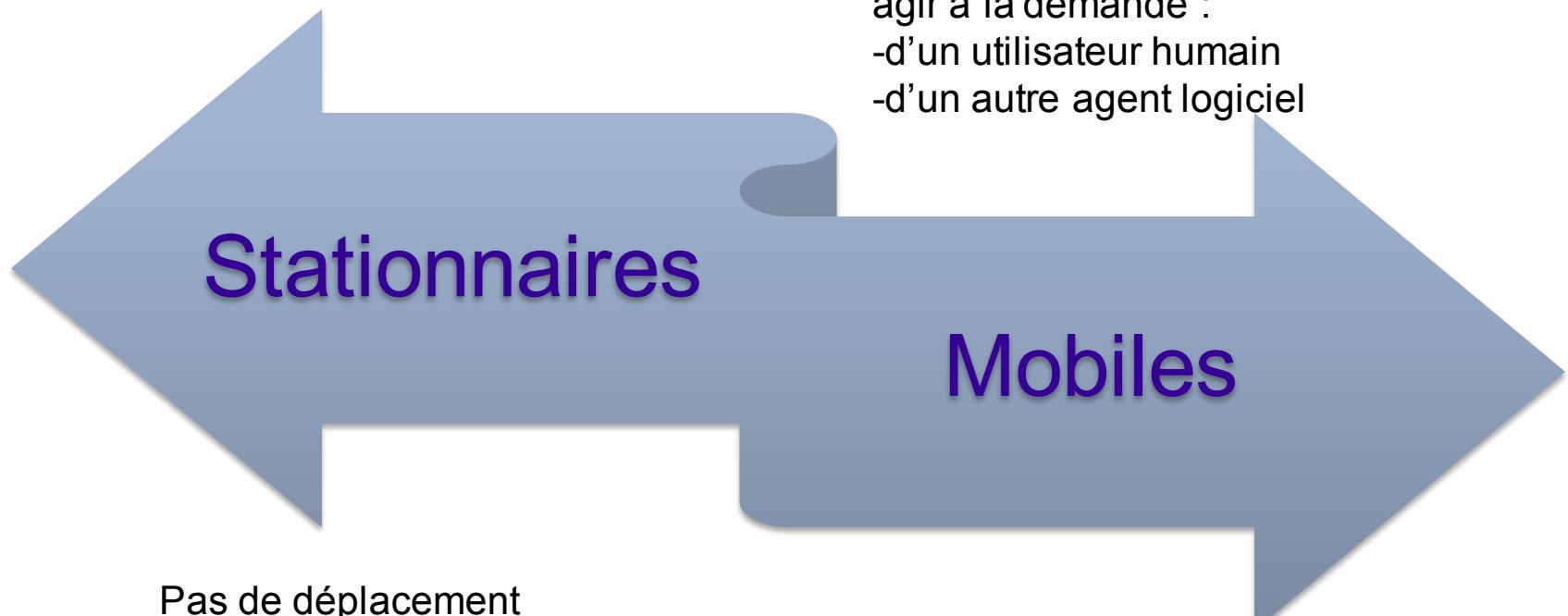
- fonction
- mobilité
- granularité
- capacité de coopération
- ...



Classification des agents selon la fonction

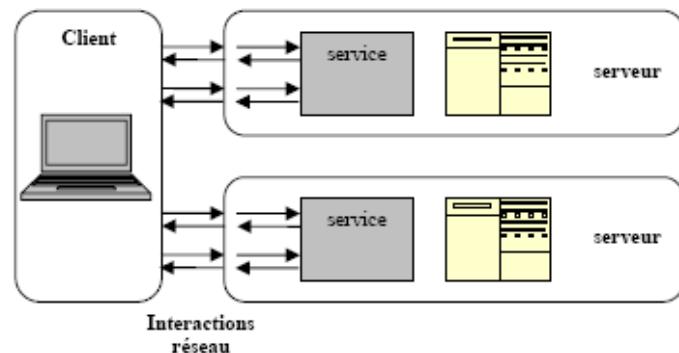
- Rôle joué par l'agent dans le système
- Par exemple:
 - *agent Interface* : joue le rôle d'interface entre les utilisateurs et le système,
 - *agent de détection d'intrusions*: repérer les tentatives de nuire à un système informatique;
 - *agent d'information/base de données*: gestion d'un grand volume d'information; fouille de données « datamining », collecte de données, recherche de l'information, traitement parallèle des requêtes;
 - *agent de commerce*: shopping;
 - *agents ordonnanceur* : ordonne le déroulement de tâches dans un système;
 - *agent optimisateur* : cherche la solution optimale à un problème selon un ou plusieurs critères (temps, coût...) ;
 - ...

Classification des agents selon la mobilité

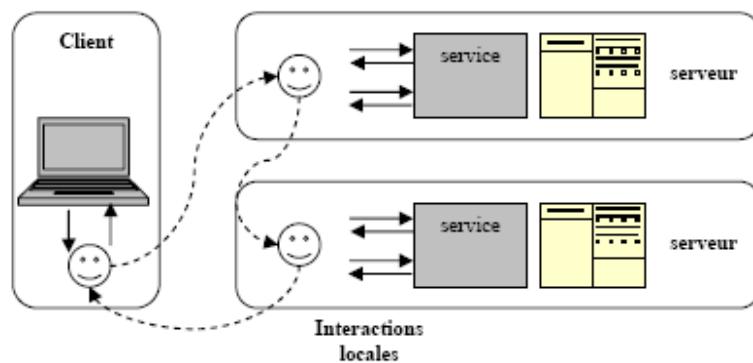


Les Agents Mobiles (AMs)

modèle client/serveur

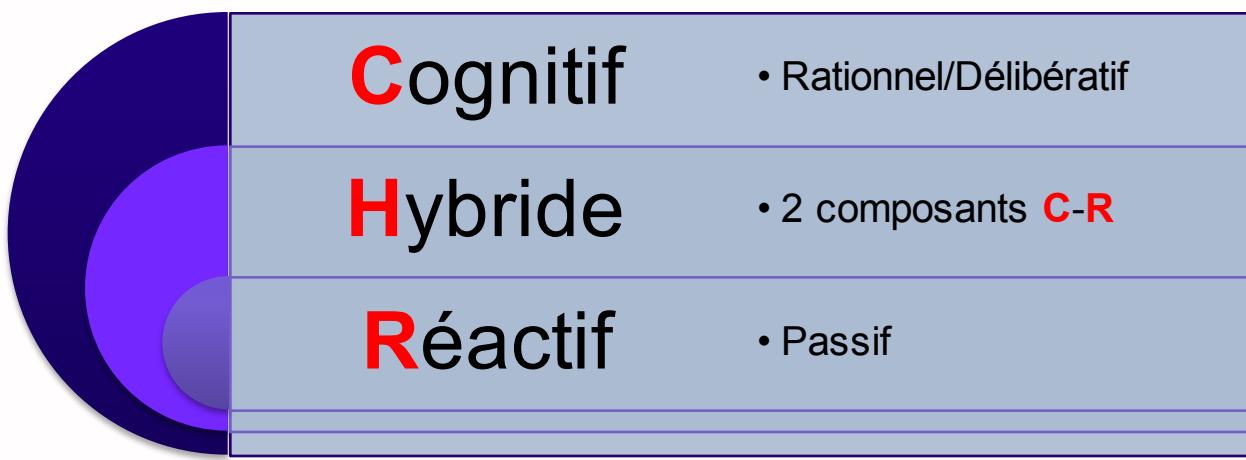


modèle AM



Classification des agents selon la granularité

- **Intelligence/rationalité** = raisonnement, prise de décision, apprentissage/adaptation, compréhension, planification...
- Intelligents: autonomes et flexibles dans un environnement dynamique
- Degrés d'intelligence :



Agent Réactif : Exemple Agent Garde (jeux)

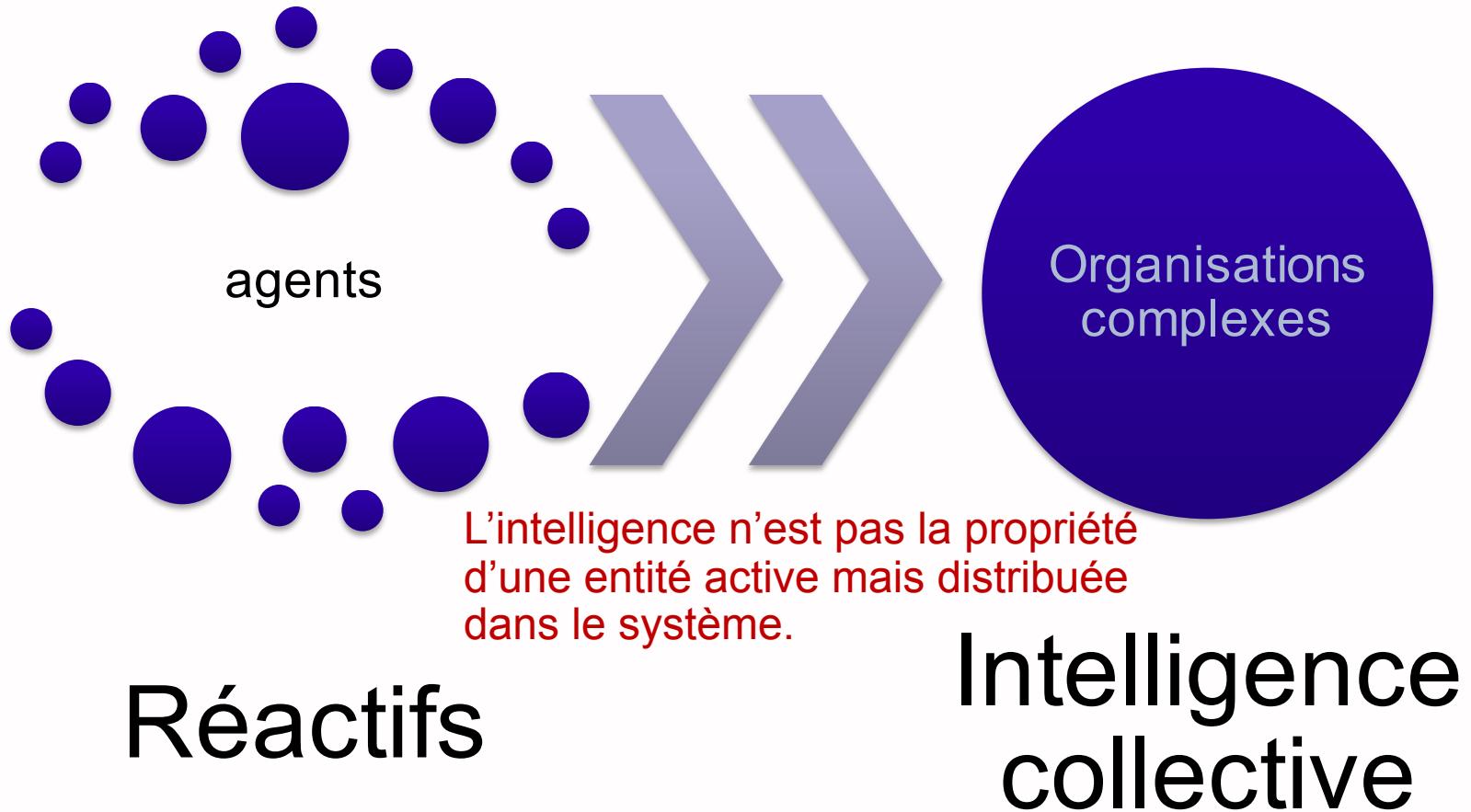
■ Les règles suivies par l'agent garde :

- Tant que je ne vois rien, je suis mon chemin de garde ;
- Si je vois un ennemi:
 - S'il n'est pas menaçant et si je ne suis pas blessé alors je l'attaque,
 - S'il est menaçant ou si je suis blessé, je sonne l'alarme et je m'éloigne.



J.Ferber (2009/2010) Université de Montpellier

L'interaction d'agents simples peut faire émerger des organisations complexes

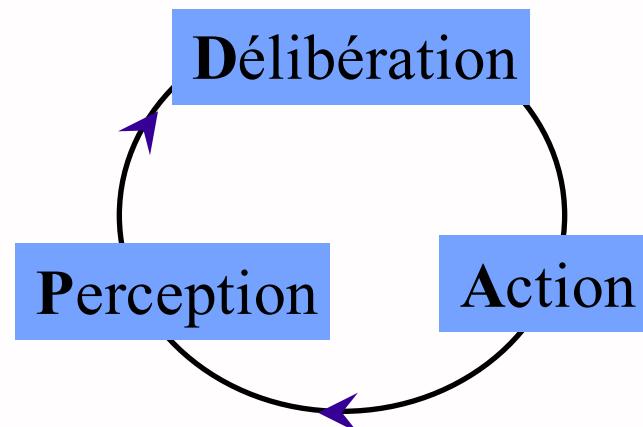


Agent Réactif : l'algorithme

```
do {  
    percepts = perceive();  
    action = getAction(percepts, rules);  
    execute(action);  
} while (true);
```

Agent Cognitif (Délibératif/Rationnel)

```
State state = initialize();
while (true) {
    percepts = perceive();
    state = updateState(state, percepts);
    action = deliberate(state, plans);
    execute(action);
}
```



Modèles Cognitifs

- Les agents doivent être dotés d'un **outil d'abstraction** pour les décrire comme des **systèmes intentionnels**
 - utilisation des notions de psychologie
 - moyens : attitudes mentales
- Choix d'un modèle cognitif, plusieurs modèles existent
- Exemple de modèle classique pour l'implémentation des agents Rationnels:
 - **BDI** (Beliefs, Desires, Intentions)
 - **BUC** (Beliefs, Uncertainty, Choice(goal))

Modèles Cognitifs & attitudes mentales

Connaissances

- sur le monde extérieur
- «je sais que les êtres humains sont mortels»

Croyances

- résultat des connaissances de l'agent et de ses perceptions de l'univers auquel il appartient
- peuvent être vraies ou fausses (**croyance vraie = connaissance**)
- «je crois que le distributeur de café fonctionne »

Désirs

- buts ou objectifs (États à atteindre)
- peuvent être contradictoires
- ensemble des opportunités offertes à l'agent à partir des croyances

Choix

- « j'ai décidé de prendre un café au distributeur »

Intensions

- désirs retenus que l'agent a décidé d'accomplir
- ne doivent pas être contradictoires
- peuvent se traduire en actions
- « j'ai l'intention de terminer ce rapport aujourd'hui»

Engagements

- Persistance
- « je continue à travailler tant que ce rapport n'est pas encore terminé »

Agent Cognitif : l'algorithme

Algorithme de la fonction de décision d'un agent BDI

Boucle de contrôle de l'agent

$B = B_0$

$I = I_0 \quad D = D_0$

TQ Vrai faire

percevoir(p)

$B = \text{brf}(B, p)$

$I = \text{options}(D, I)$

$D = \text{des}(B, D, I)$

$I = \text{filtre}(B, D, I)$

$\pi = \text{plan}(B, I)$

exécuter(π)

fin TQ

Révision des croyances

Génération des options

Mise à jour des désirs

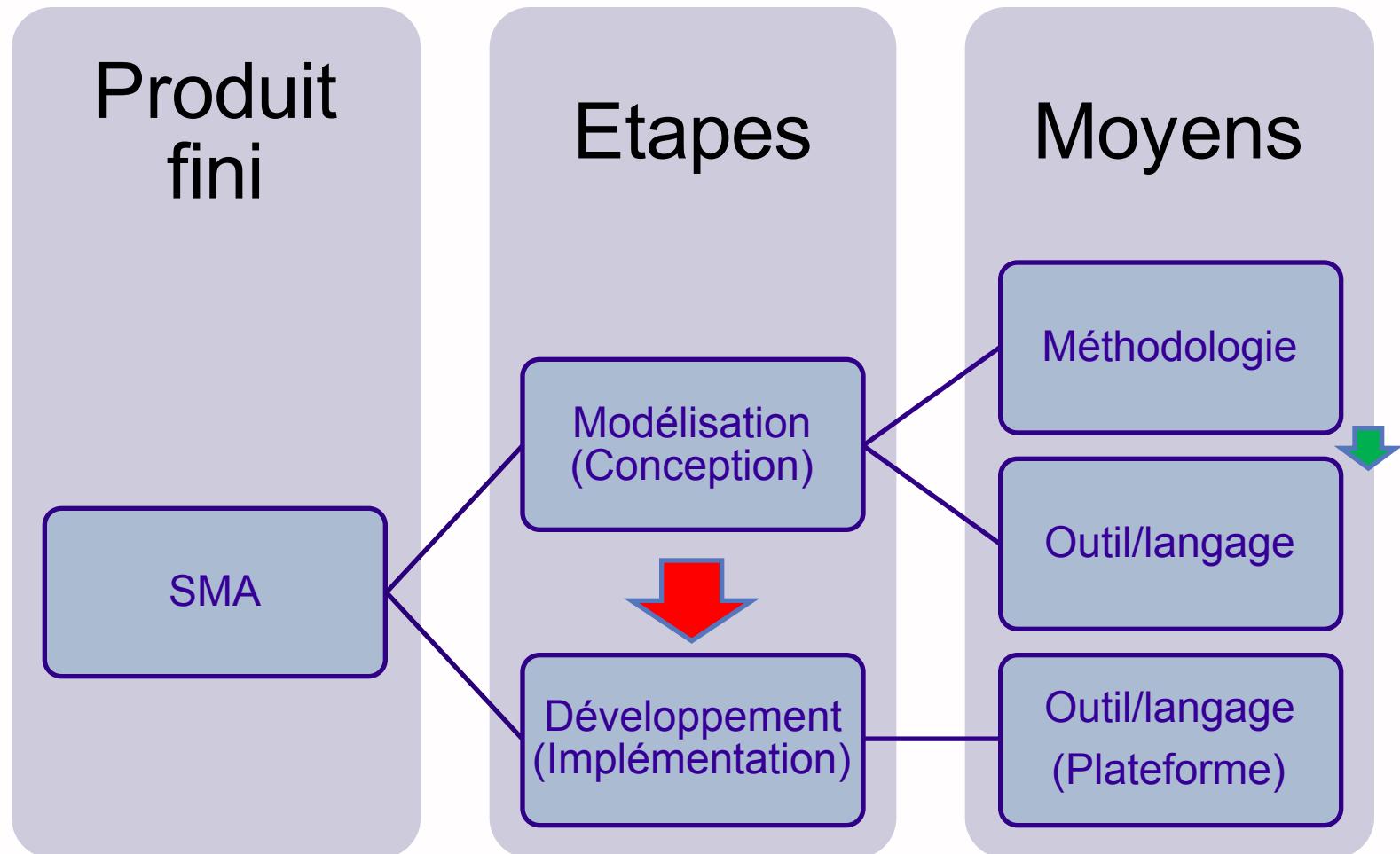
Prise de décision

Activation des intentions concrètes (« intention en action ») à partir d'intentions plus abstraites (« intentions préalables »)

Les modèles cognitifs: difficultés

- **IA** : « l'intelligence est une propriété d'une entité active »
- **IAD** : Mise en œuvre du fonctionnement de la structure de tous les composants « intelligents »
 - ! Avoir des comportements stables et cohérents même dans un environnement dynamique
 - ! Prise en compte des intérêts et buts des autres agents: certains buts ne peuvent être réalisés qu'avec le concours des autres agents.

Comment réaliser un SMA?



Modélisation SMA : définir l'architecture SMA

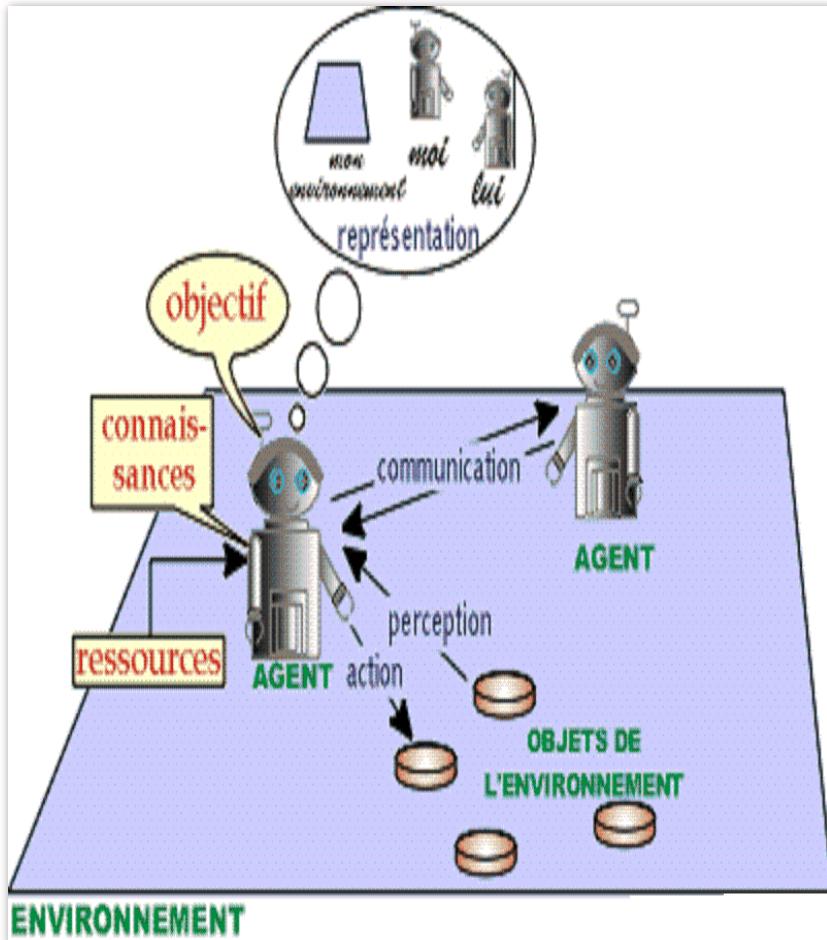
Représentation de :

L'environnement

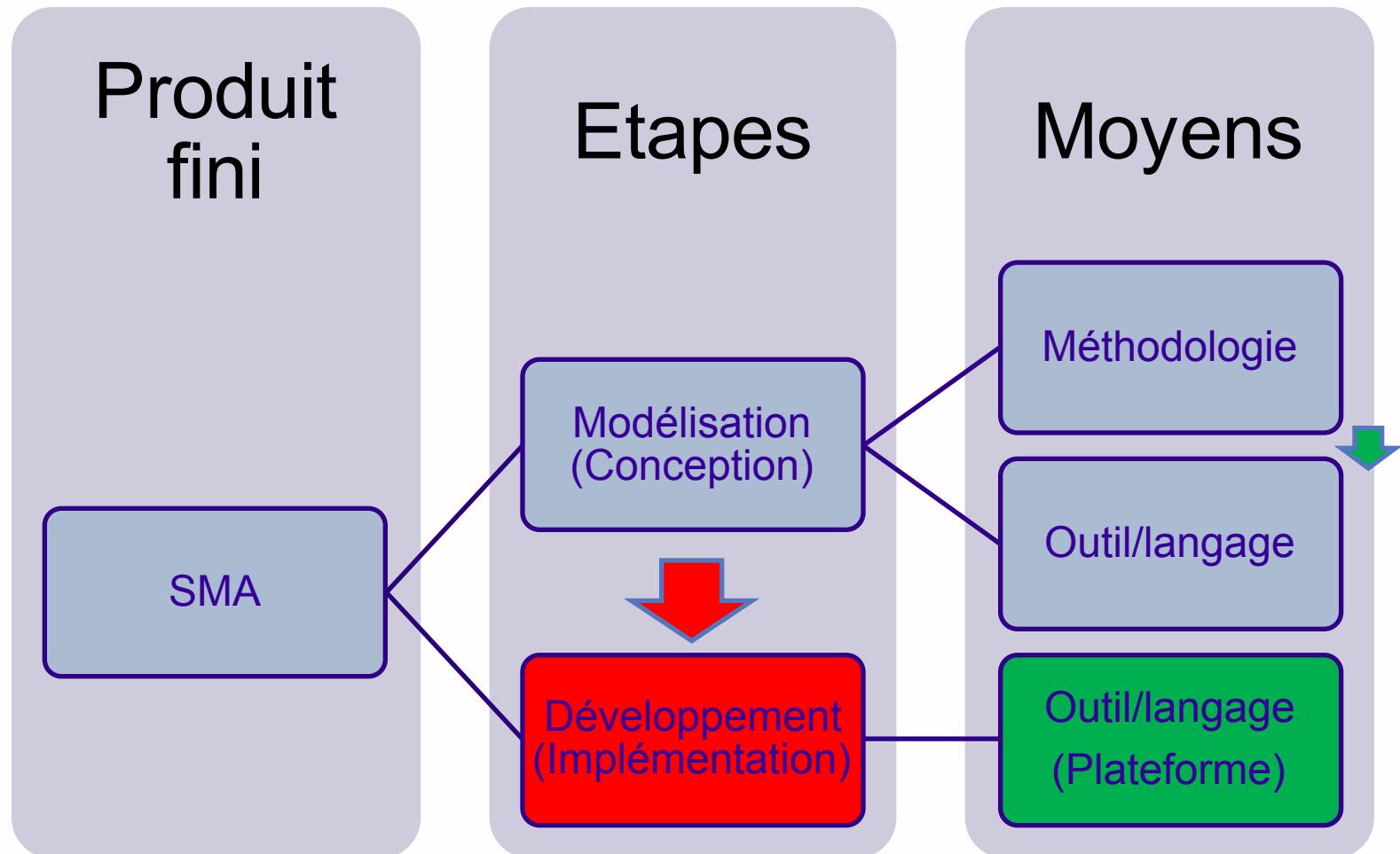
1. composition (objets & agents)
2. caractéristiques

Les agents du système, agent définir dans le cas échéant :

1. Attributs (nom, position, âge...)
2. Attitudes mentales (croyances, objectifs...)
3. Rôles
4. Comportements pour chaque rôle :
 - a) Interaction (protocoles d'interaction)
 - a) Collaboration
 - b) Négociation
 - c) ...
 - b) Décision
5. Les ressources utilisées pour atteindre les objectifs...



Comment réaliser un SMA?



Développement : Plateformes & Langages

- Plateforme multi-agent est l'infrastructure qui supporte pour les SMA:
 - Création
 - Manipulation
 - Expérimentation.
- Se caractérise par:
 - Les modèles mis en œuvre
 - Les procédures d'interactions et de communications
 - La modélisation des environnements et des connaissances

Développement : Plateformes & Langages

- Généralement deux types:
 - Générique
 - Spécifique (dédiée à un domaine particulier)
- Outils:
 - D'analyse
 - De conception
 - De développement
 - D'expérimentation

Choix de la plateforme

- Disponibilité?
- Configuration?
- Documentation?
- Applications?
- Hétérogénéité?
- Adaptabilité?
- Standards?
- Interopérabilité?
- Distribution?
- Facilité de développement?
- Facilité d'expérimentation?
- Mécanismes de simulation?
- Mobilité des agents?
- Outils graphiques?
- ...

Quelques Plateformes

- AGENT-BUILDER (java)
 - Les agents de type BDI...
- IBM AGLETS (java)
 - Implémente la mobilité...
- CONCORDIA (java)
 - Implémente la mobilité...
- JACK (Jack Agent et java)
 - Les agents de type BDI...
- JADE (java)
 - Implémente la mobilité
 - Compatible FIPA...
- MadKit (java)
 - Modèle d'organisation Alaadin AGR (Agent/Groupe/Rôle)
- JEDI
 - Modèle IODA
- ...

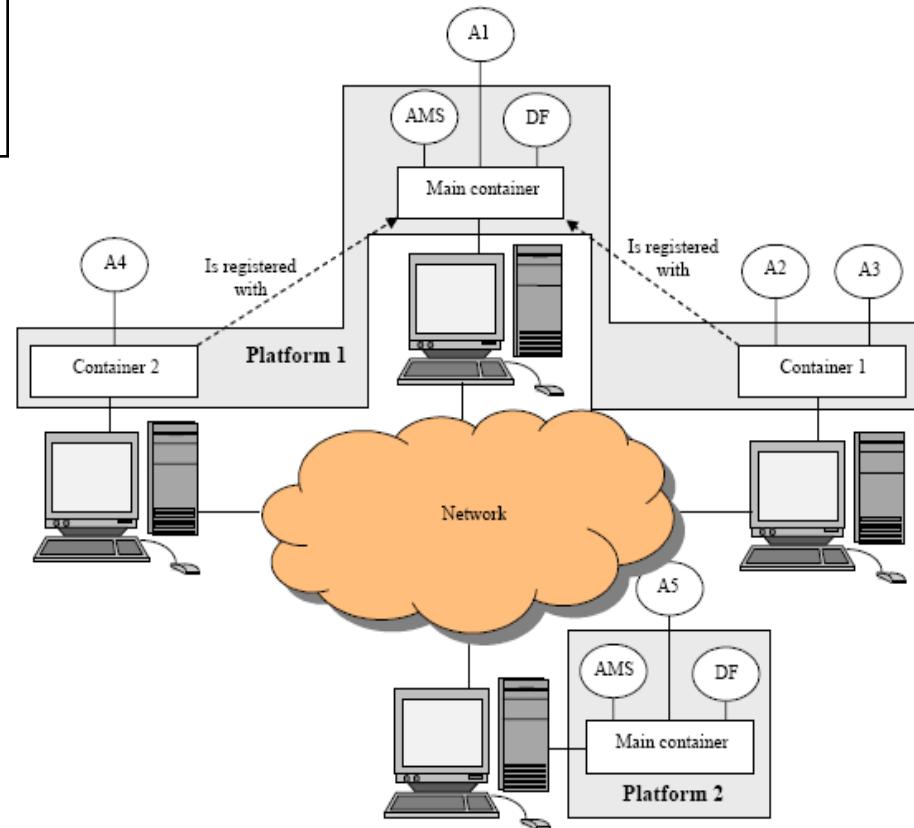
Plate-forme: JADE

- Java Agent Development framEwork: Un environnement d'exécution
- Un middleware (communication entre applications réparties)
- Implémentation flexible des SMA
- Le langage de Communication: FIPA-ACL
- Écrit en java, supporte la mobilité, évolue
- Intégration des web-services
- Une librairie de classes pour le développement des agents à utiliser:
 - Directement
 - Par spécialisation
- Des outils graphiques pour l'administration et le contrôle des activités des agents au cours d'exécution: Sniffer Agent, Remote Management Agent (RMA), Dummy Agent, The Introspector Agent, ...

JADE: containers et plateformes

JADE est un *middleware* permettant à des applications réparties de communiquer entre elles dans un réseau.

- **Plateforme:** un ensemble de containers actifs
- **Container:** un environnement d'exécution, contient un ou plusieurs agents,
- **Main-container:** un container spécial unique dans la Plateforme avec qui un container « normal » doit s'enregistrer.



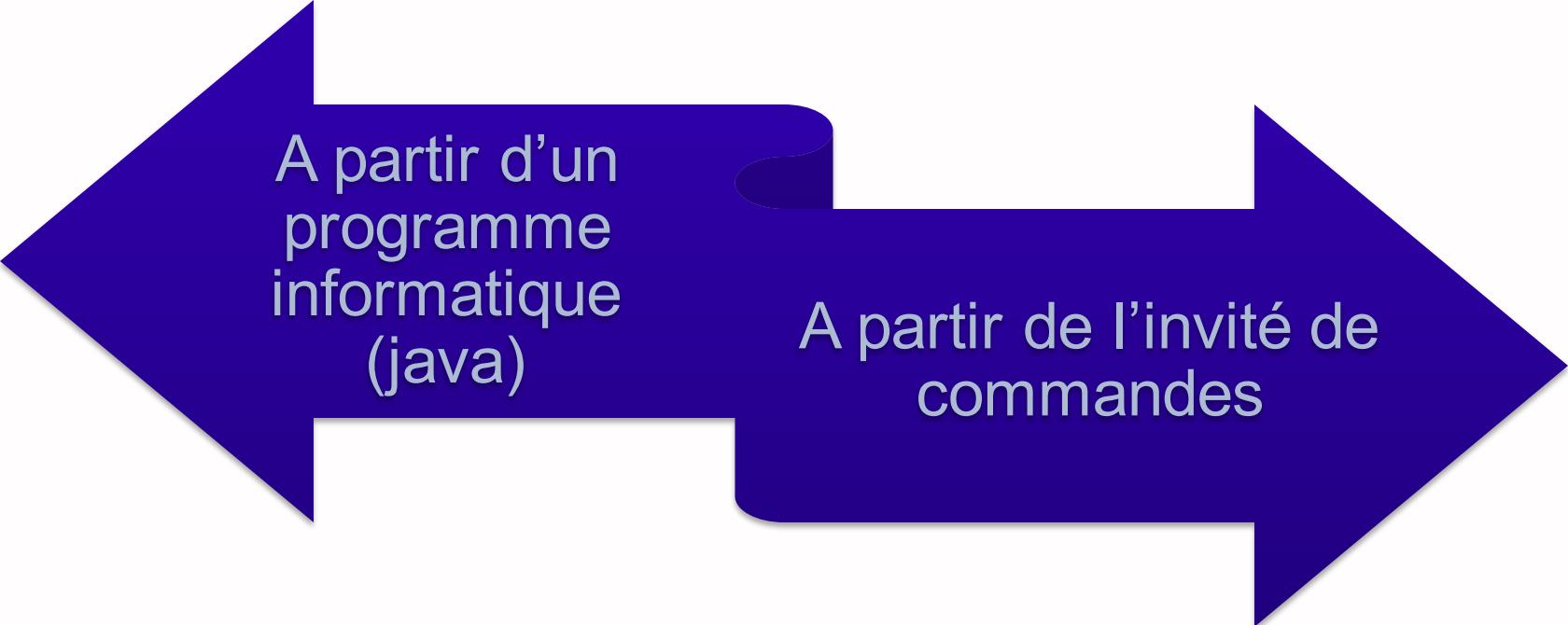
TP N° 1

Installation de la plateforme et
lancement d'un premier agent par un
programme Java et par la plateforme
directement

Installation de la plateforme

- Télécharger la dernière version de jade :
<http://jade.tilab.com/>
- Une archive : JADE-all-x.y.z.zip
- Choisir un emplacement et décompresser l'archive (4 dossiers):
 - JADE-doc-x.y.z : la documentation
 - JADE-bin-x.y.z : l'archive java (API) de JADE
 - JADE-src-x.y.z : le code source de JADE
 - JADE-examples-x.y.z : les exemples

Lancement de la plateforme



A partir d'un
programme
informatique
(java)

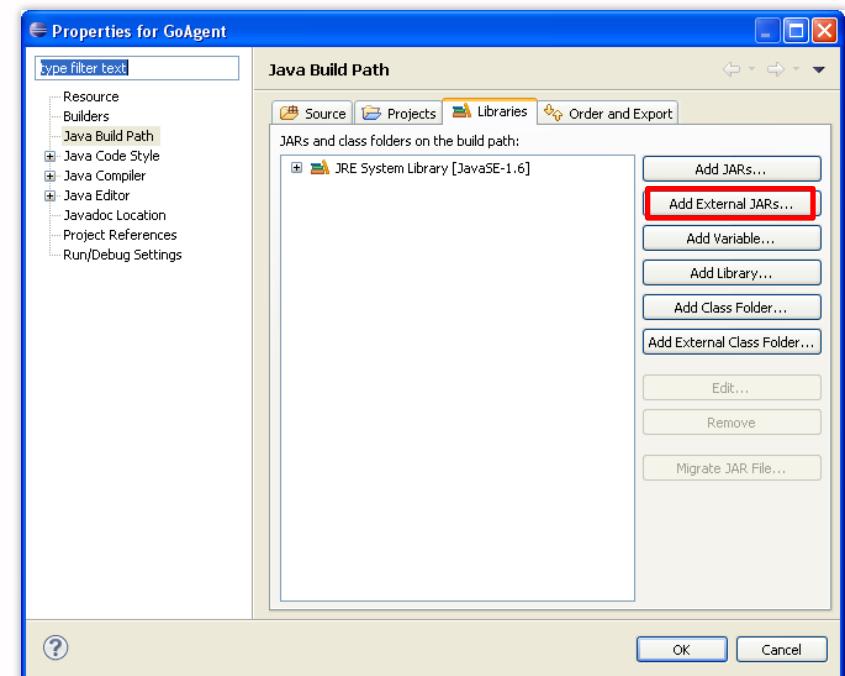
A partir de l'invité de
commandes

Lancement de la plateforme JADE à partir d'un programme java

■ Etape N° 1:

Utilisation de l'éditeur Eclipse

1. Créez un projet java Eclipse
2. Accédez aux propriétés du projet (menu Projet Propriétés)
3. Jade Build Path (Add External JAR)
4. Ajoutez tous les « **.jar** » de Jade retrouvés dans le répertoire « **bin/lib** »



Lancement de la plateforme JADE via un programme java

■ Etape N° 2: Programme principal

- Application Programming Interface de Jade :
<http://jade.tilab.com/doc/api/index.html>

```
import jade.core.ProfileImpl;  
import jade.wrapper.AgentContainer;  
import jade.wrapper.AgentController;
```

Etape	Rôle	Code
1	Création d'une instance de l'environnement Jade	jade.core.Runtime rt = jade.core.Runtime.instance();
2	Création d'un profil de Container par défaut pour lancer la plateforme (création du main container)	ProfileImpl pMain = new ProfileImpl() ;
3	Création du main container	AgentContainer mc = rt.createMainContainer(pMain);
Option	Interface Graphique de JADE : l'agent RMA	
4	Créer l'agent RMA	AgentController rma = mc.createNewAgent("rma", "jade.tools.rma.rma", null);
5	Lancer l'agent RMA	rma.start();

Sélectionner le fichier du main → Run as Java Application

Attention pour arrêter la plateforme il ne suffit pas de fermer l'interface graphique : console en bas cliquer sur le carré rouge et les croix tant que c'est actif !

La plateforme (Interface Graphique)

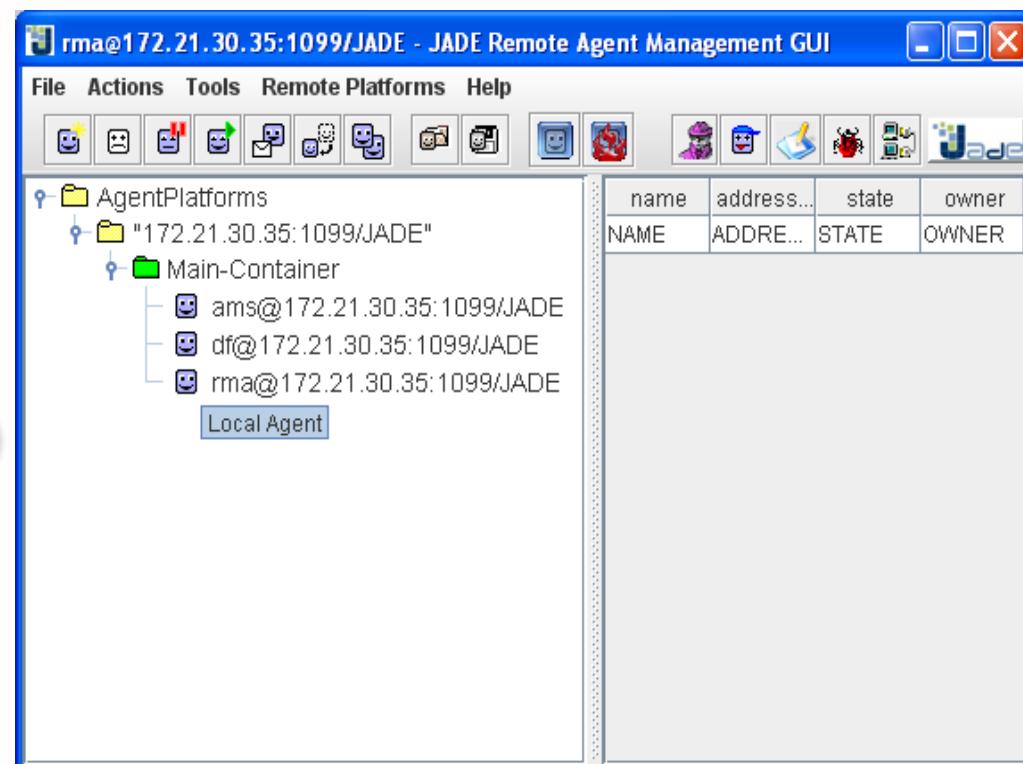
Le **Main-Container** Contient 3 agents spéciaux:

Remote Management Agent (RMA):
Console de gestion et de contrôle de la plateforme.

Agent Management System (AMS):

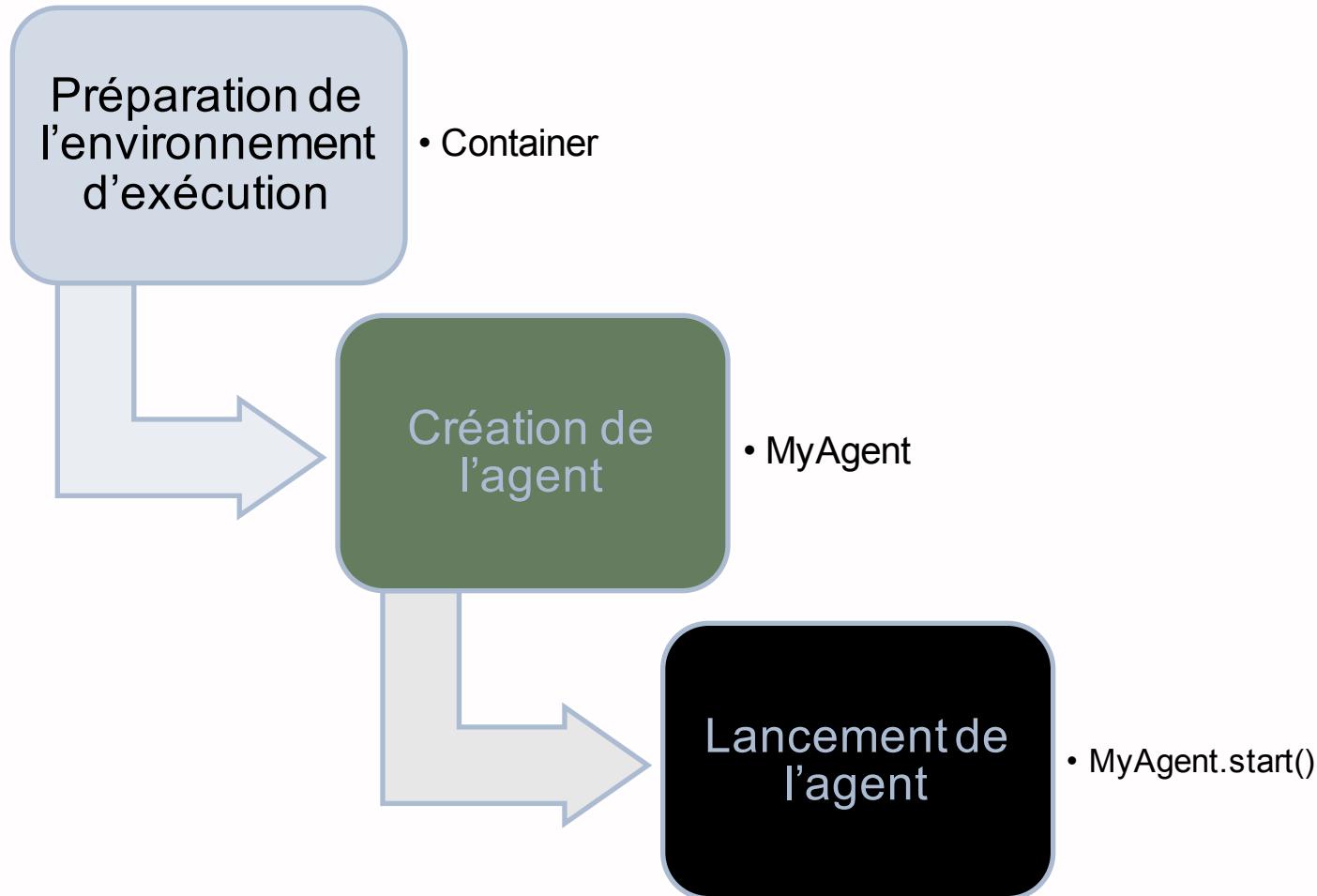
- Service de nommage: un nom unique (**Global Unique IDentifier**) pour chaque agent et pour chaque container.
- Création/destructions d'agents

Directory Facilitator Agent (DF):
Service de « pages jaunes »: un agent peut chercher des agents qui offrent les services dont il a besoin pour atteindre son but.

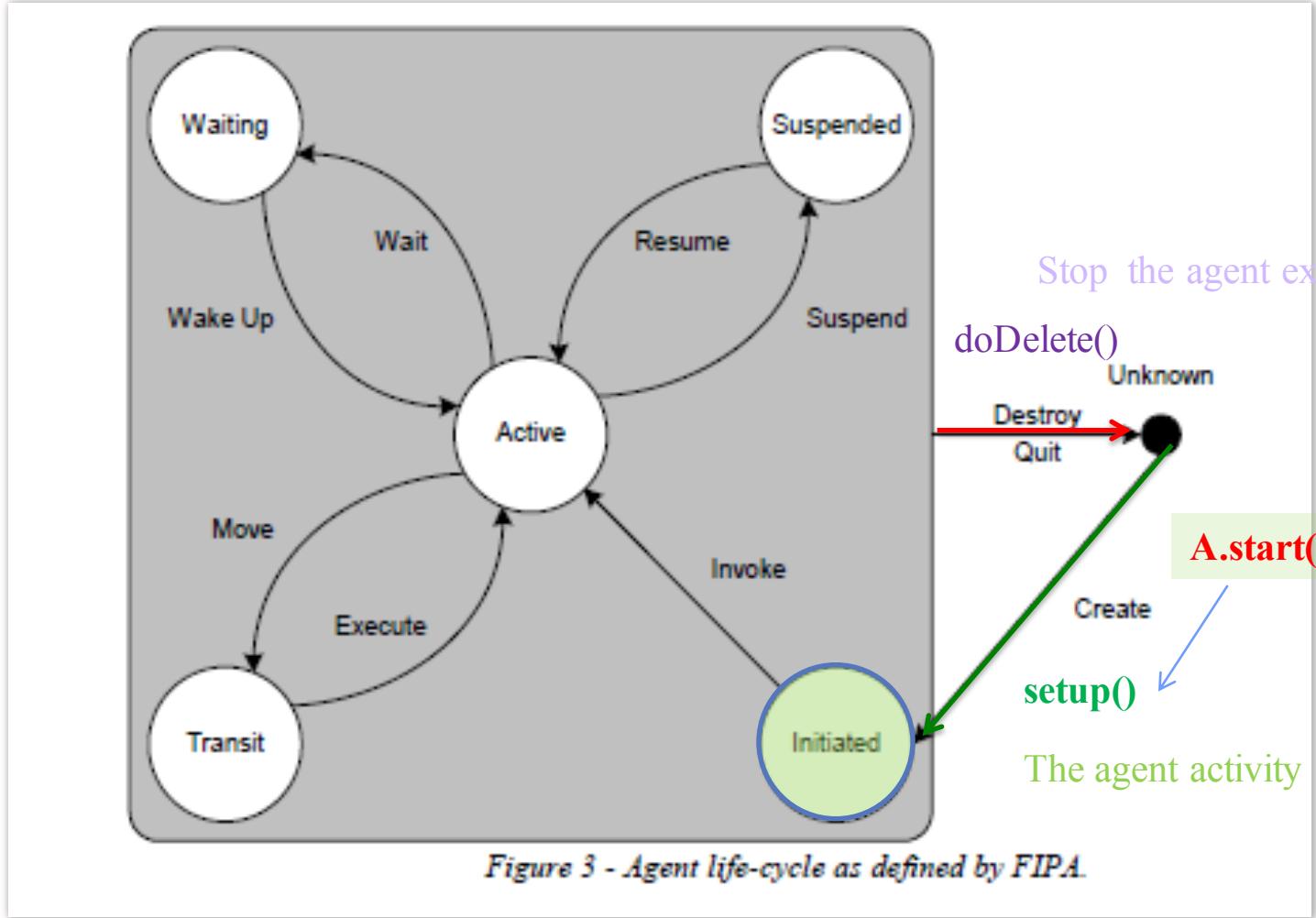


Création d'un agent

classe publique java qui hérite de jade.core.Agent



Agent JADE



La méthode setup()

- Programme principal de l'agent (équivalent au main pour un programme java)
- Permet de :
 - initialiser l'agent
 - lui attribuer un ou plusieurs comportements
 - Un comportement ≡ une tâche à réaliser par l'agent

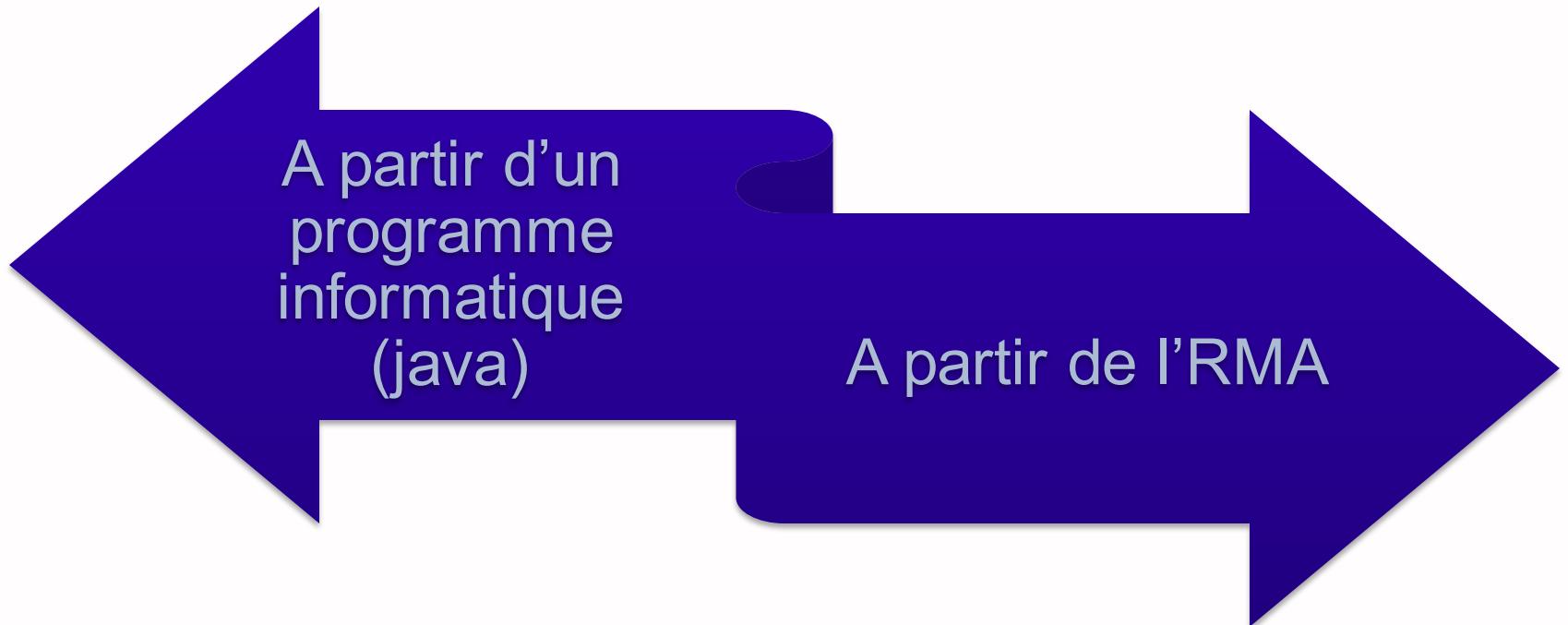
**Exemple de l'agent HelloWorld
(à implémenter):**

protected void setup() {

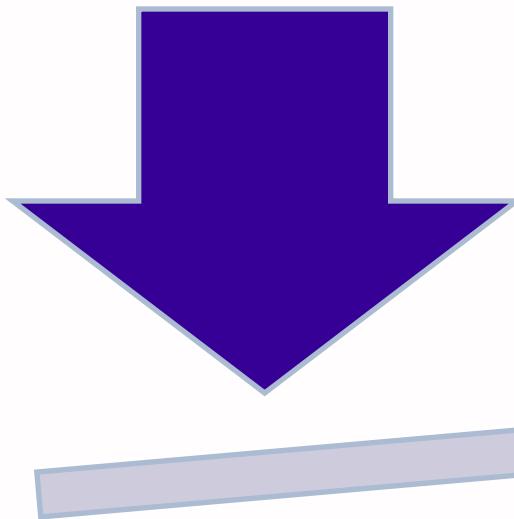
```
System.out.println("Hello World!  
My name is "+getLocalName());  
doDelete(); //détruit l'agent
```

}

Lancement d'un agent



A partir de l'RMA

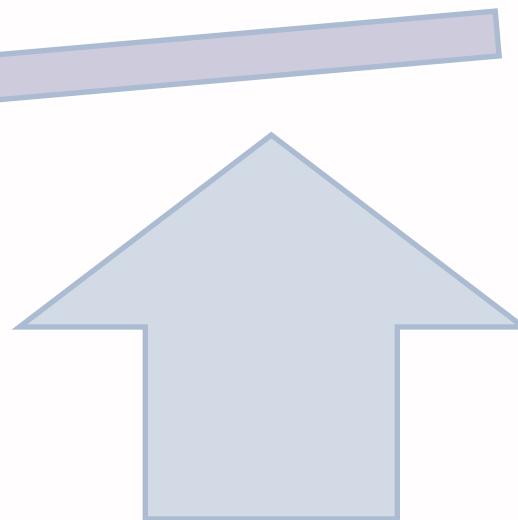


Lancer la plateforme à
partir d'un programme
Java

Lancer les agents à partir de l'RMA

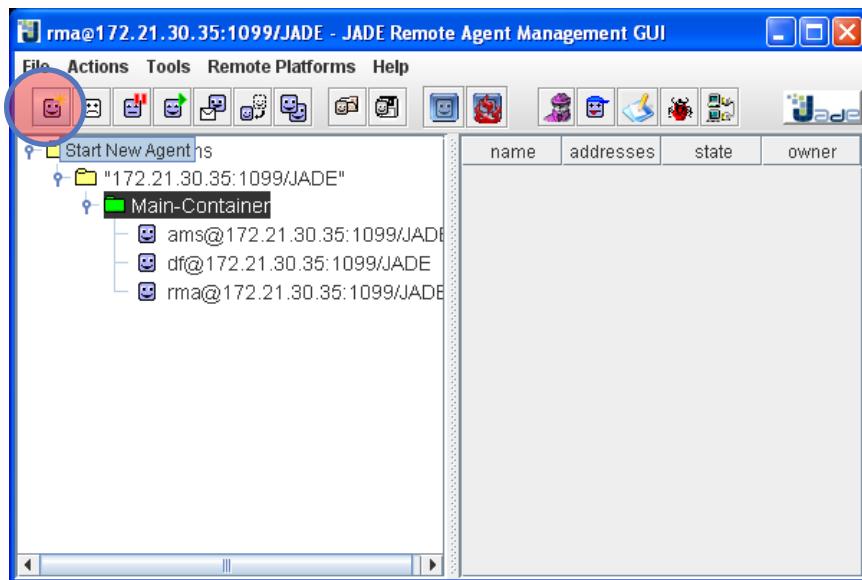
Il faut créer une bibliothèque
« Package » et mettre les agents
dedans... afin qu'ils soient visibles à
partir de l'RMA

Paramètres séparés par des virgules



Lancement de l'agent JADE à partir de l'RMA

Etape 1 : Lancer l'agent (Start new Agent)



- Nommer l'agent
- Choisir sa classe
- Le Paramétrer (si nécessaire)
 - Les paramètres sont séparés par des « , »
 - Exemple : Un agent qui a besoin de certaines informations pour démarrer son exécution
- Exemple de l'agent Hello
 - L'agent s'autodétruit, donc pas le temps de le voir dans la liste....

Lancement de l'agent JADE à partir du programme Java

- Utilisation de l'environnement ECLIPSE

```
AgentController hello = mc.createNewAgent("toto", "X>HelloAgent", null);  
hello.start();
```

- **toto** : le nom de l'agent
- **HelloAgent** : le nom de la classe (précédé par la bibliothèque **X**)
- **null** : pas de paramètres

Lancement de l'agent JADE à partir du programme Java

Lancer l'agent HelloAgent avec des paramètres entiers à afficher !

```
Integer tab[] = new Integer[2];  
tab[0] = new Integer(10);  
tab[1] = new Integer(20);
```

```
AgentController hello = mc.createNewAgent("toto", "X>HelloAgent", tab);  
hello.start();
```

- Toto : le nom de l'agent
- X>HelloAgent : le nom de la classe (précédé par la bibliothèque)
- tab : paramètres tableau de 2 entiers

L'agent récupère ses paramètres

```
public class HelloAgent extends Agent{
```

```
    Object tab[];
```

```
    protected void setup(){
```

```
        tab=getArguments();
```

```
        System.out.println("Hello my name is "+getLocalName());
```

```
        System.out.println("and I am "+tab[0].toString()+"years old");
```

```
    }
```

```
}
```

Comportements

- Exprime le travail (tâches) affecté à un agent.
- Un agent peut avoir un ou plusieurs rôles
- Pour jouer un rôle, un agent pour avoir un ou plusieurs comportements!

Les tâches à faire : Obligatoire

Super-classe

Hiérarchie des comportements

Condition d'arrêt

Behavior

+action()
+done()
+onStart()
+onEnd()
+block()
+restart()

Models a generic task

Models a task
that is made
of other tasks

A simple task

CompositeBehavior

SimpleBehavior

OneShotBehavior

CyclicBehavior

A cyclic task.
Its done()
returns false.

done() returns true

FSMBehavior

+registerState()
+registerTransition()

SequentialBehavior

+addSubBehavior()

ParallelBehavior

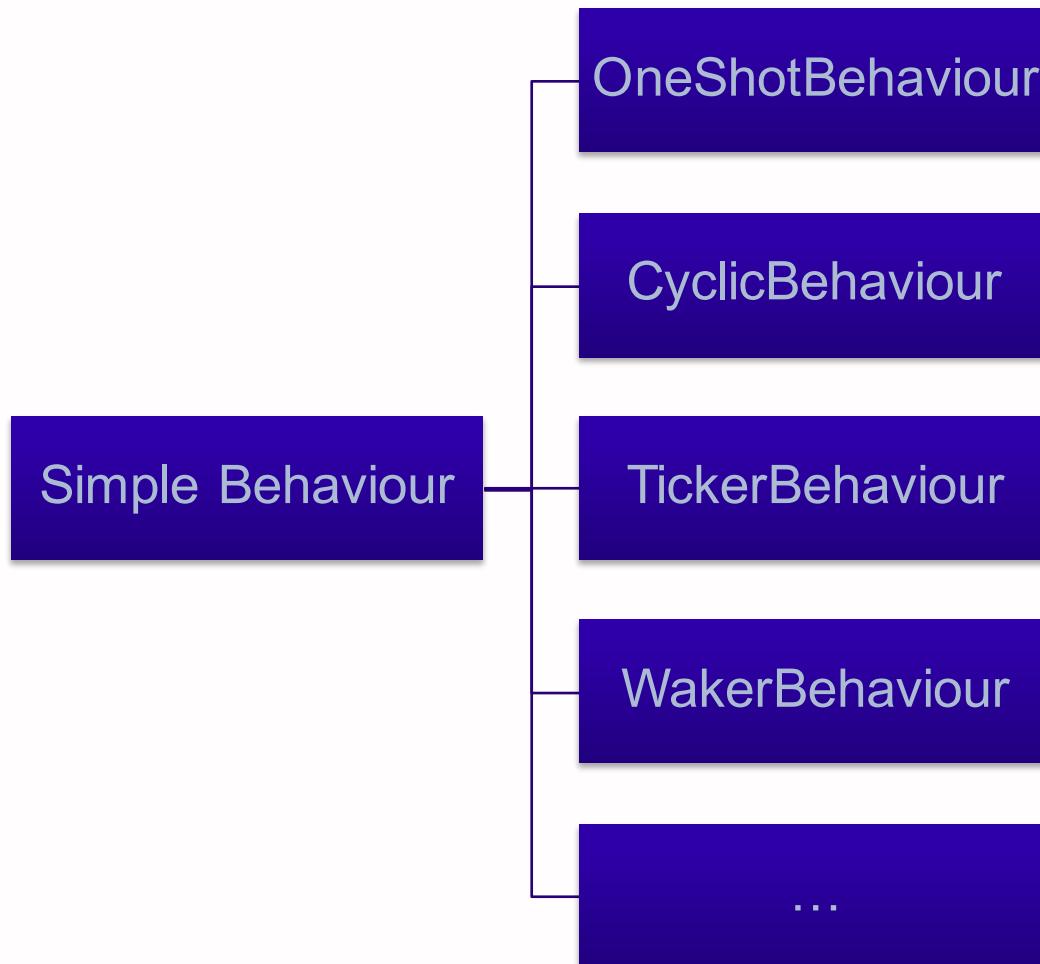
+addSubBehavior()

A complex task
whose subtasks
correspond to the
activities performed
in the states of a
finite state machine

Subtasks
are
executed
sequentially

Subtasks are
executed concurrently

Sous classes de Simple Behaviour



Ajout de comportements

3 méthodes

- Classe de comportement dans le même fichier que l'agent (e.g.classe interne)
 - Classe indépendante (non publique)
 - Classe interne
- Classe anonyme : utilisation ponctuelle du comportement.
- Package de comportement (l'idéal).

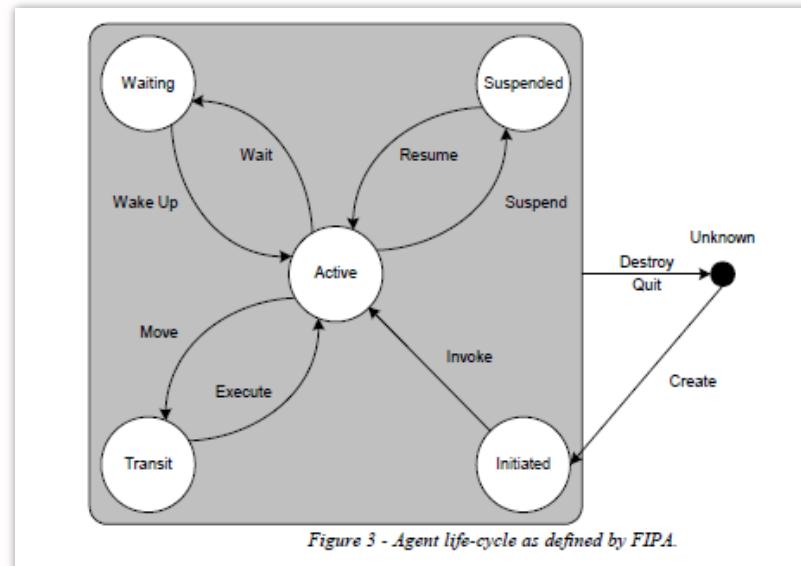
Comportements

0) Définir le(s) comportement(s) :

Un comportement est une sous-classe de **Behaviour** qu'il faut spécifier en implémentant :

- (1) Fonction principale du comportement selon le type de comportement: **action()**, **onTick()**...
- (2) La Condition d'arrêt : **done()** – ça dépend du type du comportement-
- (3) Le Traitement spécifique après la fin du comportement : **onend()** – facultative-

- 1) Ajouter les instances à la liste de tâches de l'agent avec **addBehaviour(instance)** dans le **setup()**
- 2) On peut bloquer (**block()** typiquement pour attendre l'arrivée d'un message) ou débloquer (**restart()**) un comportement (méthode de la classe Behaviour).
- 3) On peut bloquer un comportement pendant un délai exprimé en millisecondes (**block(long)**).



TickerBehaviour

- ***myAgent*** : fait référence à l'agent
- Sous classe de SimpleBehaviour
- Comportement cyclique avec une durée déterminée entre les cycles en ms
- Ici, le comportement est ajouté dans le setup de l'agent par :
`addBehaviour(new MyTickerBehaviour(this,1000));`

- Exemple :

```
class MyBehaviour extends  
TickerBehaviour{  
  
    public MyBehaviour(Agent a, long  
    period) {  
        super(a, period); //obligatoire  
    }  
  
    protected void onTick(){  
        System.out.println(myAgent.getLoc  
alName() + " Tick =  
"+getTickCount());  
    }  
  
}
```

L'attribut myAgent (jade.core.Behaviours)

- Référence de l'agent qui possède ce comportement
 - Permet l'appel des méthodes de l'agent à travers son comportement
- Exemple :

```
import jade.core.behaviours.CyclicBehaviour

class MyCyclicBehaviour extends CyclicBehaviour{
```



```
public MyCyclicBehaviour(Agent a) {
super(a); //Facultatif
}

public void action(){
System.out.println("received Message
from agent "+myAgent.getLocalName());
}
```

TickerBehaviour (classe anonyme)

- Ici, le comportement est ajouté dans le setup de l'agent en utilisant une classe anonyme.

addBehaviour

```
new TickerBehaviour(this, 1000) {  
    protected void onTick() {  
  
        System.out.println("Agent"+myAgent.getLocalName()  
            +": tick="+getTickCount());  
  
    }  
};
```

WakerBehaviour

- Sous classe de SimpleBehaviour
- Un comportement qui s'active au bout d'un laps de temps...

```
addBehaviour(new WakerBehaviour(this, 10000)
{
    protected void onWake() {
        System.out.println("Agent"+myAgent.getLocalName()+": It's wakeup-time. Bye... ");
        myAgent.doDelete();
    }
});
```

Behaviour

- Fonction principale :
public void action()
- Fonction d'arrêt :
public boolean done()
- Fonction d'initialisation :
public void onStart()
- Fonction de fin :
public int onEnd()
- ...
- Exemple de comportement :
 - Int i=0, n=5;
 - While (i<n)
 {println(i); i++;}

Introspector Agent



- Outil de débogage :
 - état (cycle de vie) d'un agent
 - ses comportements
 - ses messages reçus ou émis

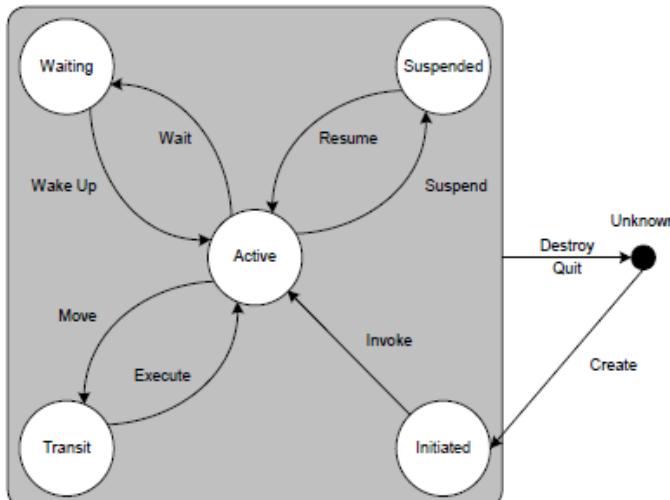


Figure 3 - Agent life-cycle as defined by FIPA.

The screenshot shows the JADE Introspector Agent interface for a complex named 'Complex@172.31.70.211:1099/JADE'. The window has tabs for View, State, and Debug. The State tab is active, displaying the current state of the agent as 'Active' and the incoming/outgoing message counts as 'Pending'.

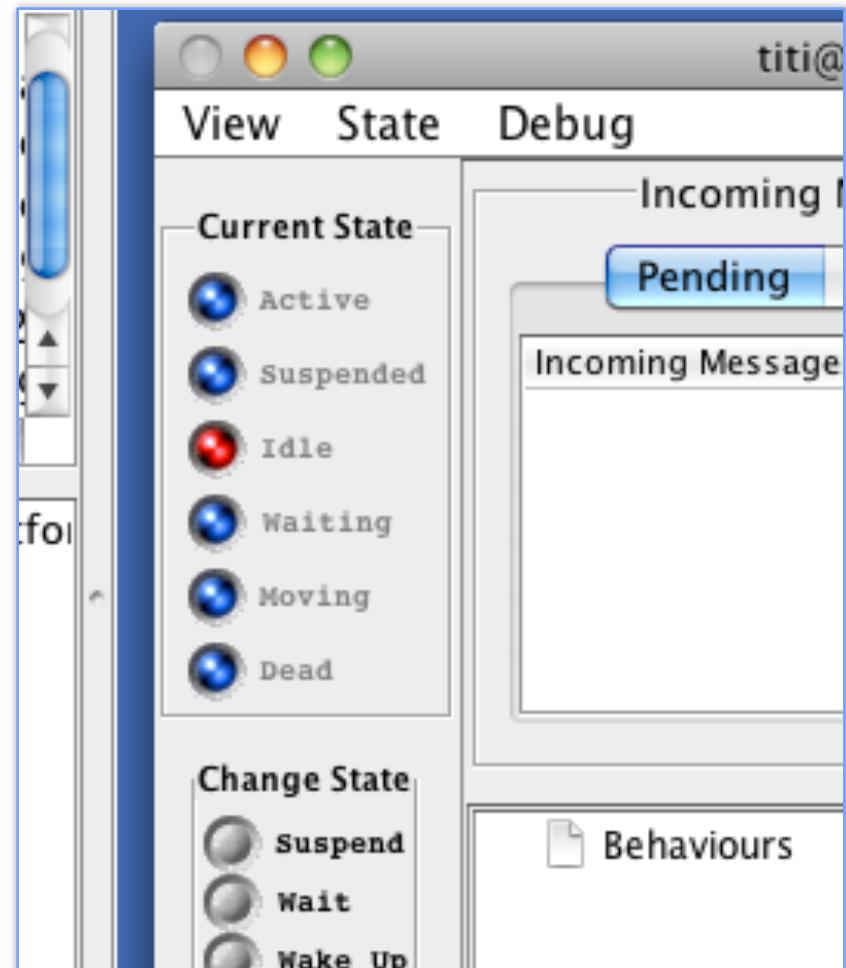
Incoming Messages	Outgoing Messages
Pending	Pending
Received	Sent

The Debug tab displays the current behaviors of the agent. A tree view shows a 'Behaviours' node with several 'SequentialBehaviour' instances, each containing multiple 'SingleStepBehaviour' instances. The details pane shows the selected behavior's name, class, and kind:

Name: SingleStepBehaviour
Class: MyFirstPackage.ComplexBehaviourAgent\$5
Kind: OneShotBehaviour

L'état endormi-inactif (Idle)

- L'agent attend un message lorsque la fonction `block()` est utilisée,
- Lorsque la liste des comportements de l'agent est vide
- Différent de l'état d'attente.



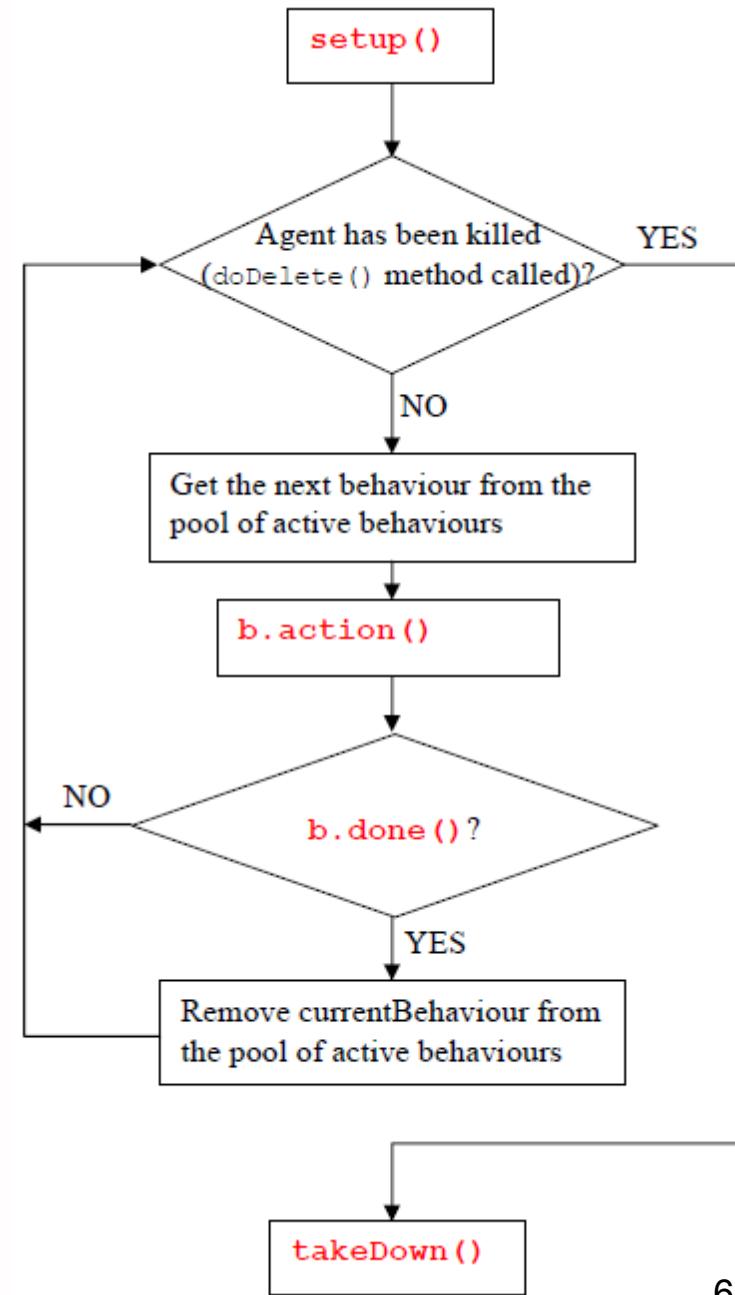
Le « process-Agent »

- Initialisations
- Ajout des comportements initiaux

Cycle de vie de l'agent

« Vidange »

méthodes _à _implémenter ()



Le « process-Agent »

- Initialisations
- Ajout des comportements initiaux

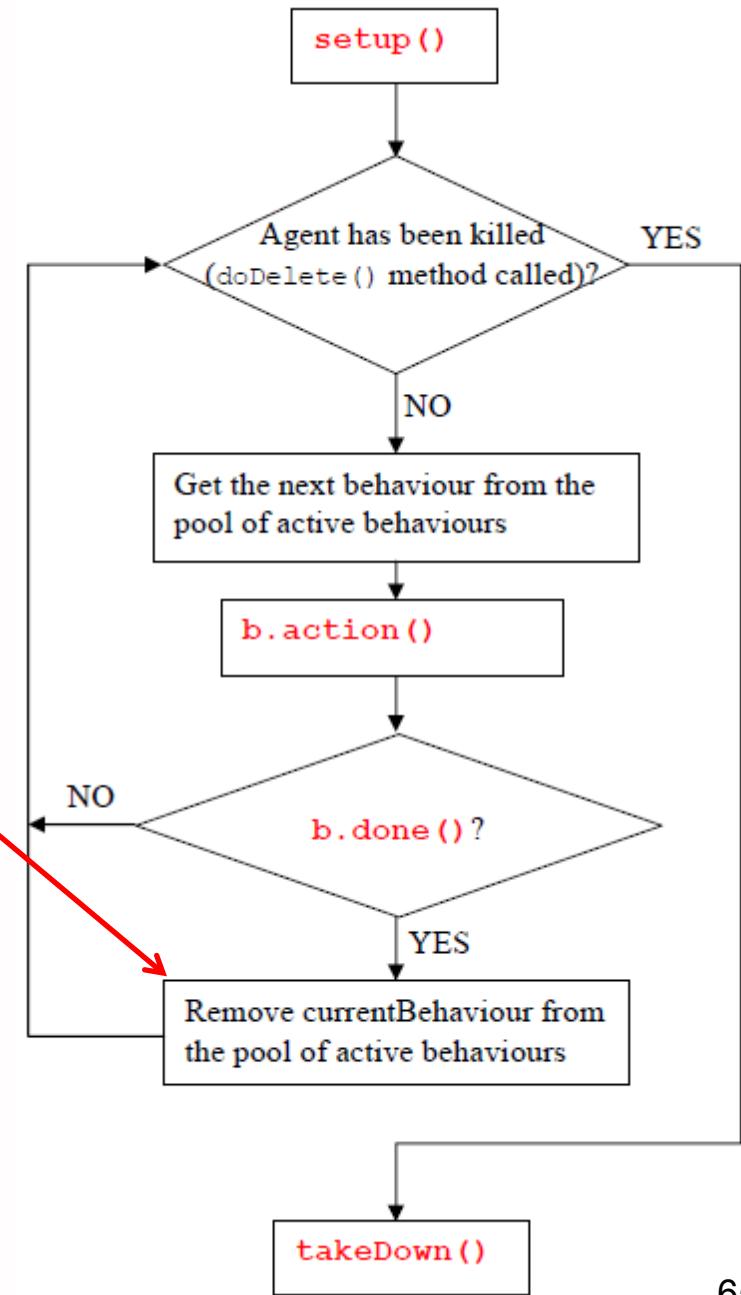


Pas de comportements en stock → l'agent « dort » en attendant d'autres comportements (Opt CPU)



« Vidange »

méthodes _à _implémenter ()



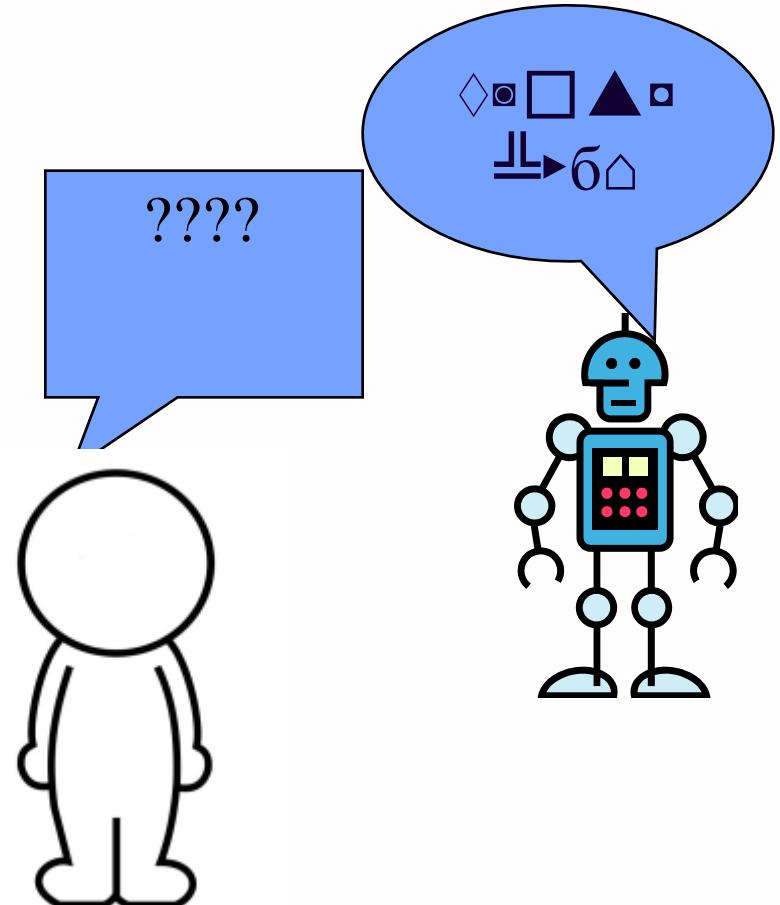
TP N° 2

Implémentez vos propres optimiseurs :
Comportements intégrez les mét-heuristique
intégrez les développées dans le comportement
développées dans le comportement
d'un agent et tester

La communication

Langages de communication inter-agents

- Description syntaxique et sémantique exhaustive des actes de communication,
- Support d'un langage de représentation des connaissances
- Quelques langages:
 - KQML (Knowledge and Query Manipulation Language)
 - FIPA-ACL (Foundation of Intelligent Physical Agents-Agent Communication Language)
 - KIF (Knowledge Interchange Format)



Le message FIPA-ACL

- Un message :
 - **Sender** : celui qui envoie le message
 - **Receivers** : les receveurs du message
 - **Communicative** : le type du message
 - Agree : accord
 - Cancel : annulation
 - Inform : information
 - ...
- Mais aussi :
 - **Content** : le contenu du message
 - **Ontologie** : « le jargon »
 - ...

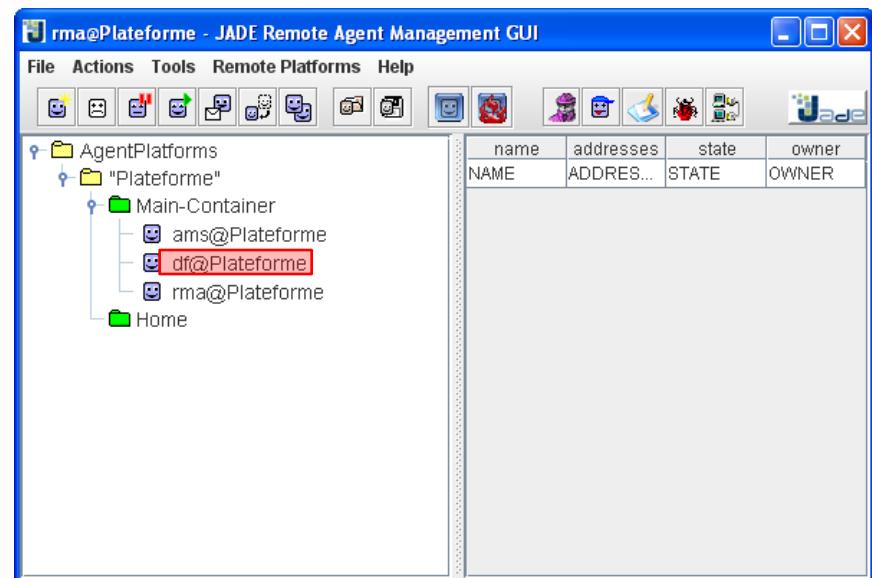
JADE : 2 types de communication

Communication directe



L'agent X connaît l'identité de l'agent Y qu'il veut contacter

Service de pages jaunes



L'agent X cherche des « fournisseurs » d'un service en particulier.

Communication directe A↔B

```
AgentB :  
import jade.core.AID;  
import jade.core.Agent;  
import jade.core.behaviours.*;  
import jade.lang.acl.*;  
  
public class AgentB extends Agent{  
  
    protected void setup() {  
        addBehaviour(new TickerBehaviour(this, 1000) {  
            protected void onTick() {  
                System.out.println("Agent "+myAgent.getLocalName()+" :  
tick="+getTickCount());  
                ACLMessage msg=new ACLMessage(ACLMessage.AGREE);  
                msg.addReceiver(new AID("A@Plateforme", AID.ISGUID));  
                myAgent.send(msg);  
            }  
        });  
    }  
}
```

Communication directe A↔B

AgentA :

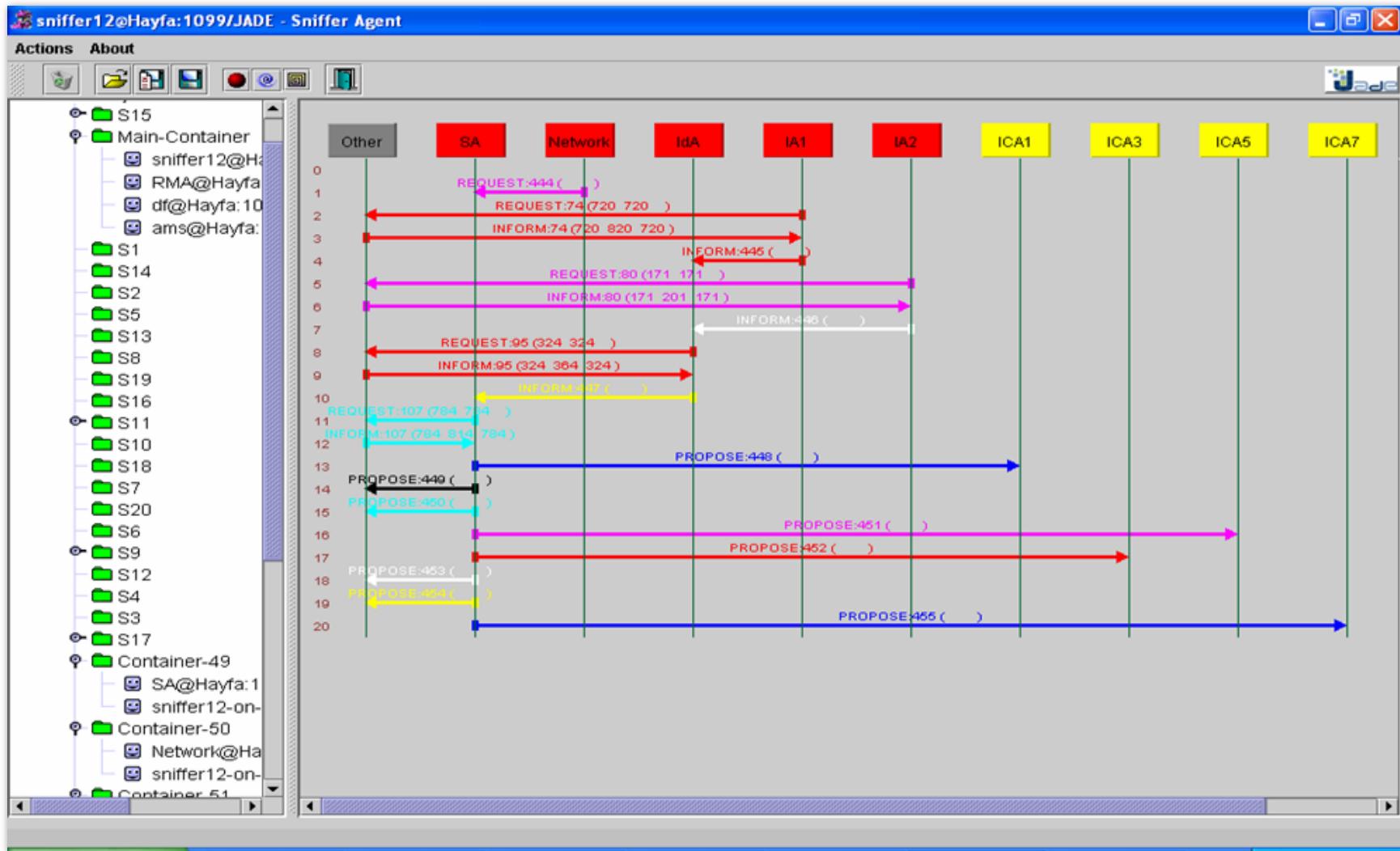
```
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

public class AgentA extends Agent{
    protected void setup() {
        addBehaviour(new CyclicBehaviour(this) {
            public void action() {
                ACLMessage msg = myAgent.receive();
                if (msg != null) {
                    System.out.println("Received message from agent
"+msg.getSender().getName());
                    ACLMessage reply = msg.createReply();

                    reply.setPerformative(ACLMessage.INFORM);
                    reply.setContent("OK");
                    myAgent.send(reply);
                }else {
                    block();
                }
            }
        });
    }
}
```

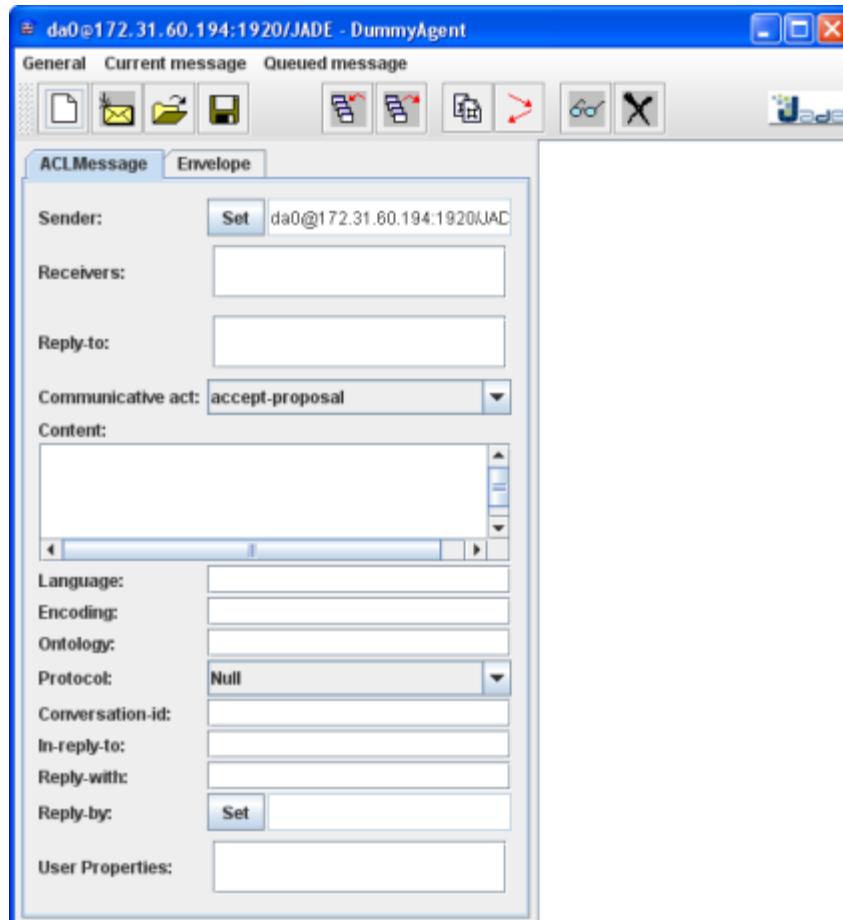
Outil graphique JADE

Sniffer: échange de messages



Outil graphique JADE

Dummy Agent

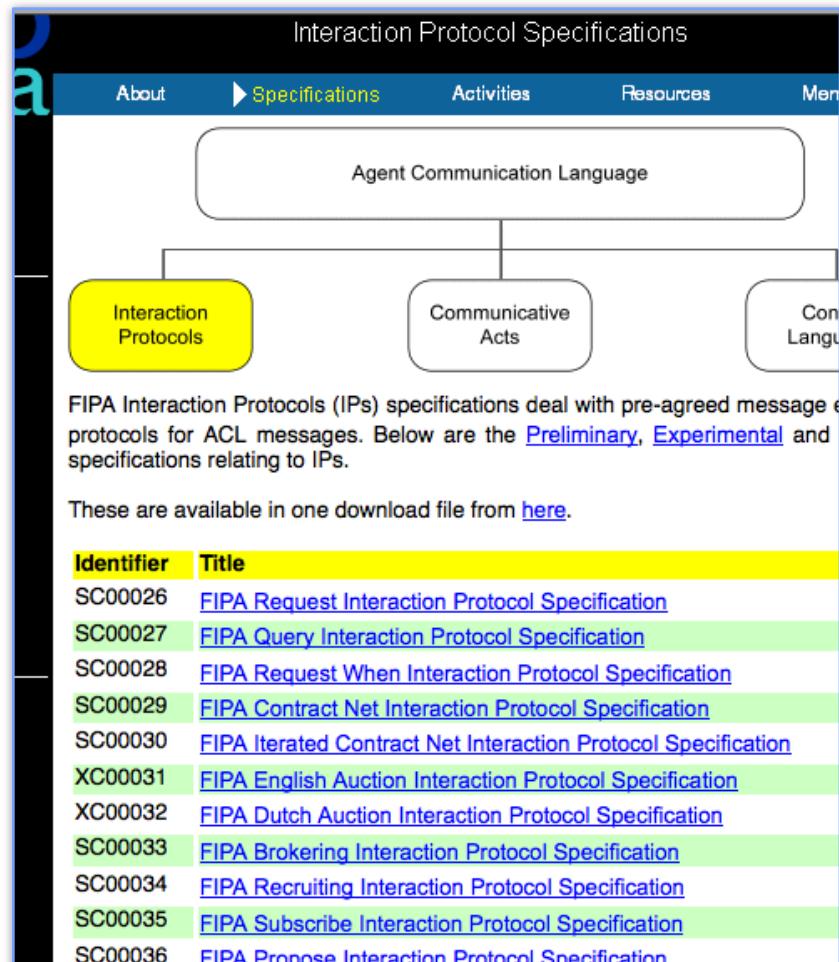


- Envoi des ACL-messages aux agents
- Reçoit et examine des ACL-messages des agents
- Permet, à partir d'une interface graphique de:
 - éditer, écrire et envoyer des messages ACL vers d'autres agents,
 - recevoir et lire des messages des autres agents,
 - sauvegarder les messages.

Protocoles d'interaction

Les protocoles : types de conversations

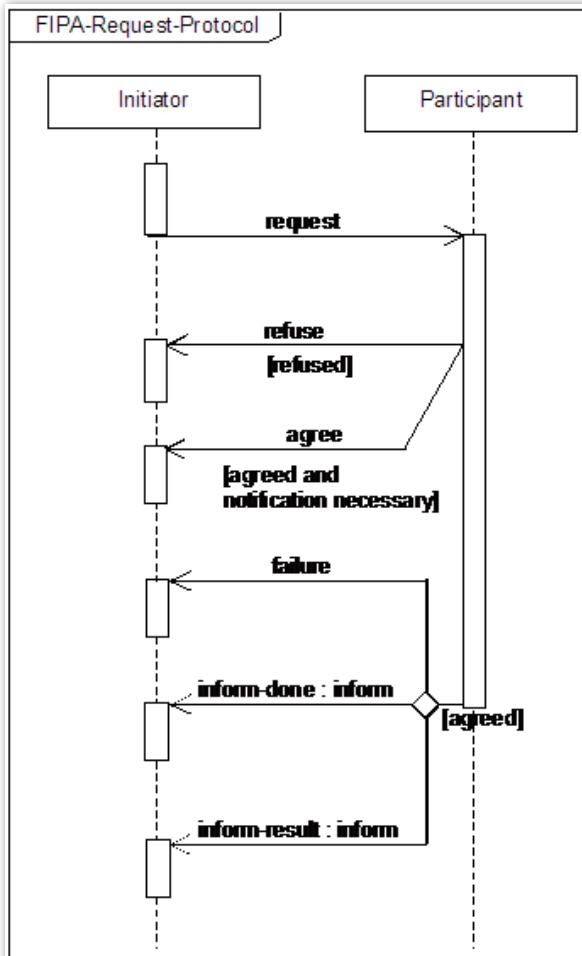
- Séquence de messages conformes à des règles plus ou moins génériques.
- Des modèles « standards » existent et peuvent être adaptés.
- Dépend du type de l'Organisation Multi-Agent (OMA).
 - Les liens entre les agents...



Protocoles : Rôles et comportements

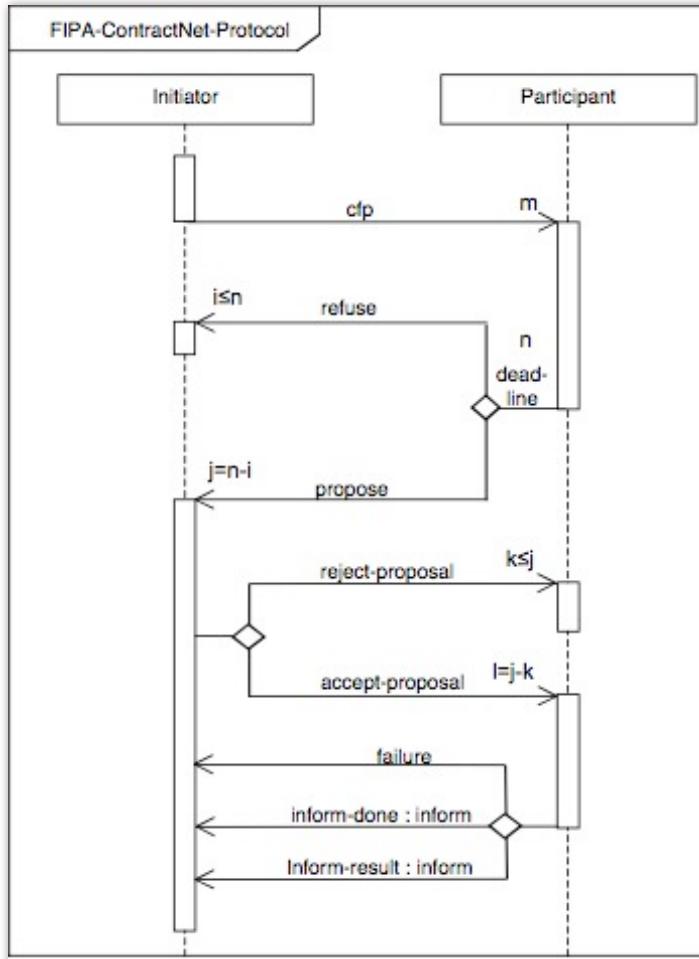
- Rôles :
 - ***Initiateur*** : l'agent qui commence la conversation
 - ***Participant*** : l'agent qui s'engage dans une conversation après avoir été sollicité
- Jade fournit des classes de comportements pour la plupart des Protocoles d'Interaction FIPA (PIFIPA)
- **PIFIPA** :
<http://www.fipa.org/repository/ips.php3>
- Package : **jade.proto**

FIPA-request interaction protocol



1. $I \rightarrow P$: requête
 - refus
 - accord
2. $P \rightarrow I$:
 - échec
 - fait
 - fait + résultat
3. $I \rightarrow P$:

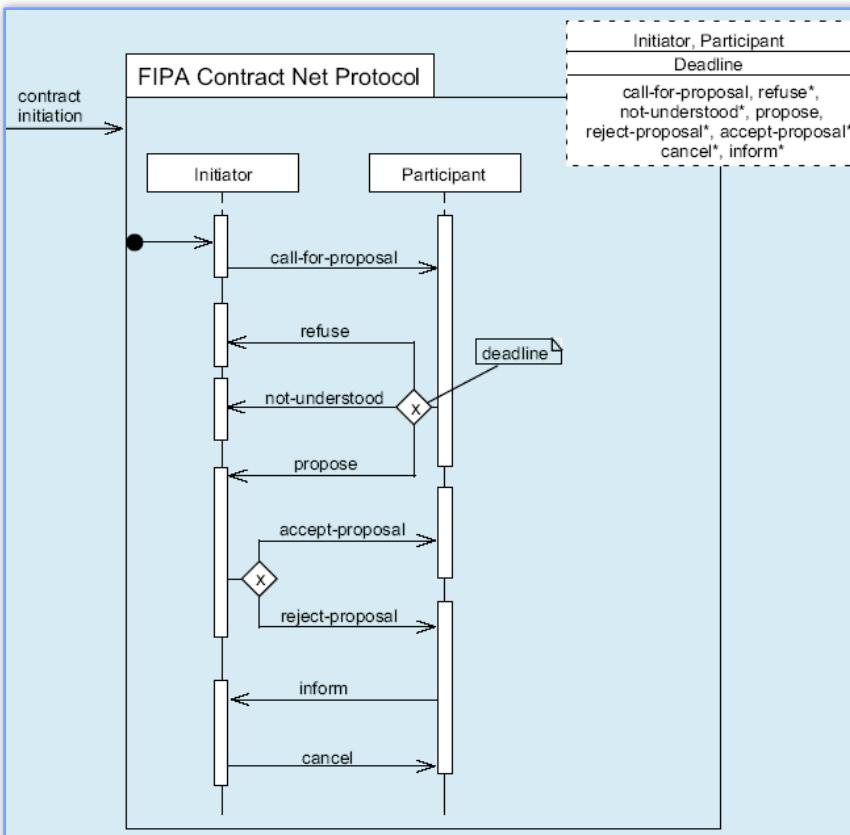
FIPA-Contract Net interaction Protocol (CNP)



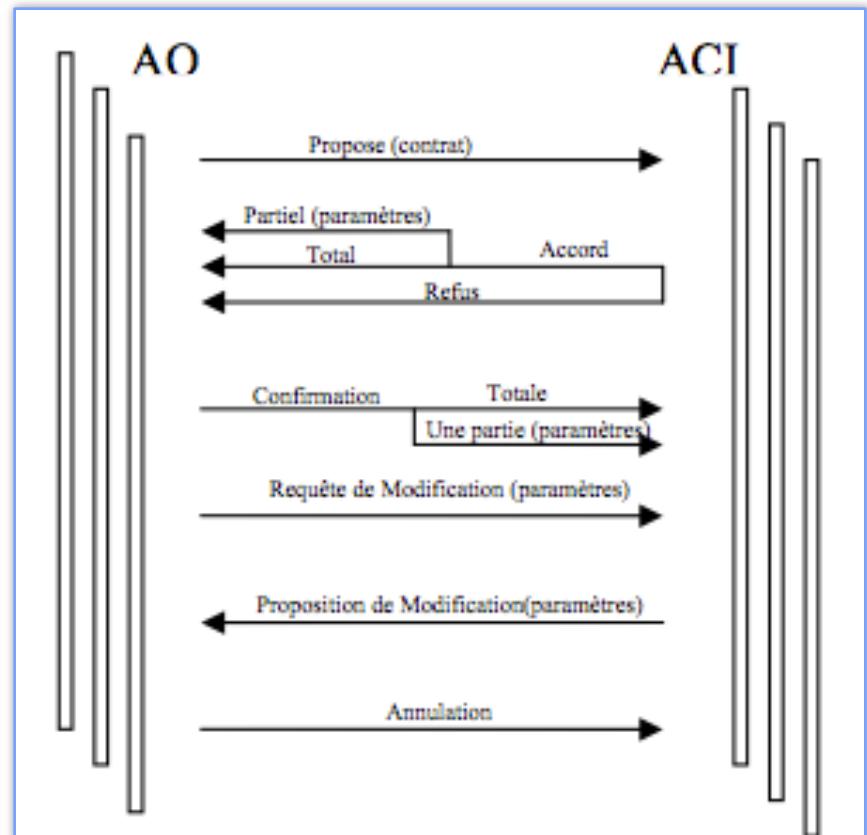
1. $I \rightarrow P : m$ cfp (call for proposals)
 - $i \leq n$ refus
 - $j=n-i$ propositions
2. $P \rightarrow I : n$ reçues (deadline)
 - $i \leq n$ refus
 - $j=n-i$ propositions
3. $I \rightarrow P :$
 - $k \leq j$ rejet
 - $I = j-k$ accords
4. $P \rightarrow I :$
 - Échec
 - Fait
 - Fait + résultat

CNP : autres variantes

CNP



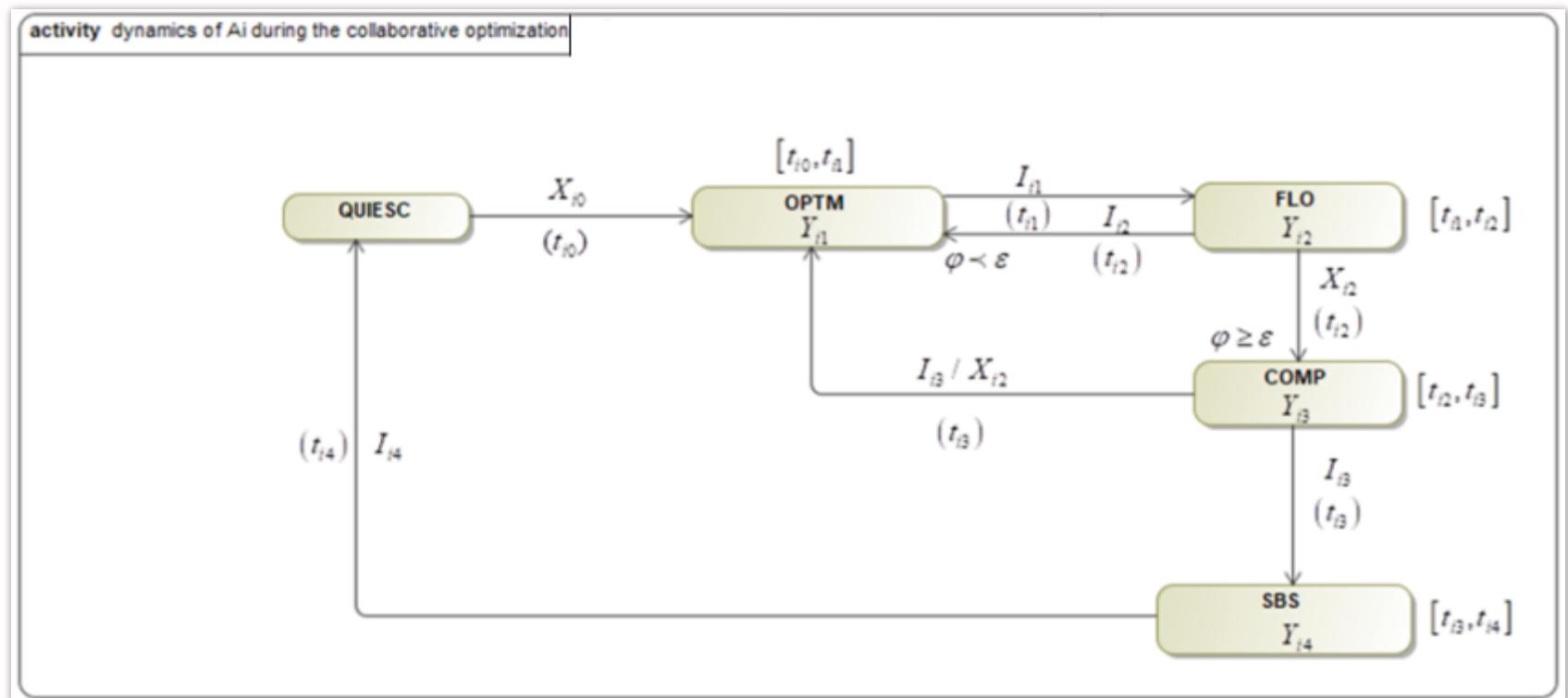
CNP évolué



1 initiateur – n participants

Zgaya, 2007

The COP Collaborative Optimization Protocol



Agent : Etats possibles

Status	
QUIESC	the quiescent status at $t < t_{i0}$, with t_0 the start-up time of the multi-agent system
OPTM	the optimization process status
FLO	the fall in a local optimum status
COMP	the comparison process status
SBS	the sending best solution status

The state of an agent evolves according of the triggering of an internal or external event through the (internal or external) transition functions (δ_{int} or δ_{ext}).

Decision laws

DL		Status
DL0	The agent identifies the COP and its features and decides to participate to the collaborative optimization process	OPTM
DL1	the agent converges to an effective solution and decides whether it was trapped in a local optimum solution or not	→ FLO
DL2	the agent decides returns back to the OPTM status in order to change the configuration parameters because of the optimization process stagnation	→ OPTM
DL3	the agent perceives that its found solution is the best so it decides to begin a comparison process with the solution found by the other agents	→ COMP
DL4/DL 5	the agent compares its solution with those of the other collaborative agents (X_{i2}). DL4: if the best solution is the one found by the other agents (I_{i3}), it uses it in its optimization process (OPTM status). Otherwise, DL5: the best solution is the one found by itself, so the agent goes to the SBS status.	→ SBS
DL6	The agent decides to send its best solution to the other collaborative agents and goes to the QUIESC status.	→ QUIESC

The DL table

Q			I	X	S
s	e	DL			
QUIESC	t_{i0}	DL0		X_{i0} (contains the COP and its features)	OPTM
OPTM	t_{i1}	DL1	I_{i1}		FLO
FLO	t_{i2}	DL2	I_{i2}		OPTM
FLO	t_{i2}	DL3		X_{i2} (contains the solutions of the other collaborative agents)	COMP
COMP	t_{i3}	DL4	I_{i3}	X_{i2}	OPTM
COMP	t_{i3}	DL5	I_{i3}	X_{i2}	SBS
SBS	T_{i4}	DL6	I_{i3}		QUIESC

Fil Rouge : apprentissage

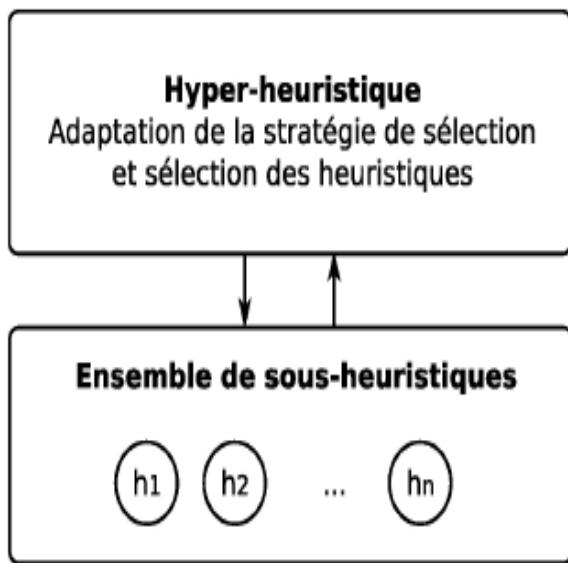
Application de l'apprentissage par renforcement à l'optimisation collaborative au PVC (Fil Rouge)

- L ' idée : chaque agent est capable d' effectuer indépendamment une recherche des solutions et adapte son comportement par apprentissage.
- Intégrer des mécanismes d' apprentissage qui adaptent dynamiquement la stratégie des recherche afin de rendre la méthodes plus robuste face à différentes instances du PVC.
- L ' agent choisi une métaheuristique (tabou, RS,AG) en fonction du temps d' exécution moyen de cette dernière.
- L ' agent choisi une métaheuristique (tabou, RS,AG) en fonction de la différence de fitness apportée en moyenne par la métaheuristique lors de ses précédentes applications.

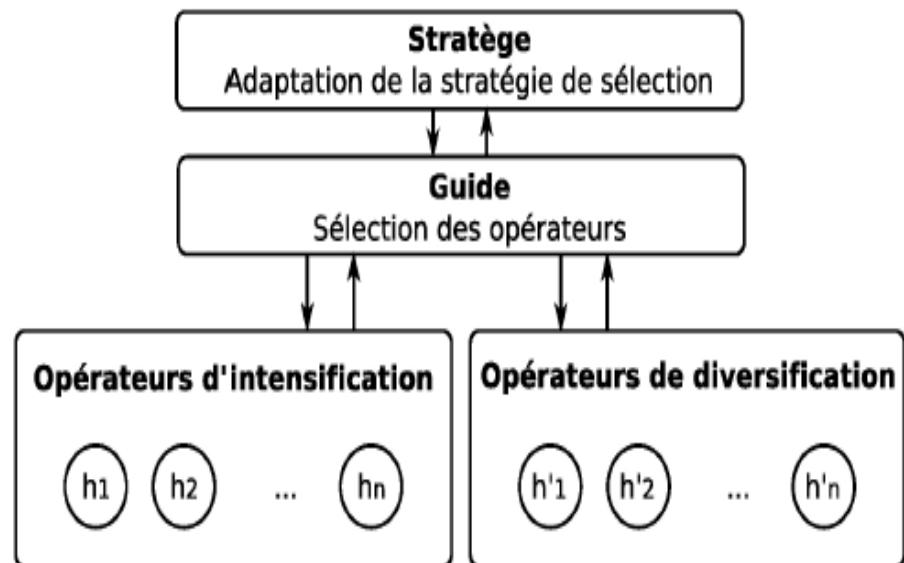
Le mécanisme de sélection est indépendant du problème

Optimisation collaborative & Apprentissage

Approche hyper-heuristique



*Approche **OPTIM + SMA + Apprentissage***



Exemple d'un Dataset à construire pour l'apprentissage renforcé

Méthode	Paramètres de convergence					Stratégies	
						Temps d'exécution moyen	Ecart de fitness
RS	h1	h2	h3	h4...	hn		
RS	h1	h2	h3	h4...	hn		
RS	h1	h2	h3	h4...	hn		
Tabou	h'1	h'2	h'3	h'4...	h'n		
Tabou	h'1	h'2	h'3	h'4...	h'n		
Tabou	h'1	h'2	h'3	h'4...	h'n		
AG	h"1	h"2	h"3	h"4...	h"n		
AG	h"1	h"2	h"3	h"4...	h"n		
AG	h"1	h"2	h"3	h"4...	h"n		

Résultats sur le benchmark

- ***Optimisation sans SMA & sans apprentissage***

Taille PVC	RS			Tabou			AG		
	Moyenne	Ecart	Durée	Moyenne	Ecart	Durée	Moyenne	Ecart	Durée



Benchmark

- ***Optimisation avec SMA & sans apprentissage***

Taille PVC	SMA: Stratégie par coopération (2 par 2)			SMA: Stratégie par coopération (3 par 3)			SMA: Stratégie par concurrence		
	Moyenne	Ecart	Durée	Moyenne	Ecart	Durée	Moyenne	Ecart	Durée

- ***Optimisation avec SMA & avec apprentissage***

Taille PVC	Stratégie d'apprentissage				Stratégie d'apprentissage			
	TempsExécutionMoyen	MéthodeMétaheuristiqueChoisie	Paramètres	EcartFitness	MéthodeMétaheuristiqueChoisie	Paramètres		