

Par : Youssfi Mohamed

Laboratoire : Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

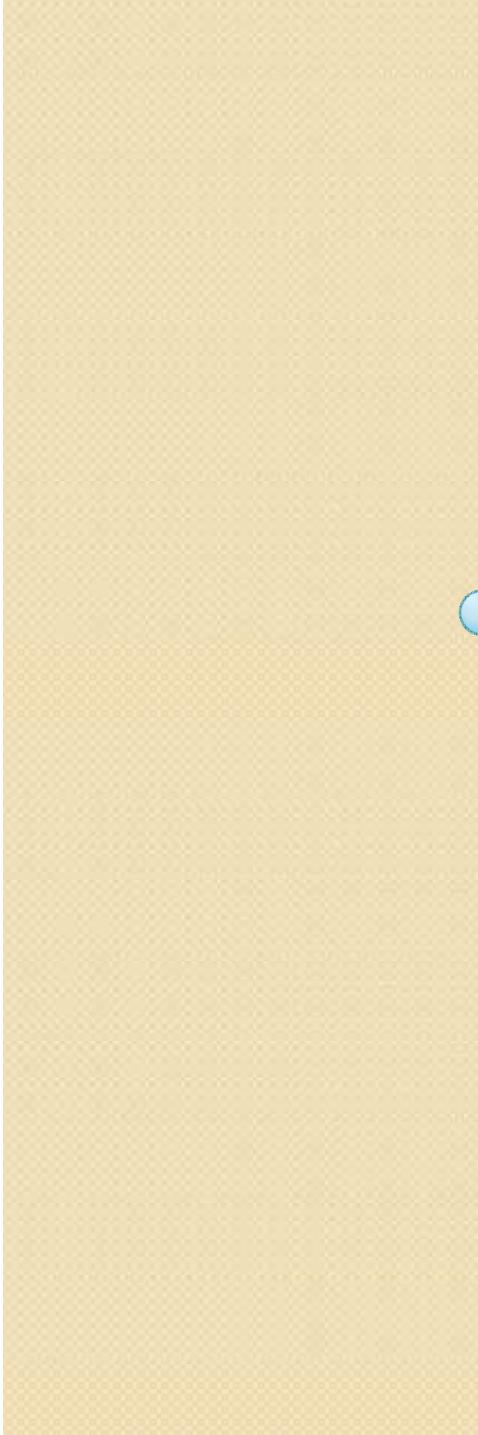
ENSET Mohammedia, Université Hassan II Casablanca,

Email : [med@youssfi.net](mailto:med@youssfi.net)



# Plan

- **Concepts fondamentaux des SMA**
  - Influence historique :
  - Agent et Systèmes multi agents
  - Interactions des les SMA
  - Architectures parallèles et SMA
  - Applications des SMA
  - Plateformes SMA
- **Mise en œuvre des SMA avec JADE**
  - Architecture de JADE (AMS,ACC et DF )
  - Configuration et démarrage d'un conteneur de JADE
  - Premier Agent
  - Cycle de vie des agents
  - Comportements des agents Jade
  - Communication entre les agents avec ACL
  - Mobilité des agents
  - Publication et Recherche des services dans agents
  - Application : SMA pour l'automatisation des transactions de vente et d'achats de livres.



- **INFLUENCE  
HISTORIQUE**



# Historique sur les méthodes de programmation



**Langage Machine**

**Assembleur**

Chaque famille de CPU possède son porophore jeu d'instructions

Langage bas niveau

**Programmation Procédurale**

Sous programmes: Procédures, fonctions  
(Basic, Pascal, C, Fortran,...)

**Programmation Orientée Objet**

Objet = Etat+ Comportement + Identité  
Concepts fondamentaux : Objet, classe, Héritage,  
polymorphisme, encapsulation (C++, JAVA, C#,..)

**Programmation Orientée Objet Distribués**

Objets distribués sur plusieurs machines  
Middlewares : (RMI, CORBA, JMS, ...)

**Programmation Orientée composants**

Objets distribués, réutilisables, configurables,  
Interchangeables, évolutifs, mobiles, surveillable à  
chaud : Conteneur (EJB)

**Programmation Orientée Services**

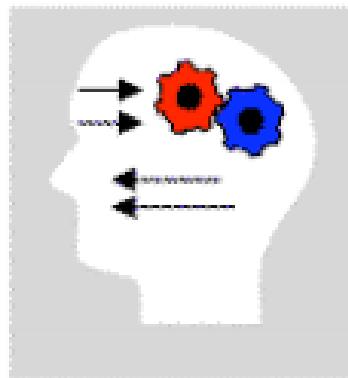
Composant disponibles à d'autres applications  
distantes hétérogènes via des protocoles (http)  
transportant des données: XML, JSON => SOAP  
et REST

**Programmation Orientée Agents**

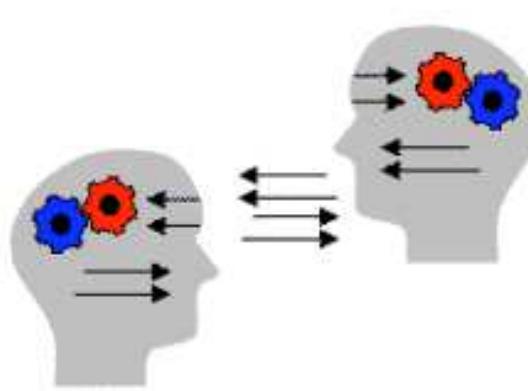
? Service + Intelligence + Apprentissage+ ...

# Intelligence Artificielle Distribuée

- L'**intelligence artificielle** est la « recherche de moyens susceptibles de doter les systèmes informatiques de capacités intellectuelles comparables à celles des êtres humains »
- L'IA s'appuie sur les sciences de l'homme :
  - Psychologie linguistique, Sociologie, Neuro Biologie, Biosphère
- **Intelligence Artificielle Distribuée** =
  - **IA** : Modéliser le savoir des agents (compétence)
  - + **Distribution** : Modéliser leurs interactions (organisation sociale)



Métaphore de l'IA  
Penseur Isolé



Métaphore de l'IAD  
Communauté de penseurs

- 1985 :
  - => IAD
  - => Systèmes Multi Agents



# **AGENT ET SYSTÈMES MULTI AGENTS**



# Systèmes multi-agents

- Un **Agent** est une entité
  - Qui **agit** d'une façon **autonome**
  - pour **atteindre les objectifs** pour lesquels il a été conçu.
  - Peut **communiquer** avec d'autres agents
  - Doté de capacités semblables aux êtres vivants
- Un agent peut être un **processus**, un **robot**, un être **humain**, etc...
- Un **système multi-agent (SMA)** est un système distribué :
  - Composé d'un ensemble d'**agents distribués**,
  - **Situés** dans un certain environnement
  - et **Interagissant** selon certaines **organisations**.
- Un **SMA** permet de résoudre des problèmes complexes en exploitant **l'intelligence collective** des agents qui le compose



# Caractéristiques d'un Agent

- Dans un SMA, on s'intéresse aux phénomènes :
  - Apprentissage
  - Raisonnement
  - Perception
  - Migration
  - Mémorisation
  - Interaction
  - Intention
  - Commandes et coordination motrice
  - Emotion
  - Conscience
  - Ethique

# Concepts fondamentaux des agents

- AGENT = BODY + MIND
- BODY : Dimension physicaliste

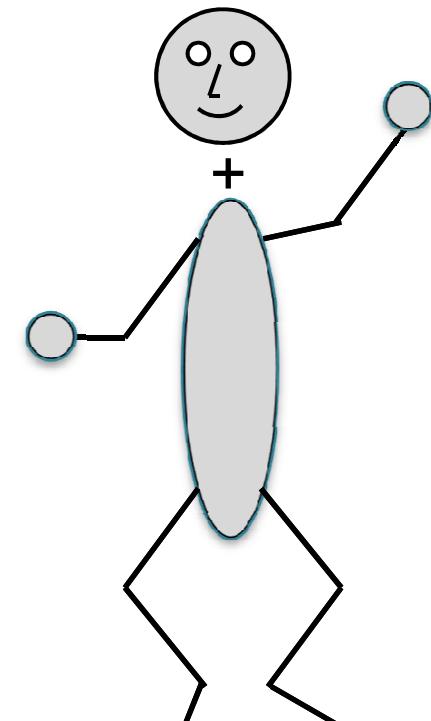
«Un agent est une entité autonome située dans un environnement ouvert »

- Situation
- Persistance
- Mobilité

- MIND : Dimension épistémique

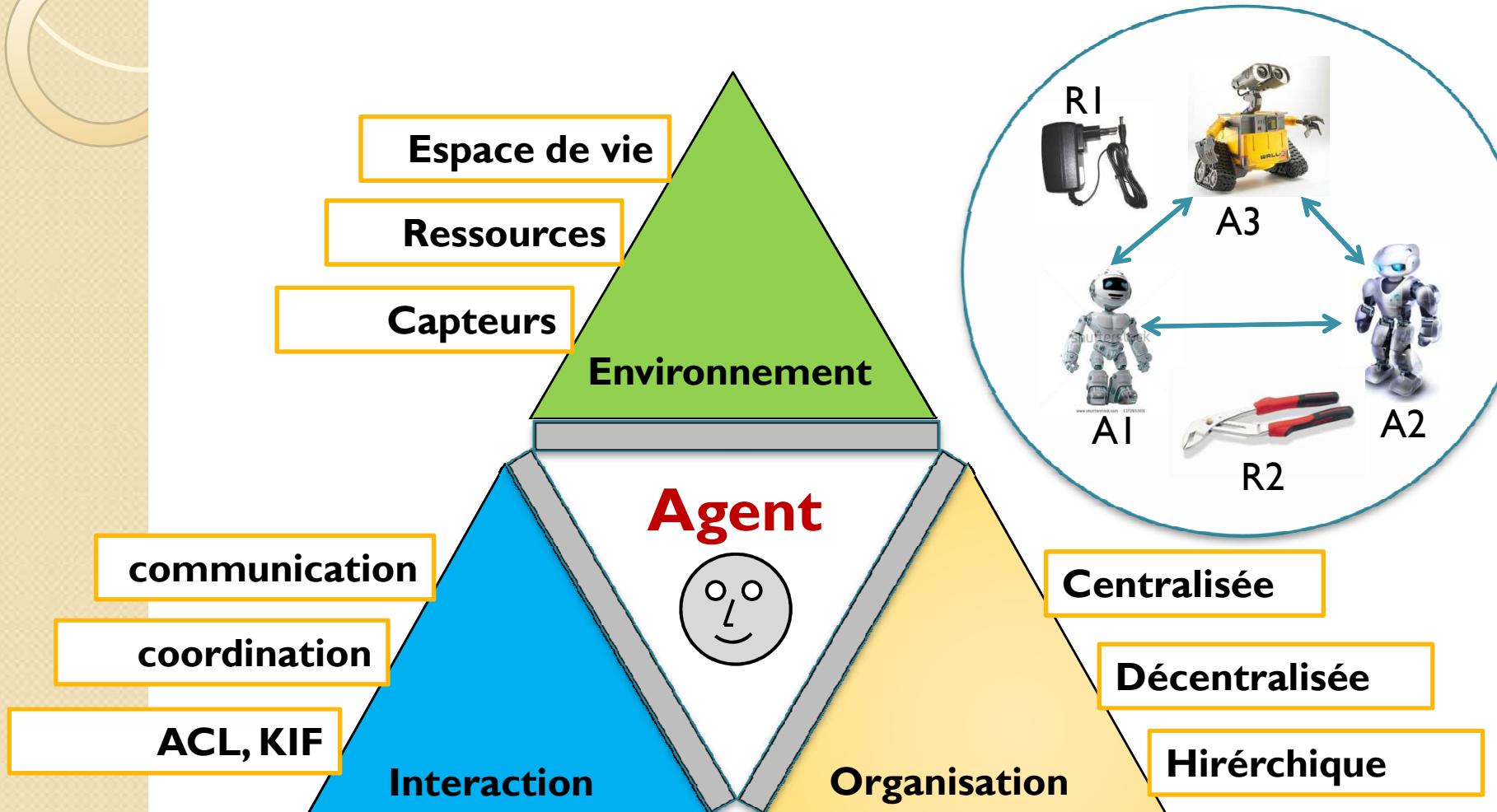
«Un agent est une entité en interaction avec d'autres agents dans un champ social»

- Population
- Interaction
- Intention
- Apprentissage
- Raisonnement



# Les trois dimensions d'un agent

- Un agent vit dans un **environnement** en **interagit**, dans un champ social, avec d'autres agents selon une **organisation**



ACL : Agent Communication Language

KIF : Knowledge Interchange Format

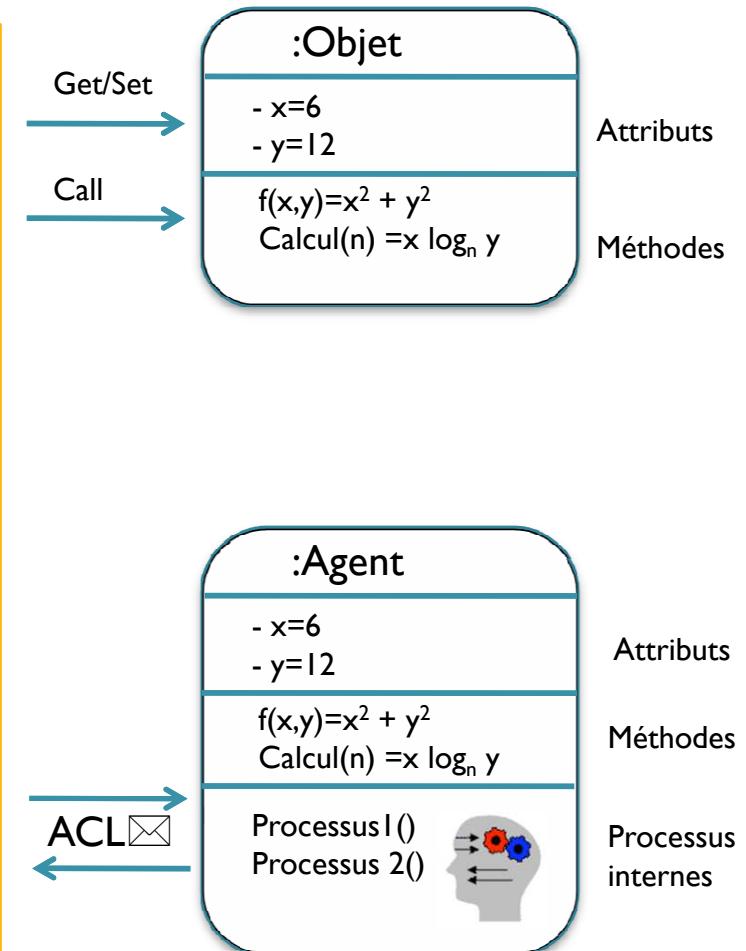
# Objet versus Agent

- **Un objet est réactif**

- Un objet est une entité passive (ou réactive).
- Si personne ne demande la valeur d'un attribut ou n'active une méthode de l'objet, alors il ne se passe rien.

- **Un agent est Proactif**

- Un agent possède, en plus des attributs et méthodes, des processus internes qui fonctionnent même en l'absence de sollicitations externes.
- Un agent peut donc agir même si personne ne lui demande rien.





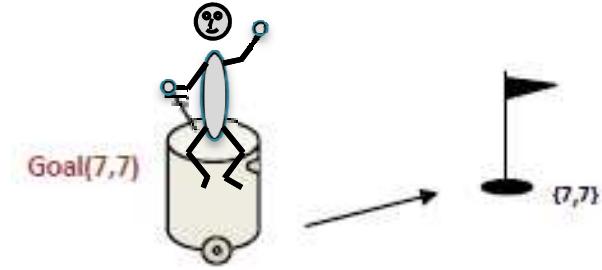
# Objet versus Agent

- **Un agent est persistant :**

- Si un agent est **proactif** c'est d'abord parce qu'il est muni d'au moins un **but** qu'il cherche à satisfaire de manière **persistante** tant que :
    - Il pense que c'est encore possible (pré condition logique)
    - Il possède les ressources pour le faire (pré condition physique)

- **Un agent est adaptatif :**

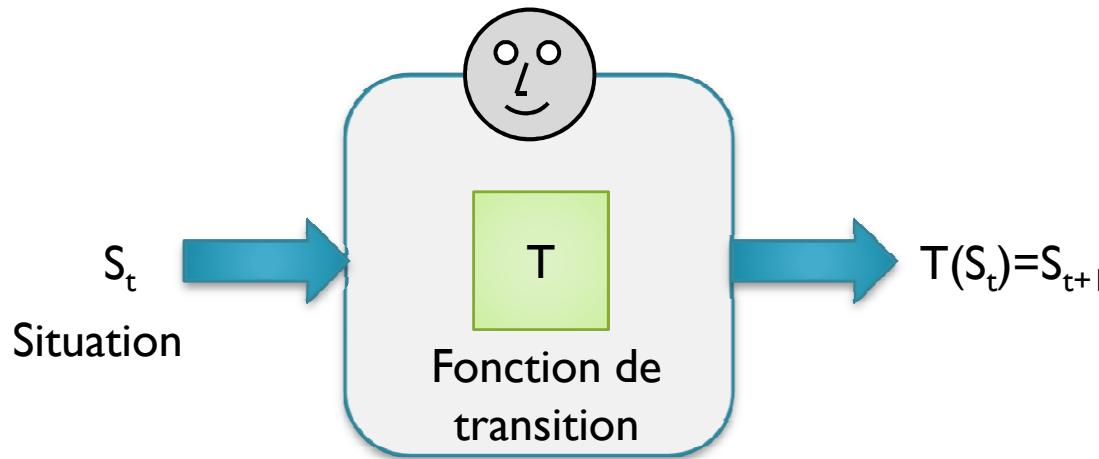
- Face à un environnement perpétuellement changeant, un agent doit constamment modifier le plan qu'il poursuit pour atteindre un but. Pour cela :
    - Il doit, de manière continue, **percevoir** et **évaluer** la situation (contexte) de son action,
    - Construire des représentations en cours même de fonctionnement (c'est-à-dire être capable **d'apprentissage**).
    - **Élaborer des plans dynamiques** qui lancent des processus internes ou au contraire les stoppent.



# Agents réactifs et cognitifs

## • Agents Réactifs

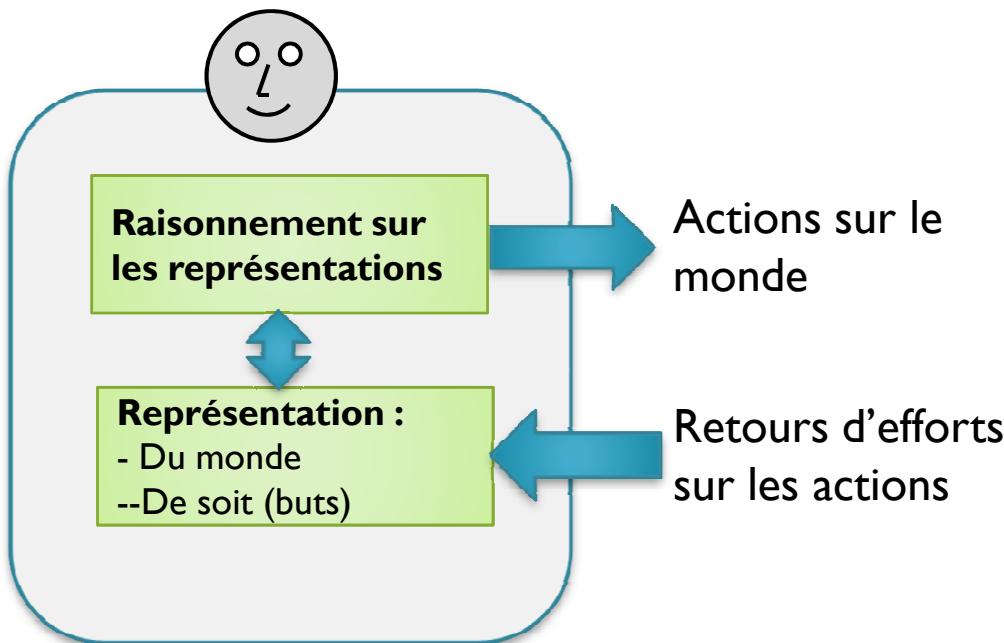
- Les agents réactifs sont issus de la modélisation physicaliste du monde : ils sont assimilés à des fonctions de transition :
- Pas d'anticipation.
- Pas d'apprentissage



# Agents réactifs et cognitifs

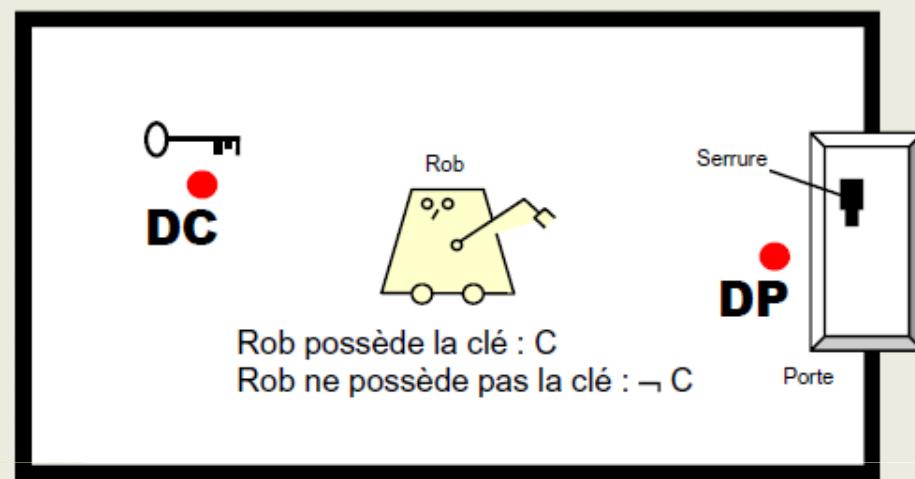
- **Agents Cognitifs**

- Les agents cognitifs sont issus de la modélisation mentaliste du monde : ils sont assimilés à des systèmes experts .



# Exemple d'agent réactif et cognitif

- Problème : « Rob le robot doit sortir de la pièce ».



- DP = Devant la PortePosition
- DC = Devant la Clé
- PO: Porte ouverte :
- PF: Porte Fermée
- PFC : PFFPorte Fermée à clé

## Rob réactif :

- La fonction de transition  $T$  est un ensemble non ordonné de règles de type
  - Si condition alors action
  - qui sont exécutées dans une boucle infinie et non déterministe :

Si DP and PO Alors SORTIR  
Si DP and PF Alors OUVRIR  
Si DP and PFC and C Alors DEVEROILLER  
Si DP and PFC Alors RANDOM-WALK  
Si DC and C Alors RANDOM-WALK  
Si DC and C Alors PRENDRE-CLE

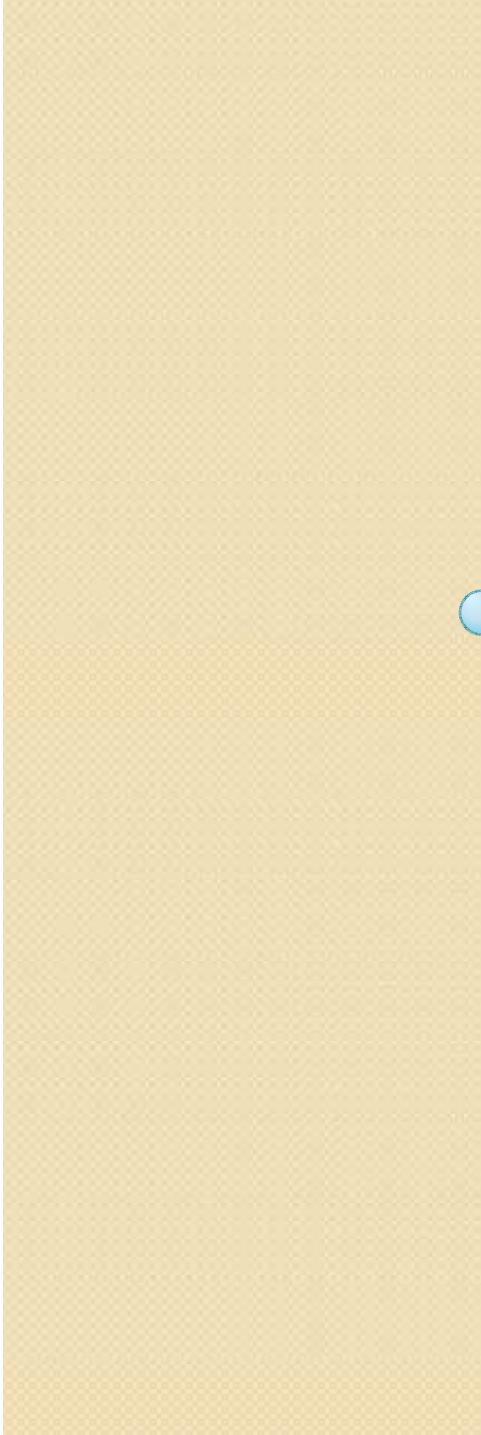
## Rob cognitif:

- L'agent possède un plan  $P$  pour sortir qu'il exécute de manière séquentielle et déterministe quelle que soit sa position de départ et quel que soit l'état initial du monde :

### ALLER-A DP

Si PO Alors SORTIR  
Sinon Si PF Alors OUVRIR;SORTIR

Sinon Si PFC  
Alors Si C Alors DEVEROILLER;OUVRIR;SORTIR  
Sinon ALLER-A DC  
PRENDRE CLE  
ALLERA DP  
DEVEROILLER;OUVRIR;SORTIR



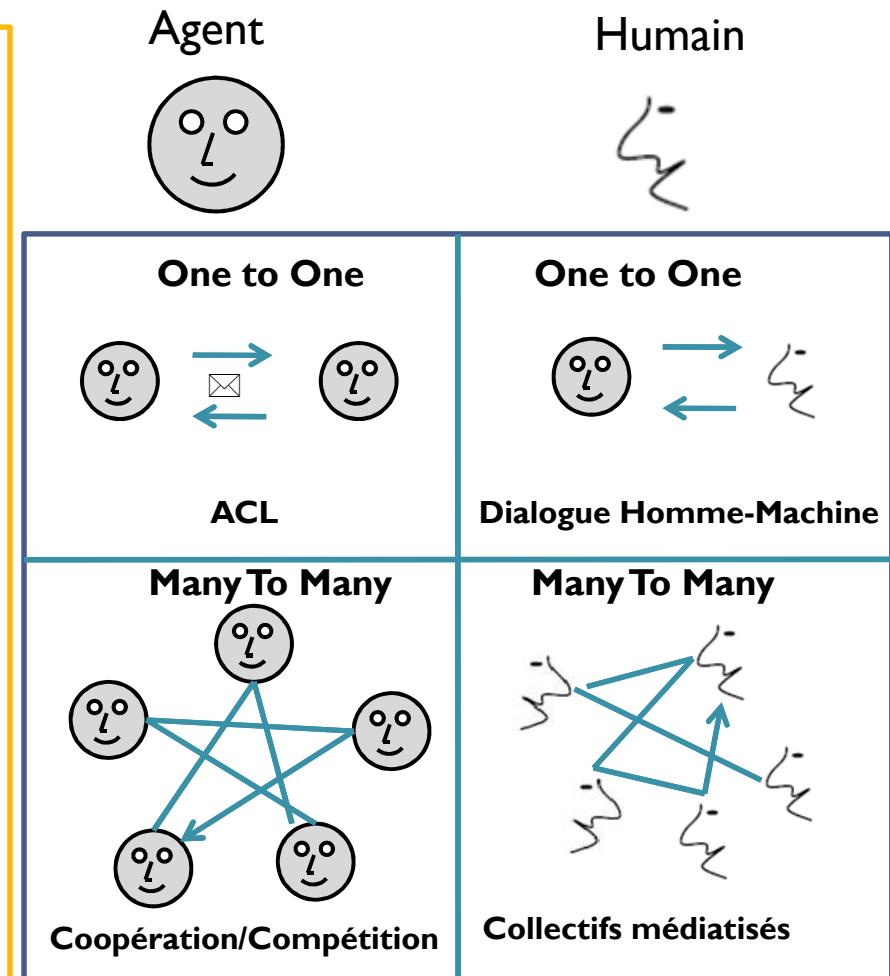
- **INTERACTIONS ENTRE AGENTS**

# Population d'agents

- Variables globales et locales
  - Dans un SMA **rien n'est complètement global**
    - L'environnement étant vaste et ouvert, il n'est pas possible en un lieu donné (par exemple, dans un agent) de stocker toute la représentation du monde.
    - Par contre, un agent peut **se déplacer** ou encore **interagir** avec d'autres agents qui sont dans son voisinage pour explorer l'environnement.
  - Dans un SMA **rien n'est complètement local**
    - Pour un agent donné, toutes ses entités (informations, processus, buts, ...) sont locales mais elles restent accessibles à **l'introspection** par d'autres agents.
    - Le moyen d'accéder à cette information passe par les interactions entre les agents.

# Modèles d'interaction

- **1. Interaction Agent ↔ Agent**
  - Niveau **communication**
    - Transactions : **Agent Communication Languages (ACL)**
  - Niveau **connaissance (Knowledge Level)**
    - Echange de connaissances : **KIF, XML**
    - Données sémantiques : **RDF, OWL**
- **2. Interaction Humain ↔ Agent**
  - Interactions langagières : Reconnaissance vocale
- **3. Interaction Multi-agents**
  - Résolution distribuée de problèmes (IAD) : coopération, conflits, négociation, ...
- **4. Interaction Multi-humains**
  - Modalité langagière : chat, mail, forums, visio-conférence, ...



**ACL** : Agent Communication Language

**KIF** : Knowledge Interchange Format

**XML** : eXtensible Markup Language

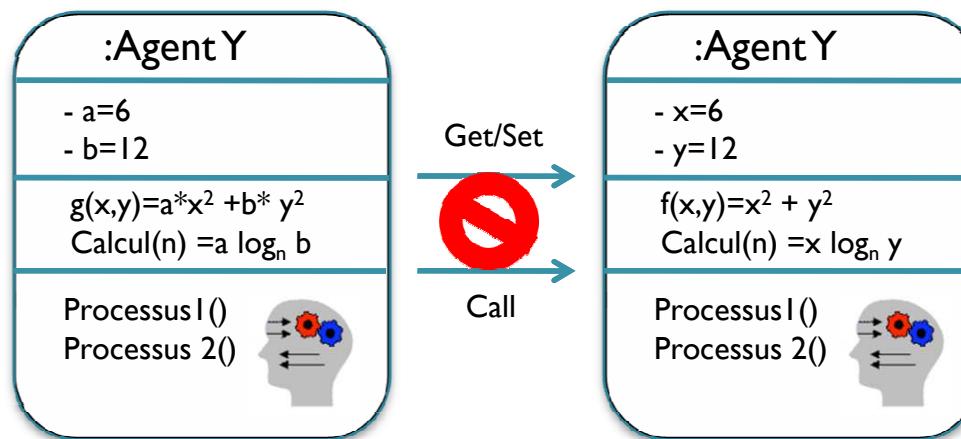
**OWL** : Ontology Web Language

**RDF** : Resource Description Framework

# Typologie des interactions : Interaction Directe

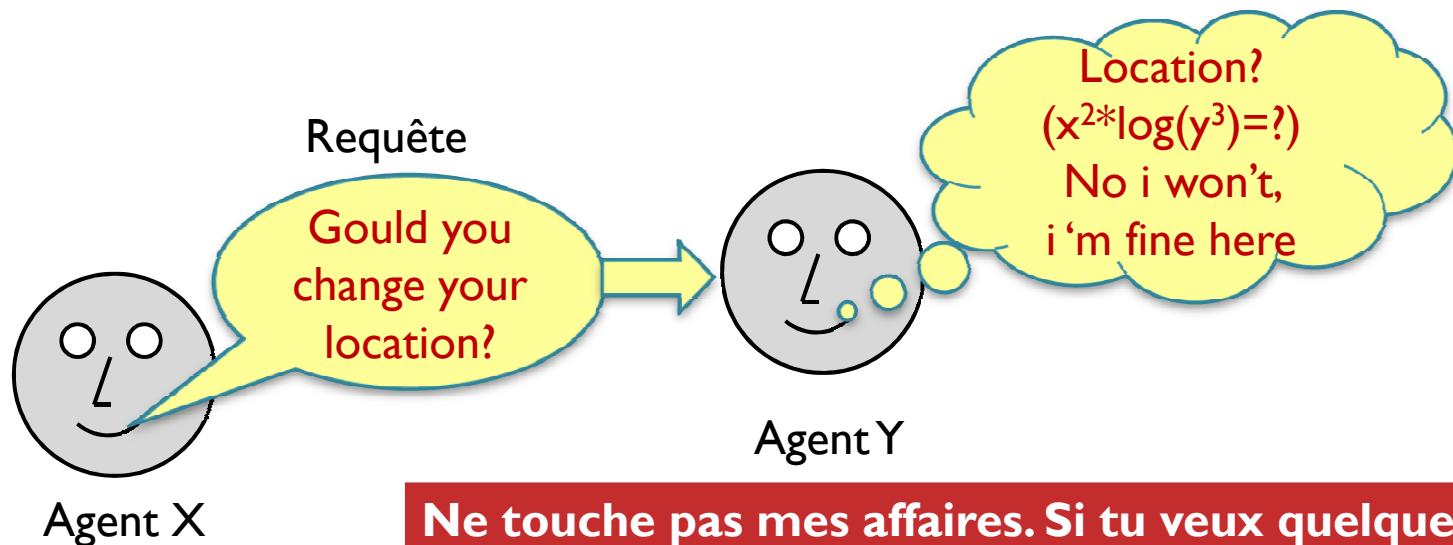
- Action directe (**interdite**) 

- Un agent ne peut pas agir directement sur l'état et le comportement d'un autre Agent.
- L'interaction des agents doit passer par l'envoi de message **ACL**



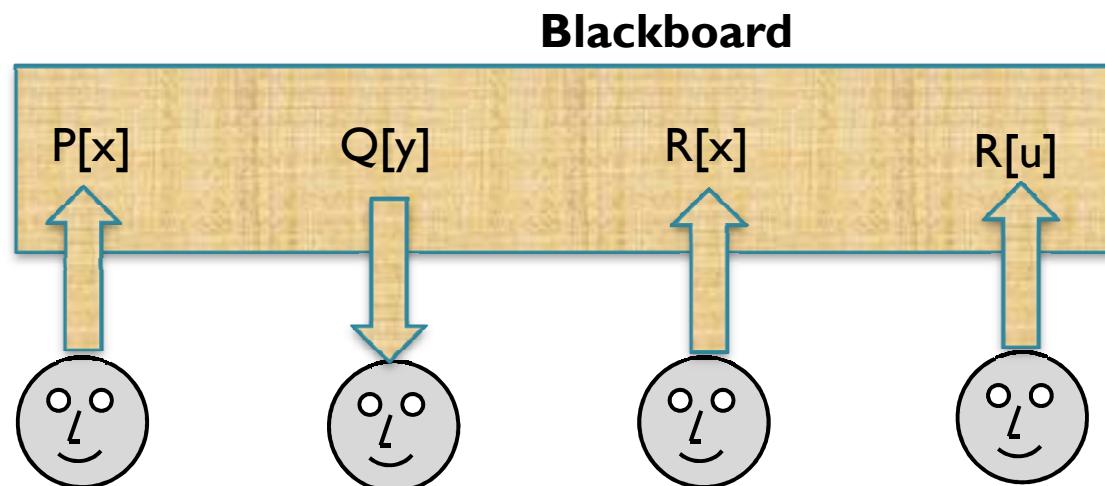
## Typologie des interactions : Interaction Directe

- Requête (formelle ou langagière)
  - L'agent envoie une requête à un interlocuteur qui est un autre agent ou à un humain de son environnement
  - L'interlocuteur interprète cette requête et la satisfait ou non en fonction de sa propre subjectivité (état physique et mental)



## Typologie des interactions : Interaction Indirecte

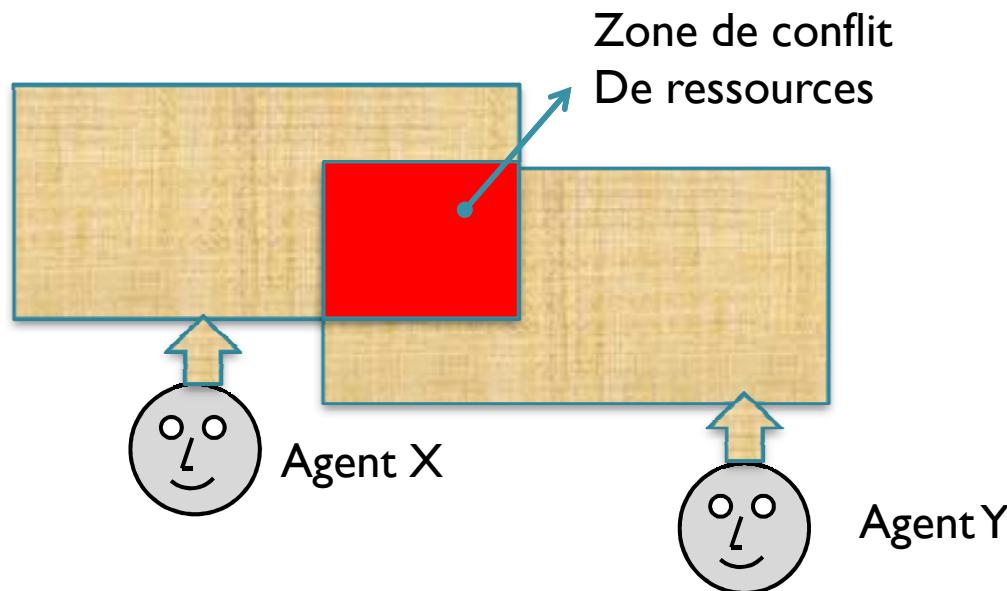
- **Blackboard (base de connaissances)**
  - Plusieurs agents déposent et recueillent des objets ou des informations dans une partie de l'environnement prévue à cet effet.
  - Cette partie commune est appelée « **Blackboard** »



## Typologie des interactions : Interaction InDirecte

- Partage de ressources (**Stigmergie**)

- Les modèles de population animales par exemple sont fondés sur une compétition pour une quantité de ressources à partager qui est fixée : **la ressource sert alors de médiateur entre les agents.**
- Les fourmis par exemple communiquent en déposant des **phéromones** derrière elles, pour que d'autres fourmis puissent suivre la piste jusqu'à la nourriture ou la colonie suivant les besoins, ce qui constitue un système stigmergique



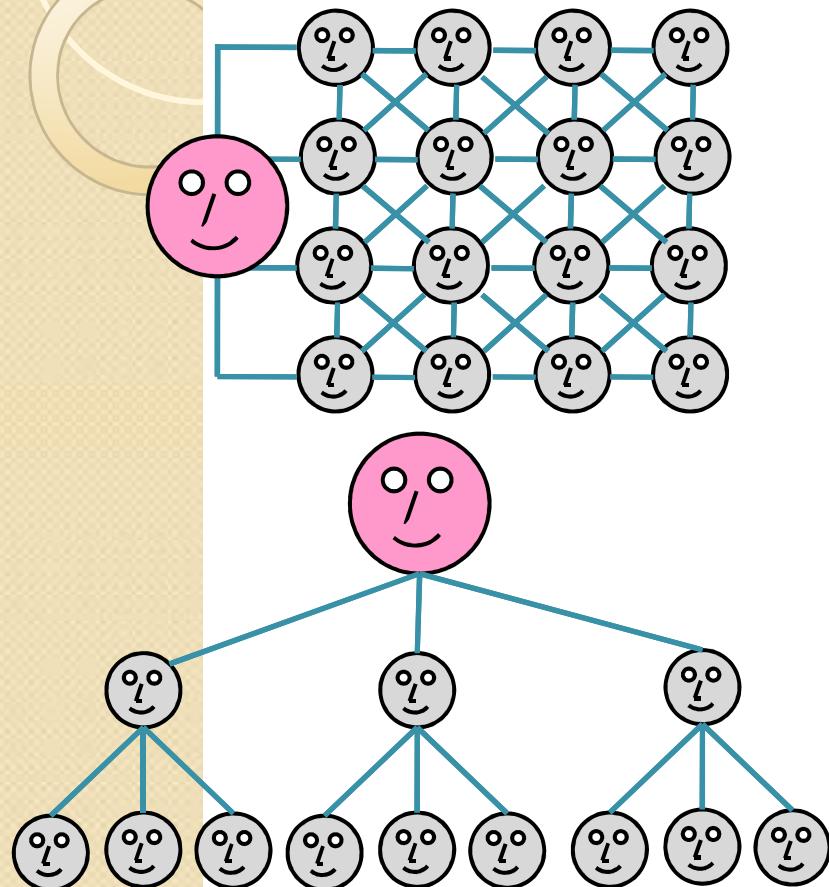
# Manifestation de l'Intention

- L'Intention dans la Communication
  - Les agents interagissent au moyen de Langages de Communication Agent (**ACL**) qui reflètent certaines des propriétés de la langue humaine :
  - **La notion de Speech Act** : permet à un agent A de communiquer à un agent B non seulement une proposition P mais aussi l'Intention que A prend au sujet de P et de B :  
**Soit P = « Il pleut »**  
**INFORM (A,B) [ BELA(P) ] = «Je crois sincèrement, et je t'informe qu'il pleut»**
  - **Conséquences :**
    1. **B sait que A croît sincèrement P (formellement INFORM impose la sincérité)**
    2. **B sait que A a voulu informer B au sujet de P (coopération ? menace ? ...)**
    3. **B choisit ou non d'adopter la croyance de P (il croît à son tour P) ou non**
    4. **B réagit à l'impact que 1-3 peut avoir sur ses propres stratégies.**
- L'intention est importante pour le raisonnement sur les traces d'interactions

# Architectures parallèles et SMA

## Architectures centralisées

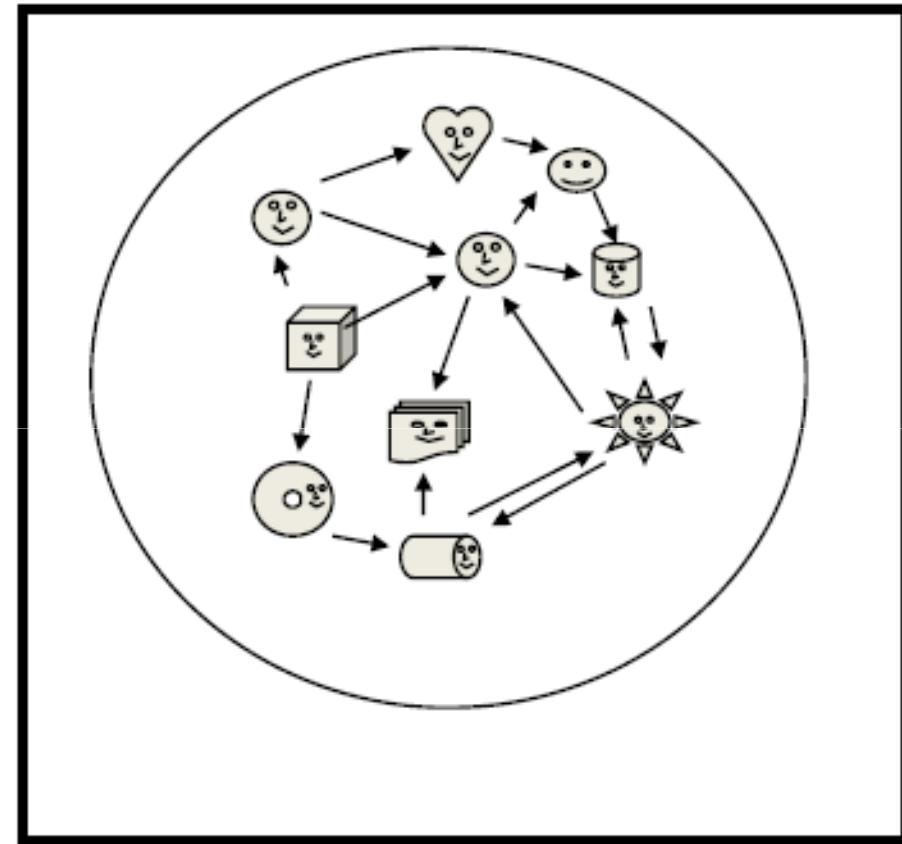
SIMD ou SPMD



- Les agents ont tous la même structure : état et comportement.
- **Emergence ou d'Intelligence Collective.**

## Architectures décentralisées

MIMD ou MPMD



- Les agents ont des structures différentes
- **Sociétés d'Experts**



# Applications des SMA

- Simulation de phénomènes complexes,
  - Sociologie, physique des particules,
  - Chimie, Robotique,
  - Biologie cellulaire, Ethologie, Ethnologie, ...
- IAD pour la résolution de problèmes complexes
- Systèmes parallèles et distribués
- Robotique, Productique
- Télécoms
- Jeux vidéos, Cinéma (Animations 3D)
- E-commerce, E-learning
- Urbanisation
- Smart Grid
- Gestion du trafic routier
- Finance : Trading automatique
- Web Sémantique
- Internet des objets (Web 3.0)
- ...



# Simulation de phénomènes complexes,

- En sociologie, paramétriser les différents agents composant une communauté. En ajoutant des contraintes, on peut essayer de comprendre quelle sera la composante la plus efficace pour parvenir à un résultat attendu (construction d'un pont). Ils permettent même d'expérimenter des scénarios qui ne seraient pas réalisables sur des populations réelles, que ce soit pour des raisons techniques ou éthiques.
- Des applications existent en physique des particules (agent = particule élémentaire),
- en chimie (agent = molécule),
- en robotique (agent = robot, dans le cas d'une implémentation sur robot réel, on parlera de système multi-robots),
- en biologie cellulaire (agent = cellule),
- en éthologie (agent = animal),
- en sociologie et en ethnologie (agent = être humain).



# Plateformes SMA

- **AnyLogic** : Logiciel de simulation multi-agents et multi-méthode
- **CORMAS** : (COmmon Resources Multi-Agent System) est un framework de développement de systèmes multi-agents, open-source et basé sur le langage de programmation orientée objet SmallTalk. Il est centré sur des problématiques de recherche en sciences du développement et de négociation entre acteurs.
- **DoMIS** : est un outil permettant la conception de Systèmes Multi-agents (orientés "pilotage opérationnel de systèmes complexes") . Utilisé Pour l'analyse décisionnelle des systèmes complexes.



# Plateformes SMA

- **JACK** : est un langage de programmation et un environnement de développement pour agents cognitifs, développé par la société Agent Oriented Software comme une extension orientée agent du langage Java.
- **JADE** : (Java Agent DEvelopment) est un framework de développement de systèmes multi-agents, open-source et basé sur le langage Java. Il offre en particulier un support avancé de la norme FIPA-ACL, ainsi que des outils de validation syntaxique des messages entre agents basé sur les ontologies.



# Plateformes SMA

- **Jadex** : est une plate-forme agent développée en JAVA par l'université de Hambourg qui se veut modulaire, compatible avec de nombreux standards et capable de développer des agents.
- **Jagent** : est un framework open source réalisé en Java dont l'objectif est de faciliter le développement et le test de systèmes multi-agents.



# Plateformes SMA

- **Janus** : est une plateforme multi-agents modulaire écrite en Java. Elle permet de créer des systèmes multi-agents avec ou sans une approche organisationnelle basée sur le modèle Capacité-Rôle-Interaction-Organisation (CARIO).
- **Jason** : est un environnement open source de développement d'agents dans le formalisme AgentSpeak, et développé en Java



# Plateformes SMA

- **MadKit** : est une plate-forme multi-agents modulaire écrite en Java et construite autour du modèle organisationnel Agent/Groupe/Rôle. C'est une plate-forme libre basée sur la licence GPL/LGPL développée au sein du LIRMM.
- **MAGIQUE** : est une plate-forme pour agents physiquement distribués écrite en Java et fournissant un modèle de communication original d'appel à la cantonade. Dans MAGIQUE, les compétences sont dissociées des agents. L'architecture des agents et les différentes compétences sont développées séparément. Les compétences sont ensuite greffées comme plugin dans les agents au gré du concepteur. Cette plate-forme est développée au sein du LIFL.



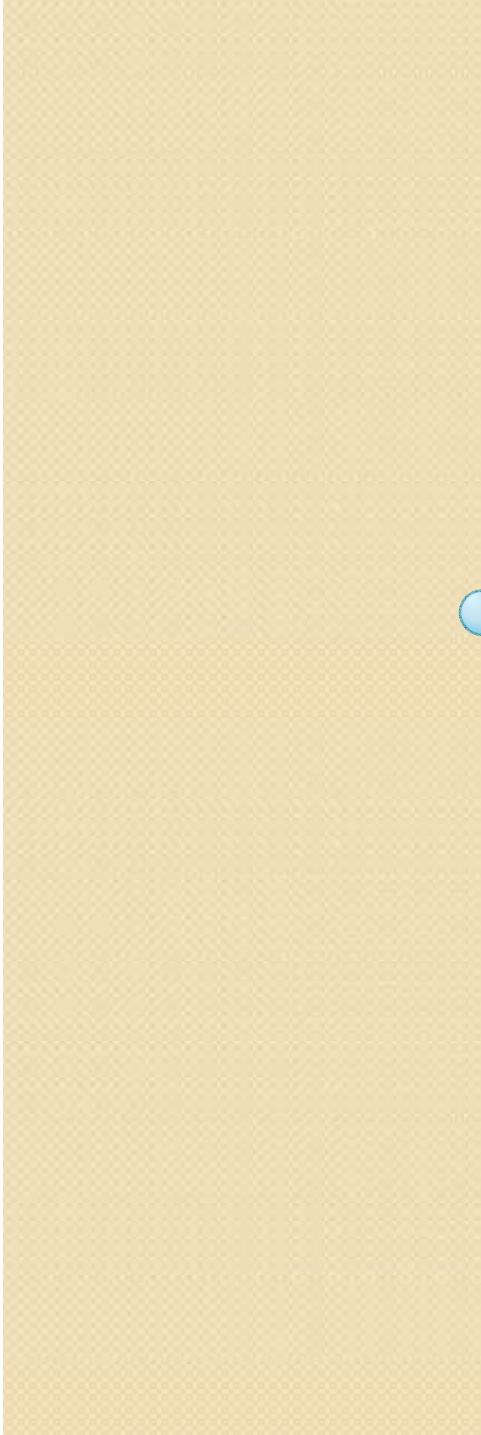
# Plateformes SMA

- **OMAS** : Open Multi-Agent Asynchronous Systems est une plate-forme de recherche développée par l'équipe d'intelligence artificielle de l'Université de technologie de Compiègne, sous la direction de Jean-Paul Barthès.
- **SemanticAgent** : est basé sur JADE et permet le développement d'agents dont le comportement est représenté en SWRL. SemanticAgent est développé au sein du LIRIS, il est open-source et sous licence GPL V3.
- **SPADE**: est un environnement de développement d'organisations multi-agents basé sur le protocole XMPP et est écrit en Python.
- **MASSIVE** : est un logiciel pour la simulation de foule, basé multi-agents, qui a permis la création d'effets spéciaux dans un grand nombre de films, ayant été développé à l'origine pour les scènes de combat dans Le Seigneur des anneaux.



# Plateformes SMA

- Golaem Crowd est un plug-in pour Maya (logiciel) basé multi-agent et permettant d'effectuer des simulations de foule pour les effets spéciaux directement dans Maya.



- **MISE EN ŒUVRE DES  
SMA AVEC JADE**





# JADE? Java Agent DEvlopement

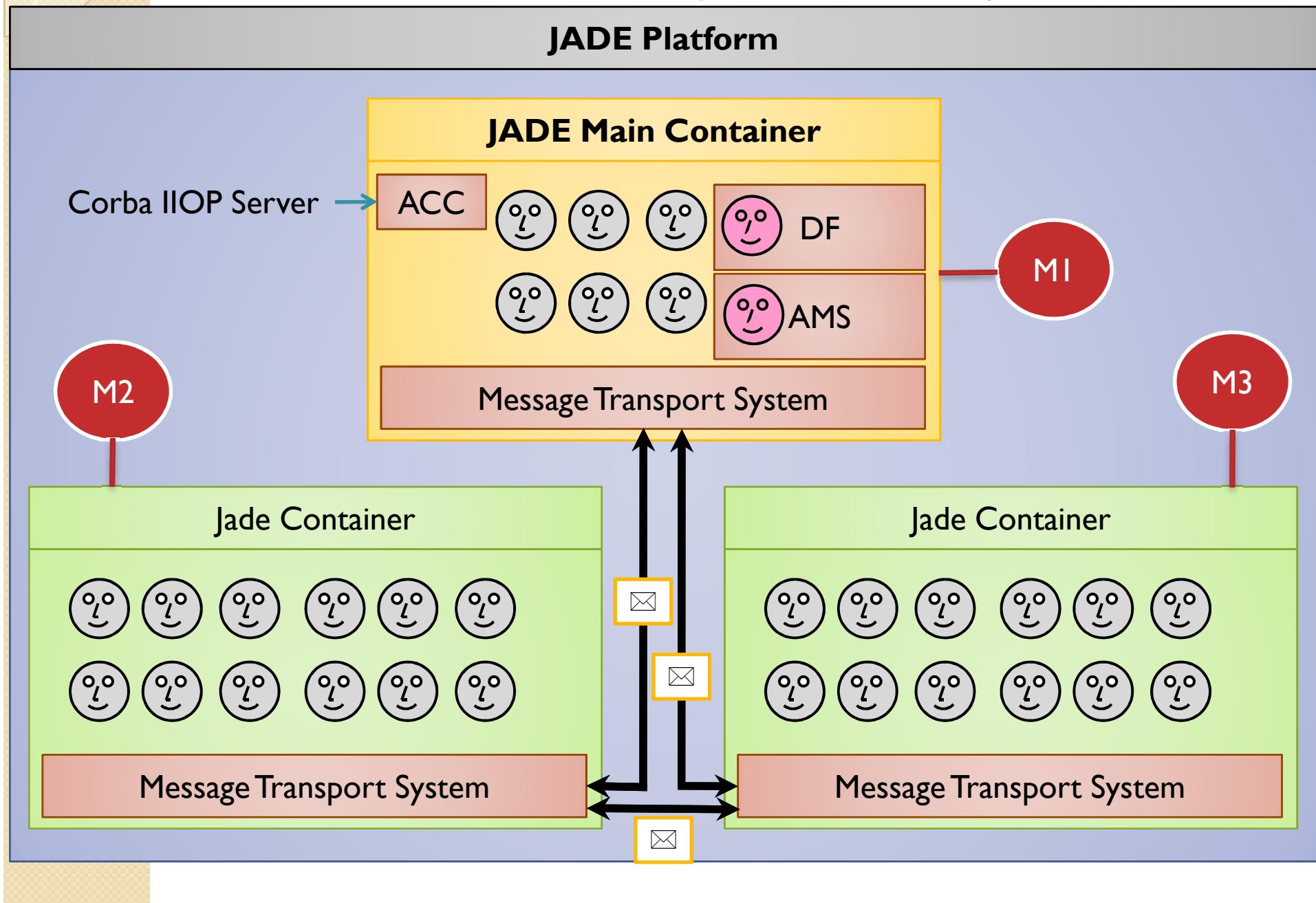
- JADE est une plate-forme multi-agent créé par le laboratoire TILAB.
- C'est un framework qui permet le développement de systèmes multi-agents et d'applications conformes aux normes FIPA (Foundation for Intelligent Physical Agents).
- La FIPA est une organisation en 1996 dont l'objectif est de produire des standards pour l'interopération d'agents logiciels hétérogènes.
- JADE possède trois modules principaux (nécessaire aux normes FIPA).
  - DF « Directory Facilitator » fournit un service de « pages jaunes» à la plate-forme ;
  - ACC «Agent Communication Channel » gère la communication entre les agents ;
  - AMS « Agent Management System » supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.
- Ces trois modules sont activés à chaque démarrage de la plate-forme.



# Plateforme SMA : JADE

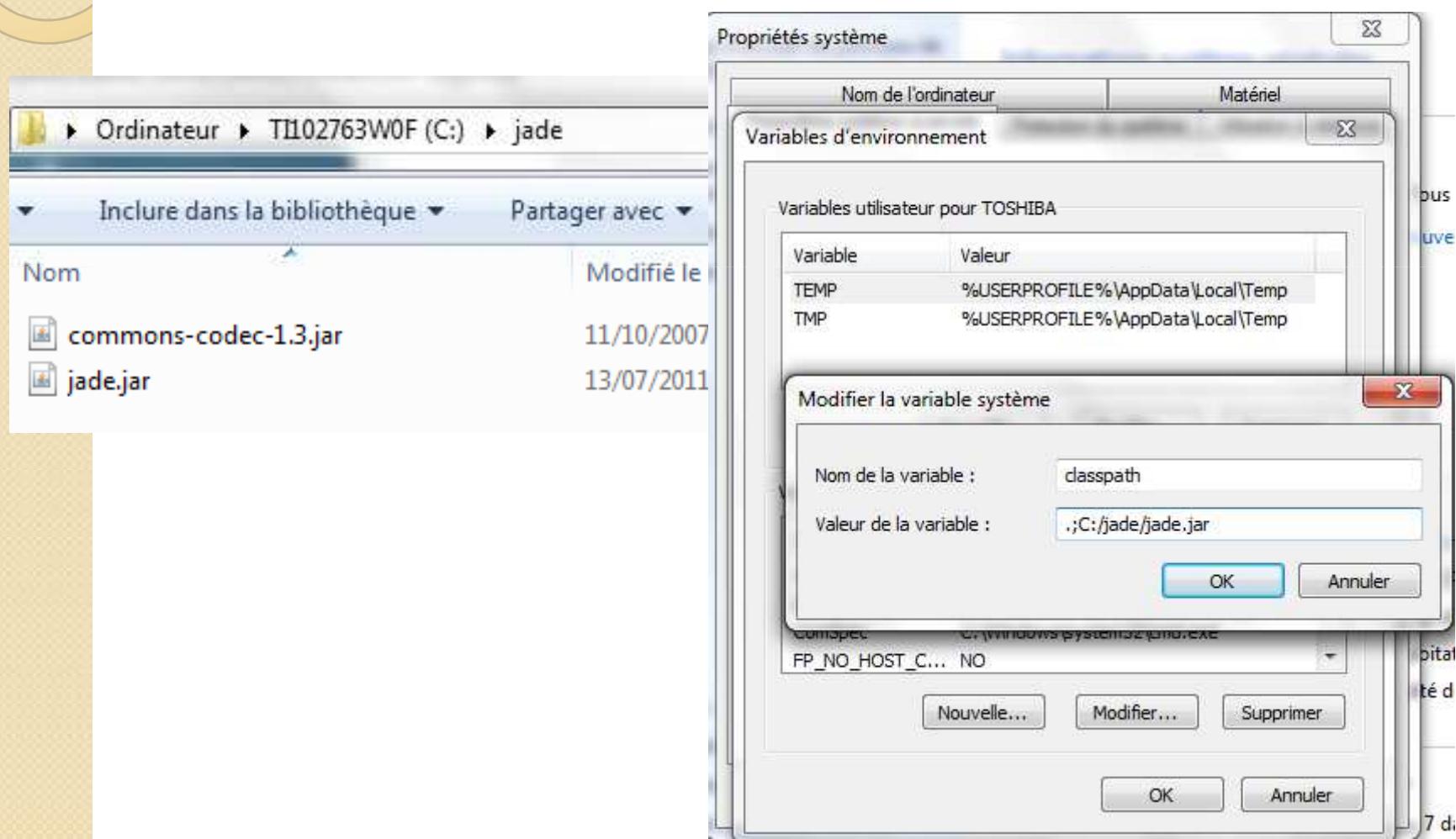
- JADE est un middleware qui facilite le développement des systèmes multi agents (SMA). JADE contient :
  - Un Runtime Environment : l'environnement où les agents peuvent vivre.
  - Une librairie de classes : que les développeurs utilisent pour écrire leurs agents
  - Une suite d'outils graphiques : qui facilitent la gestion et la supervision de la plateforme des agents
- Chaque instance du JADE est appelée conteneur « **Container** », et peut contenir plusieurs agents.
- Un ensemble de conteneurs constituent une **plateforme**.
- Chaque plateforme doit contenir un conteneur spécial appelé **main-container** et tous les autres conteneurs s'enregistrent auprès de celui-là dès leur lancement

# Architecture Logicielle de JADE



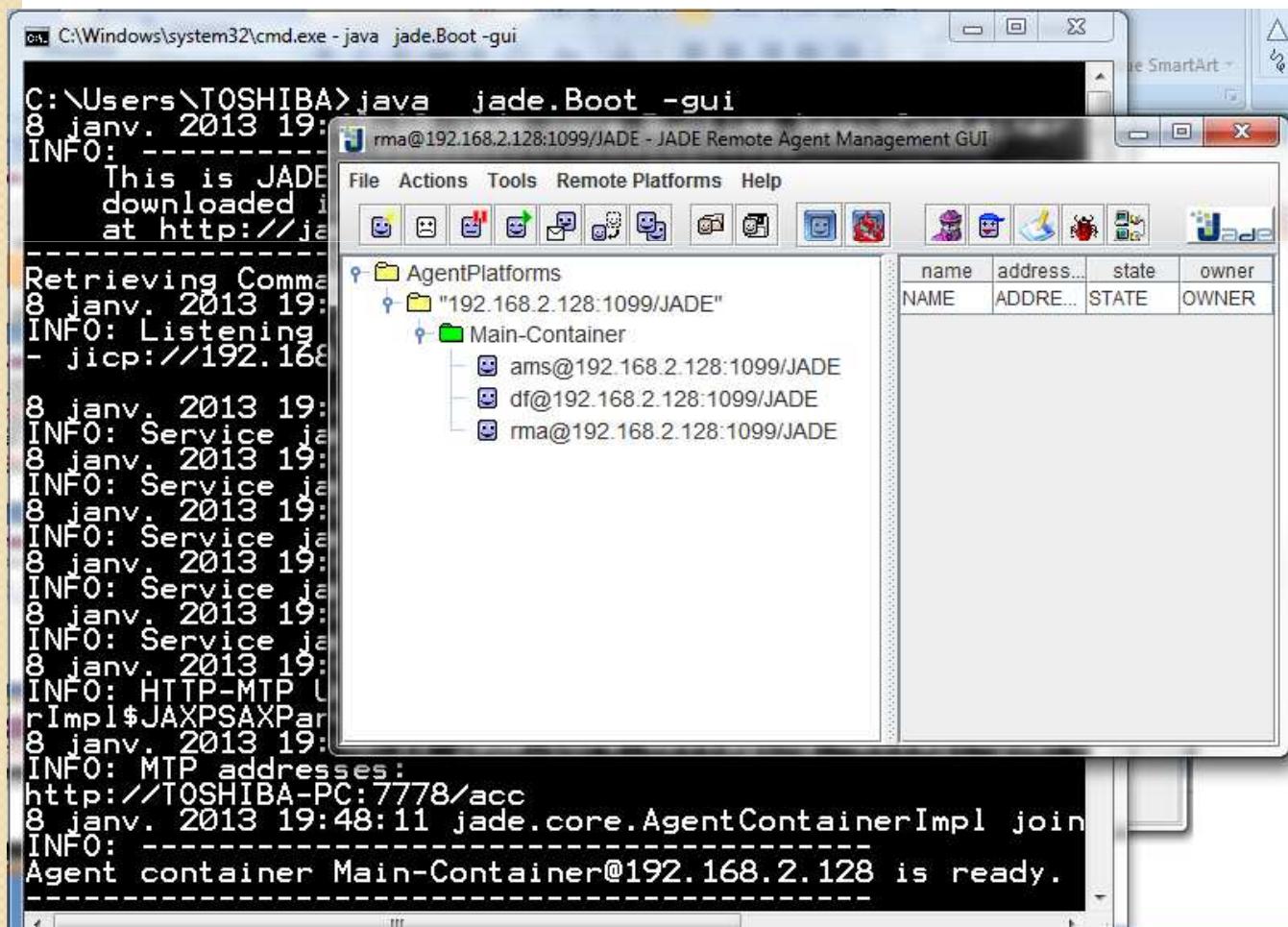
# Configuration

- Pour manipuler JADE sur ligne de commande, vous aurez besoin d'ajouter la variable d'environnement classpath je fchier **jade.jar**



# Lancer la plateforme jade sur ligne de commande

- pour lancer MainContainer sur ligne de commande:
  - Taper la commande : **java jade.Boot -gui**



# Lancer la plateforme jade sur ligne de commande

- pour lancer un autre container dans une autre machine :
  - Taper la commande : `java jade.Boot -container -host Main_IP_Adress`

The screenshot illustrates the process of launching a JADE container via the command line and monitoring its status through a graphical interface.

In the background, a Windows Command Prompt window shows the command being run:

```
C:\Windows\system32\cmd.exe - java jade.Boot -container -host localhost
```

The output of the command is displayed in the foreground, showing the creation of a "Main-Container" and a "Container-1" on the specified host:

```
C:\Users\TOSHIBA>java jade.Boot -container -host localhost
8 janv. 2013 20:04:53 jade.core.Runtime beginContainer
INFO: -----
      This is JADE 4.1
      downloaded in Op
      at http://jade.t
-----
Retrieving CommandDi
8 janv. 2013 20:04:5
INFO: Listening for
- jicp://192.168.2.1
8 janv. 2013 20:04:5
INFO: Service jade.c
8 janv. 2013 20:04:5
```

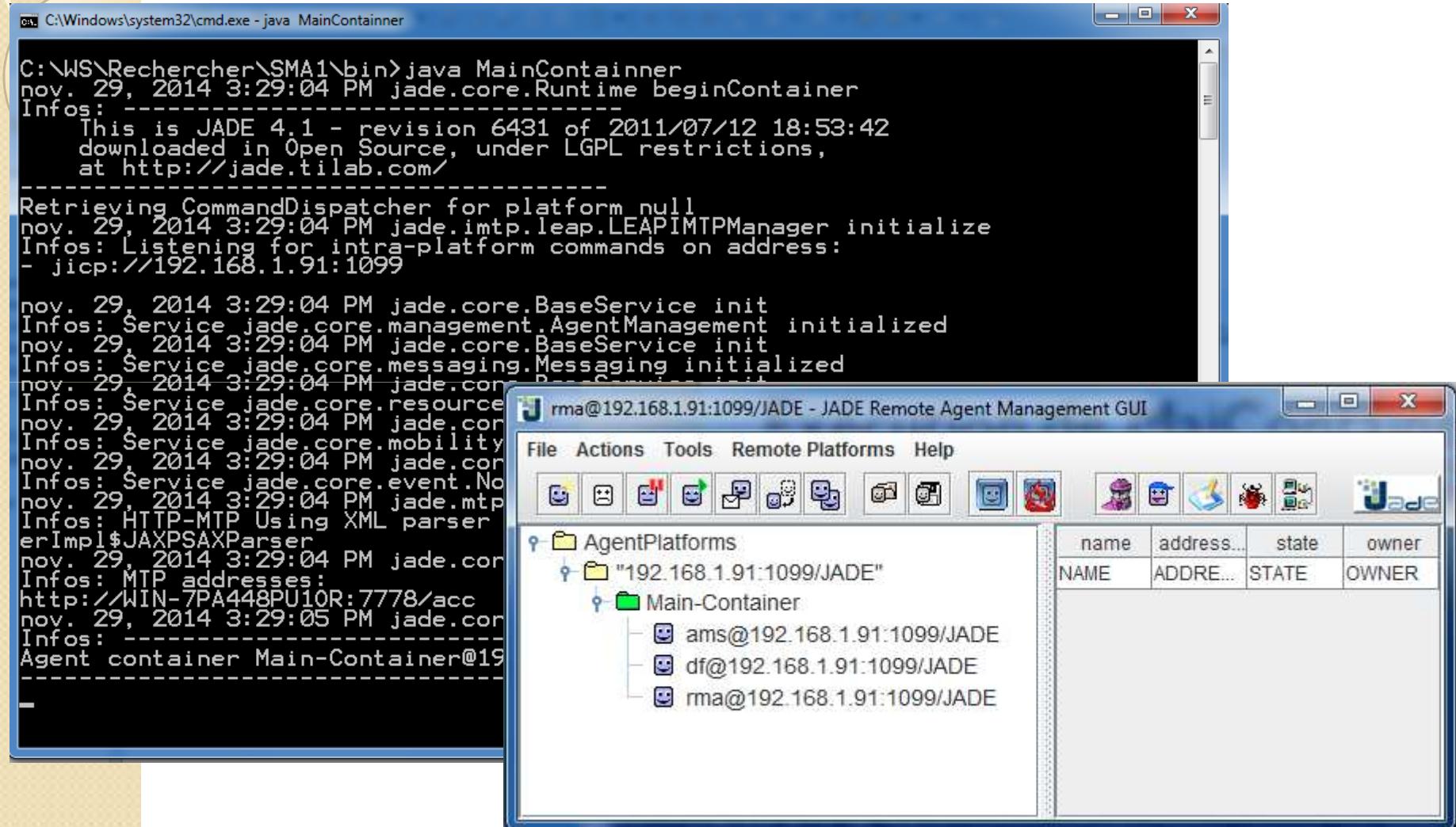
Simultaneously, the JADE Remote Agent Management GUI is open, showing the same information. The left pane displays the hierarchical structure of agent platforms and containers. The right pane is a table listing agents with columns for name, addresses, state, and owner.

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

# Démarrer le MainContainer avec une application java

```
import jade.core.Profile; import jade.core.ProfileImpl;
import jade.core.Runtime; import jade.util.ExtendedProperties;
import jade.util.leap.Properties;
import jade.wrapper.AgentContainer;
public class MainContainner {
public static void main(String[] args) {
try{
    Runtime runtime=Runtime.instance();
    Properties properties=new ExtendedProperties();
    properties.setProperty(Profile.GUI, "true");
    ProfileImpl profileImpl=new ProfileImpl(properties);
    AgentContainer mainContainer=runtime.createMainContainer(profileImpl);
    mainContainer.start();
}
catch(Exception e){ e.printStackTrace(); }
}
}
```

# Exécution de MainContainer



The image shows two windows demonstrating the execution of a MainContainer. On the left is a command-line window titled 'cmd C:\Windows\system32\cmd.exe - java MainContainerner'. It displays the startup logs for JADE 4.1, including the beginning of the container, the retrieval of a CommandDispatcher, and the initialization of various services like jade.core.management.AgentManagement and jade.core.messaging.Messaging. It also shows the creation of an agent named 'Main-Container' at address '192.168.1.91:1099/JADE'. On the right is a 'JADE Remote Agent Management GUI' window titled 'rma@192.168.1.91:1099/JADE'. This window lists the agents currently running on the platform, including 'ams@192.168.1.91:1099/JADE', 'df@192.168.1.91:1099/JADE', and 'rma@192.168.1.91:1099/JADE'.

```
C:\WS\Rechercher\SMA1\bin>java MainContainerner
nov. 29, 2014 3:29:04 PM jade.core.Runtime beginContainer
Infos: -----
    This is JADE 4.1 - revision 6431 of 2011/07/12 18:53:42
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform null
nov. 29, 2014 3:29:04 PM jade.imtp.leap.LEAPIMTPManager initialize
Infos: Listening for intra-platform commands on address:
- jicp://192.168.1.91:1099
-----
nov. 29, 2014 3:29:04 PM jade.core.BaseService init
Infos: Service jade.core.management.AgentManagement initialized
nov. 29, 2014 3:29:04 PM jade.core.BaseService init
Infos: Service jade.core.messaging.Messaging initialized
nov. 29, 2014 3:29:04 PM jade.core.resource.Resource initialized
Infos: Service jade.core.resource.Resource initialized
nov. 29, 2014 3:29:04 PM jade.core.mobility.Mobility initialized
Infos: Service jade.core.mobility.Mobility initialized
nov. 29, 2014 3:29:04 PM jade.core.event.Notification initialized
Infos: Service jade.core.event.Notification initialized
nov. 29, 2014 3:29:04 PM jade.mtp.MTP initialized
Infos: HTTP-MTP Using XML parser
erImpl$JAXPSAXParser
nov. 29, 2014 3:29:04 PM jade.core.Iface initialized
Infos: MTP addresses:
http://WIN-7PA448PU10R:7778/acc
nov. 29, 2014 3:29:05 PM jade.core.Iface initialized
Infos: -----
Agent container Main-Container@192.168.1.91:1099/JADE
-----
```

JADE Remote Agent Management GUI

name	address...	state	owner
NAME	ADDRE...	STATE	OWNER

- AgentPlatforms
- "192.168.1.91:1099/JADE"
  - Main-Container
    - ams@192.168.1.91:1099/JADE
    - df@192.168.1.91:1099/JADE
    - rma@192.168.1.91:1099/JADE

# Démarrer un AgentContainer avec une application java

```
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
public class Container {
    public static void main(String[] args) {
        try {
            Runtime runtime=Runtime.instance();
            ProfileImpl profileImpl=new ProfileImpl(false);
            profileImpl.setParameter(ProfileImpl.MAIN_HOST, "localhost");
            AgentContainer agentContainer=runtime.createAgentContainer(profileImpl);
            agentContainer.start();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

# Lancement d'un conteneur JADE

Lancement de 3 conteneurs JADE

The image displays three windows illustrating the launch of three JADE containers:

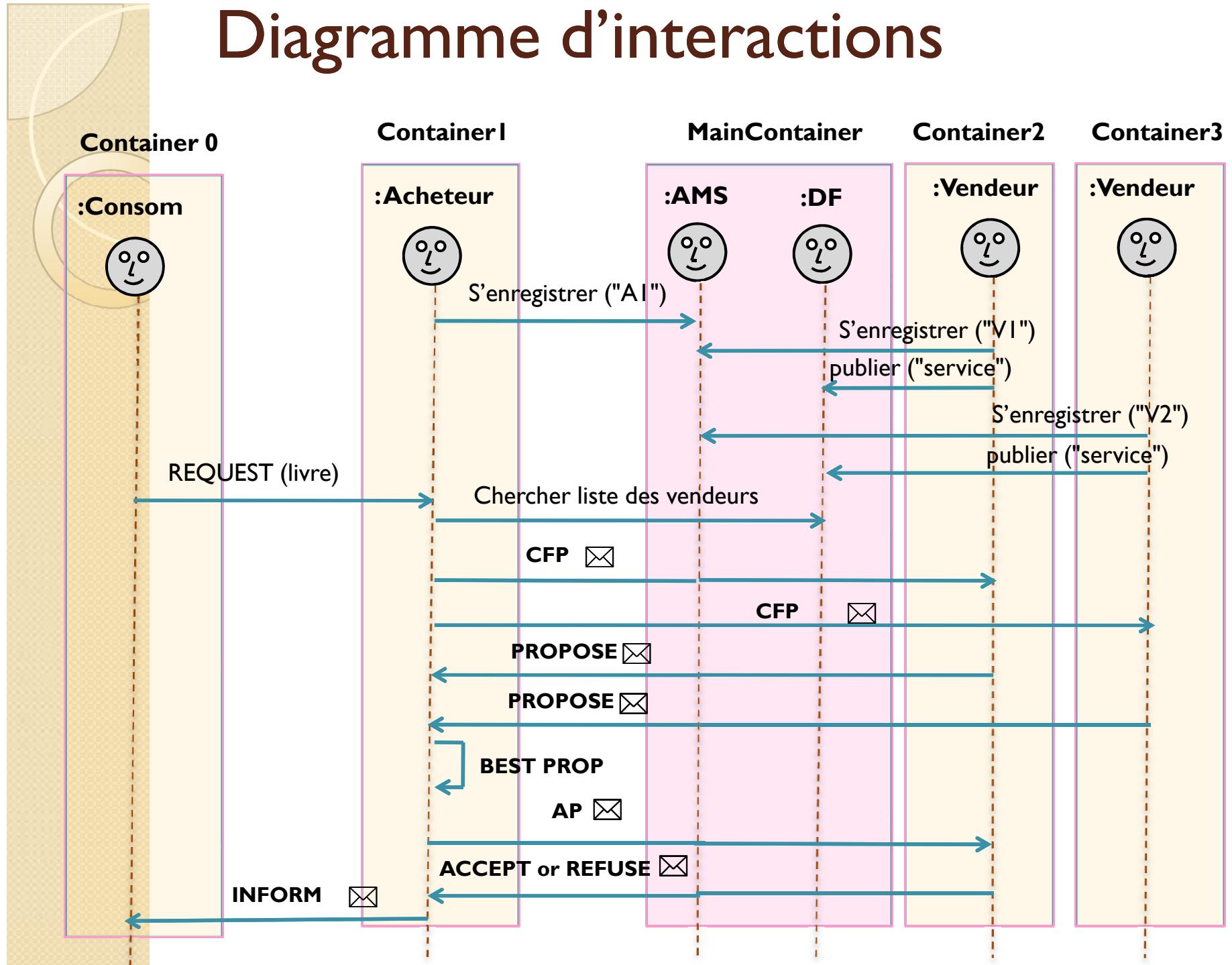
- Top Window:** A command prompt window titled "C:\Windows\system32\cmd.exe" showing four consecutive commands: "start java Container".
- Middle Window:** A command prompt window titled "Sélectionner C:\Windows\system32\cmd.exe - java Container" showing the output of launching a JADE container. It includes the JADE startup log: "nov. 29, 2014 5:58:38 PM jade.core.Runtime beginContainer", "Infos: This is JADE 4.1 - revision 6431 of 2011/07/12 18:53:42 downloaded in Open Source, under LGPL restrictions, at http://jade.tilab.com/", and "Retrieving CommandDispatcher for platform null". It also shows the listening address: "- jicp://192.168.43.36:24165".
- Bottom Window:** The "JADE Remote Agent Management GUI" window, showing the "AgentPlatforms" tree view. It lists "Main-Container" (containing agents "ams", "df", and "rma"), "Container-2", "Container-3", and "Container-4". A "Local Agent" button is visible.



## Créer les agents

- On souhaite créer une application multi-agents. Ce SMA est contient
  - Des agents qui demandent l'achat d'un livre
  - et d'autres agents qui vendent les livres.
  - l'acheteur devrait choisir l'agent qui offre le meilleur prix à travers un processus de communication entre les agents.

# Diagramme d'interactions





# Un Agent JADE

- Un agent JADE est une classe qui hérite de la classe Agent et qui redéfinie des méthodes qui définissent le cycle de vie de l'agent dans la plateforme.
- La méthode **setup** est la première méthode qui sera appelée après instantiation de l'agent par le container.
- la méthode **doDelete** permet de demander au container de détruire l'agent.
- Avant que l'agent soit détruit, la méthode **takeDown** est appelée.
- Un agent peut se déplacer d'un container à l'autre.
  - Avant chaque opération de migration, la méthode **beforeMove** est appelée
  - Après chaque opération de migration, la méthode **afterMove** est appelée
- D'autres méthodes seront présentés par la suite.

# My First Agent

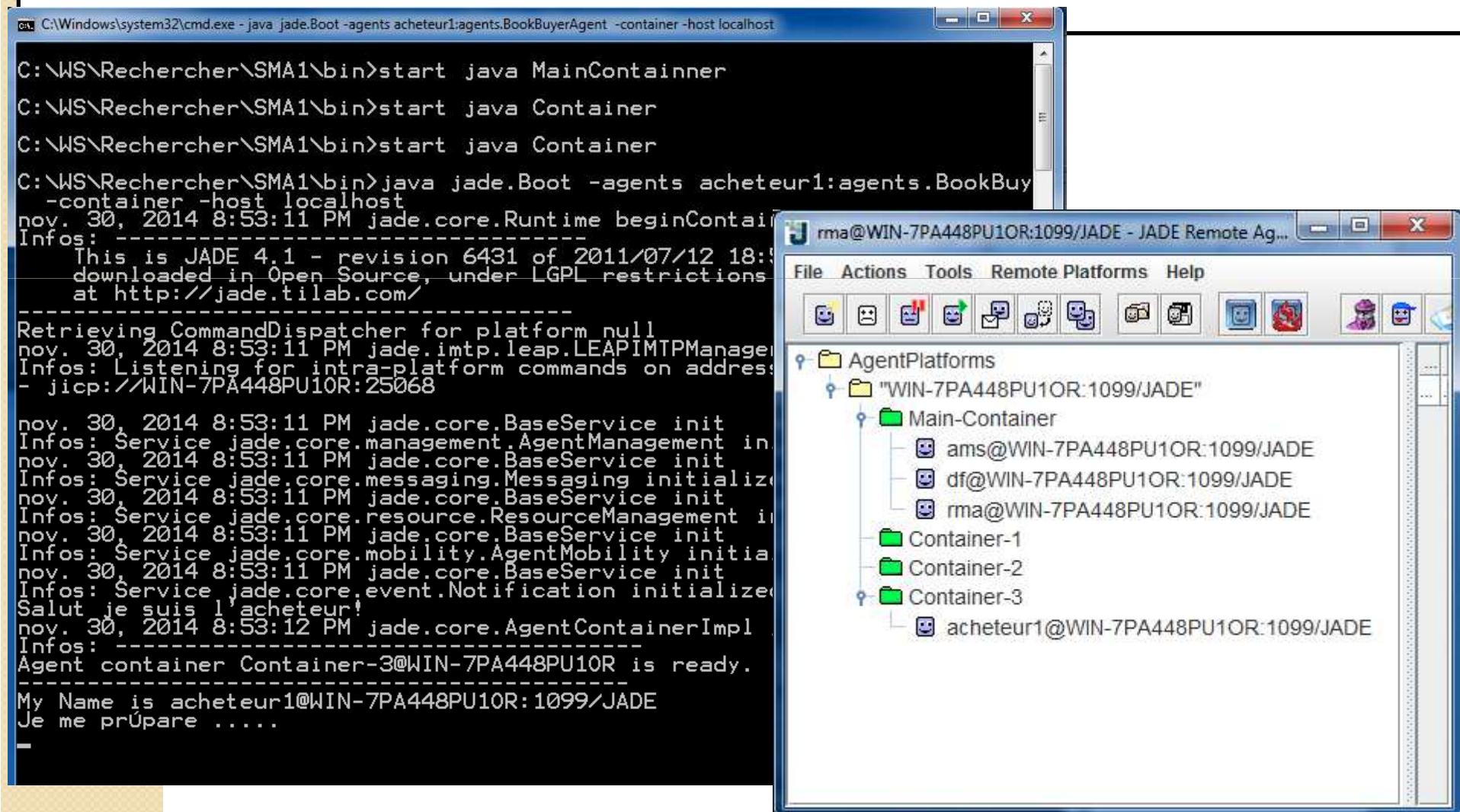
```
package agents;
import jade.core.Agent;
public class BookBuyerAgent extends Agent {
    @Override
    protected void setup() {
        System.out.println("Salut je suis l'acheteur!");
        System.out.println("My Name is "+this.getAID().getName());System.out.println("Je me prépare .....");
    }
    @Override
    protected void beforeMove() {
        System.out.println("Avant de migrer vers une nouvelle location .....");
    }
    @Override
    protected void afterMove() {
        System.out.println("Je viens d'arriver à une nouvelle location .....");
    }
    @Override
    protected void takeDown() {
        System.out.println("avant de mourir .....");
    }
}
```

# Agent identifier

- Chaque agent est identifié par un identifiant unique dans la plateforme.
- L'identifiant d'un objet de type **jade.code.AID**
- La méthode `getAID()` de la classe `Agent` permet de récupérer l'`agentIdentifier`
- Un objet de type `AID` contient le nom de l'agent plus le nom de la plateforme (nom de domaine ou adresse IP) ainsi que le numéro de port de l'annuaire.
- Exemple de nom complet : **acheteur1@192.168.30.12:1099/JADE**
- **Pour créer un objet de type AID sachant le nom local, on peut écrire :**
  - `AID vendeur=new AID("Vendeur1", AID.ISLOCALNAME);`
- **Le paramètre `ISLOCALNAME` indique que le premier paramètre représente le nom local.**

# Déployer l'agent sur ligne de commande

- Pour déployer un agent dans un nouveau container sur ligne de commande on peut écrire la commande suivante :
  - Java jade.Boot -container –host localhost –agents acheteur1:agents.BookBuyerAgent**



```
C:\Windows\system32\cmd.exe - java jade.Boot -agents acheteur1:agents.BookBuyerAgent -container -host localhost

C:\WS\Rechercher\SMA1\bin>start java MainContainner
C:\WS\Rechercher\SMA1\bin>start java Container
C:\WS\Rechercher\SMA1\bin>start java Container
C:\WS\Rechercher\SMA1\bin>java jade.Boot -agents acheteur1:agents.BookBuyerAgent -container -host localhost
nov. 30, 2014 8:53:11 PM jade.core.Runtime beginContainer
Infos: -----
This is JADE 4.1 - revision 6431 of 2011/07/12 18:54
downloaded in Open Source, under LGPL restrictions
at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform null
nov. 30, 2014 8:53:11 PM jade.imtp.leap.LEAPIMTPManager
Infos: Listening for intra-platform commands on address
- jicp://WIN-7PA448PU1OR:25068

nov. 30, 2014 8:53:11 PM jade.core.BaseService init
Infos: Service jade.core.management.AgentManagement initialized
nov. 30, 2014 8:53:11 PM jade.core.BaseService init
Infos: Service jade.core.messaging.Messaging initialized
nov. 30, 2014 8:53:11 PM jade.core.BaseService init
Infos: Service jade.core.resource.ResourceManagement initialized
nov. 30, 2014 8:53:11 PM jade.core.BaseService init
Infos: Service jade.core.mobility.AgentMobility initialized
nov. 30, 2014 8:53:11 PM jade.core.BaseService init
Infos: Service jade.core.event.Notification initialized
Salut je suis l'acheteur!
nov. 30, 2014 8:53:12 PM jade.core.AgentContainerImpl
Infos: -----
Agent container Container-3@WIN-7PA448PU1OR is ready.
-----
My Name is acheteur1@WIN-7PA448PU1OR:1099/JADE
Je me prépare ....
```

The JADE Remote Agent interface window shows the following agent list:

- AgentPlatforms
  - "WIN-7PA448PU1OR:1099/JADE"
    - Main-Container
      - ams@WIN-7PA448PU1OR:1099/JADE
      - df@WIN-7PA448PU1OR:1099/JADE
      - rma@WIN-7PA448PU1OR:1099/JADE
    - Container-1
    - Container-2
    - Container-3
      - acheteur1@WIN-7PA448PU1OR:1099/JADE

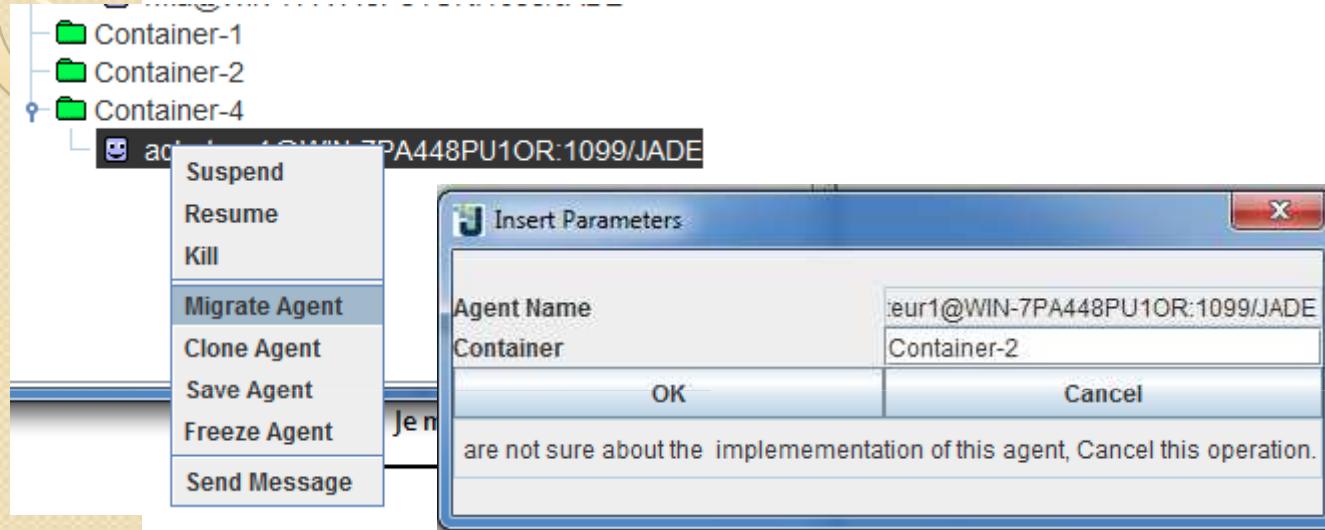
# Déployer l'agent dans une application java :

```
import jade.core.ProfileImpl; import jade.core.Runtime;
import jade.wrapper.AgentContainer; import jade.wrapper.AgentController;
public class BookBuyerContainer {
    public static void main(String[] args) {
        try {
            Runtime runtime=Runtime.instance();
            ProfileImpl profileImpl=new ProfileImpl(false);
            profileImpl.setParameter(ProfileImpl.MAIN_HOST, "localhost");
            AgentContainer agentContainer=runtime.createAgentContainer(profileImpl);
            AgentController agentController=agentContainer.createNewAgent("acheteur1",
"agents.BookBuyerAgent", new Object[]{});
            agentController.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Infos: -----  
Agent container Container-4@192.168.1.26 is ready.  
-----  
Salut je suis l'acheteur!  
My Name is acheteur1@WIN-7PA448PU1OR:1099/JADE  
Je me prépare .....

# Tester la mobilité de l'agent

- L'interface graphique de JADE permet de demander à un agent de migrer d'un container à l'autre.



Agent container Container-4@192.168.1.26 is ready.

-----  
Salut je suis l'acheteur!

My Name is acheteur1@WIN-7PA448PU1OR:1099/JADE

Je me prépare .....

**Avant de migrer vers une nouvelle location .....**

```
Infos: -----
Agent container Container-2@WIN-7PA448PU1OR is ready.
-----
Je viens d'arriver à une nouvelle location .....
```



## Passer des argument à un agent au moment du déploiement

- Les agents peuvent récupérer les arguments passé au moment de déploiement sur ligne de commande ou dans le programme sous forme d'un tableau d'objets.
- La méthode `getArguments()` de la classe `Agent` permet de récupérer la liste des arguments.
- Supposant qu'au moment du déploiement de notre agents nous passons comme argument le nom du livre à acheter:
- Sur ligne de commande :

**Java jade.Boot -container –host localhost –agents acheteur1:agents.BookBuyerAgent(XML)**

- Avec le code java :

```
AgentController agentController=
agentContainer.createNewAgent("acheteur1","agents.BookBuyerAgent
", new Object[]{ "XML"});
```

# Passer des argument à un agent au moment du déploiement

```
package agents; import jade.core.Agent;import jade.core.Location;
public class BookBuyerAgent extends Agent {
    private String livre;
    @Override
    protected void setup() {
        Object[]args=getArguments();
        if(args.length==1){ livre=(String) args[0];
            System.out.println("Salut L'acheteur :"+
this.getAID().getName()+" est prêt \n je tente d'acheter lelivre :"+livre);
        }
        else{
            System.out.println("il faut spécifier Le Livre comme argument");
            doDelete();
        }
    }
    @Override
    protected void takeDown() {
        System.out.println("Destruction de L'agent "+this.getAID().getName());
    }
}
```

# Déployer l'agent avec des arguments

C:\Windows\system32\cmd.exe

```
C:\WS\Rechercher\SMA1\bin>java jade.Boot -agents acheteur1:agents.BookBuyerAgent
(XML) -container -host localhost
nov. 30, 2014 9:37:23 PM jade.core.Runtime beginContainer
Infos: -----
    This is JADE 4.1 - revision 6431 of 2011/07/12 18:53:42
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform null
nov. 30, 2014 9:37:23 PM jade.imtp.leap.LEAPIMTPManager initialize
Infos: Listening for intra-platform commands on address:
- jicp://192.168.1.26:25267
nov. 30, 2014 9:37:23 PM jade.core.BaseService init
Infos: Service jade.core.management.AgentManagement initialized
nov. 30, 2014 9:37:23 PM jade.core.BaseService init
Infos: Service jade.core.messaging.Messaging initialized
nov. 30, 2014 9:37:23 PM jade.core.BaseService init
Infos: Service jade.core.resource.ResourceManagement initialized
nov. 30, 2014 9:37:23 PM jade.core.BaseService init
Infos: Service jade.core.mobility.AgentMobility initialized
nov. 30, 2014 9:37:23 PM jade.core.BaseService init
Infos: Service jade.core.event.Notification initialized
Salut je suis l'acheteur!
nov. 30, 2014 9:37:23 PM jade.core.AgentContainerImpl joinPlatform
Infos: -----
Agent container Container-7@192.168.1.26 is ready.
-----
My Name is acheteur1@WIN-7PA448PU10R:1099/JADE
Je me prÙpare ....
Cette action est exÙcutÙe une seule fois
Tentative Num 1 pour acheter le livre XML
Tentative Num 2 pour acheter le livre XML
Tentative Num 3 pour acheter le livre XML
Tentative Num 4 pour acheter le livre XML
Tentative Num 5 pour acheter le livre XML
```

# Affecter les comportements à un agent

- Pour qu'un agent JADE exécute une tâche, nous avons tout d'abord besoin de définir ces tâches.
- Les tâches dans JADE (appelées **behaviours** ou des **comportements**) sont des instances de la classe **jade.core.Behaviours** .
- pour qu'un agent exécute une tâche on doit lui l'attribuer par la méthode **addBehaviour(Behaviour b)** de la classe **jade.core.Agent**.
- Chaque Behaviour doit implémenter au moins les deux méthodes :
  - **action()** : qui désigne les opérations à exécuter par le Behaviour;
  - **done()** : qui exprime si le Behaviour a terminé son exécution ou pas.
- Il existe deux autres méthodes dont l'implémentation n'est pas obligatoire mais qui peuvent être très utiles :
  - **onStart()** : appelée juste avant l'exécution de la méthode action();
  - **onEnd()** : appelée juste après la retournelement de **true** par la méthode done().

## Affecter les comportements à un agent

- Des fois on a besoin de savoir quel est le propriétaire d'un Behaviour, et cela peut être connu par le membre **myAgent** du Behaviour en question.
- JADE alloue un thread par agent, pour cela un agent exécute un Behaviour à la fois.
- L'agent peut exécuter plusieurs Behaviours simultanément en choisissant un bon mécanisme de passation d'un Behaviour à un autre (c'est à la charge du programmeur et non pas à la charge du JADE).



# Behaviours :

- JADE offre trois types de Behaviours simple. Ces Behaviours sont :
  - **One-shot Behaviour**
    - Un **one-shot Behaviour** est une instance de la classe **jade.core.behaviours.OneShotBehaviour**.
    - Il a la particularité d'exécuter sa tâche une et une seule fois puis il se termine.
    - La classe OneShotBehaviour implémente la méthode **done()** et elle retourne toujours **true**.
  - **Cyclic Behaviour**
    - Un **cyclic Behaviour** est une instance de la classe **jade.core.behaviours.CyclicBehaviour**.
    - un cyclic Behaviour exécute sa tâche d'une manière répétitive.
    - La classe CyclicBehaviour implémente la méthode **done()** qui retourne toujours **false**.
  - **Generic Behaviour**
    - Un Generic Behaviour est une instance de la classe **jade.core.behaviours.Behaviour**.
    - Le **Generic Behaviour** vient entre le One-shot Behaviour et le Cyclic Behaviour de fait qu'il n'implémente pas la méthode **done()** et laisse son implémentation au programmeur, donc il peut planifier la terminaison de son Behaviour selon ces besoin.



# Les Behaviours planifiés

- **WakerBehaviour**
  - Le WakerBehaviour est implémenté de façon à exécuter la méthode **onWake()** après une période passée comme argument au constructeur.
  - Cette période est exprimée en millisecondes. Le Behaviour prend fin juste après avoir exécuté la méthode **onWake()**.
- **TickerBehaviour**
  - Le TickerBehaviour est implémenté pour qu'il exécute sa tâche périodiquement par la méthode **onTick()**.
  - La durée de la période est passée comme argument au constructeur.

# ParallelBehaviour

- Pour qu'un agent puisse exécuter plusieurs comportements à la fois, il faut utiliser ParallelBehaviour

```
package fa; import ...
public class FirstAgent extends Agent {
    @Override
    protected void setup() {
        ParallelBehaviour parallelBehaviour=new ParallelBehaviour();
        addBehaviour(parallelBehaviour);

        parallelBehaviour.addSubBehaviour(new CyclicBehaviour() {
            @Override
            public void action() { // T1}
        });

        parallelBehaviour.addSubBehaviour(new TickerBehaviour(this,1000) {
            @Override
            protected void onTick() { // T2 }
        });

        parallelBehaviour.addSubBehaviour(new Behaviour() {
            int compteur;
            @Override
            public boolean done() { return (compteur==4);}
            @Override public void action() { ++compteur; // T3 }
       }); }}
```

# Exemple | :

```
public class BookBuyerAgent extends Agent {  
    private String livre; private int compteur;  
  
    @Override  
    protected void setup() {  
        System.out.println("Salut je suis l'acheteur!");  
        System.out.println("My Name is "+this.getAID().getName()); System.out.println("Je me prépare .....");  
        Object[] args=getArguments();  
        if(args.length==1){ livre=(String) args[0]; } else{  
            System.out.println("J'ai besoin du nom du Livre à acheter ...");  
            doDelete();  
        }  
    }  
  
    addBehaviour(new OneShotBehaviour() {  
        @Override  
        public void action() {  
            System.out.println("Cette action est exécutée une seule fois");  
        } });  
  
    addBehaviour(new TickerBehaviour(this,1000) {  
        @Override  
        protected void onTick() {  
            ++compteur;System.out.println("Tentative Num "+compteur+" pour acheter le livre "+livre);  
        }});  
}
```

# Redéployer l'agent

```
AgentController agentController=agentContainer.createNewAgent("acheteur1",
"agents.BookBuyerAgent", new Object[]{"XML"});
```

Salut je suis l'acheteur!

My Name is acheteur1@WIN-7PA448PUJOR:1099/JADE

Je me prépare .....

Cette action est exécutée une seule fois

Tentative Num 1 pour acheter le livre XML

Tentative Num 2 pour acheter le livre XML

Tentative Num 3 pour acheter le livre XML

Tentative Num 4 pour acheter le livre XML

...



# Communication entre agents

- Pour que plusieurs agents JADE arrivent à collaborer, ils doivent s'échanger des messages.
- Chaque agent JADE possède une sorte de boite aux lettres qui contient les messages qui lui sont envoyés par les autres agents.
- Ces boites aux lettres sont sous forme d'une liste qui contient les messages selon l'ordre chronologique de leur arrivée.



# Format d'un message JADE

- Les agents JADE utilisent des messages conformes aux spécifications de la FIPA (**FIPA-ACL**) .
- les messages JADE sont des instances de la classe **ACLMensaje** du package jade.lang.acl.
- Ces messages sont composés en général de :
  - **L'émetteur du message** : un champ rempli automatiquement lors de l'envoi d'un message.
  - **L'ensemble des récepteurs du message** : un message peut être envoyé à plusieurs agents simultanément.
  - **L'acte de communication** : qui représente le but de l'envoi du message en cours (informer l'agent récepteur, appel d'offre, réponse à une requête,...)
  - **Le contenu du message.**
  - **Un ensemble de champs facultatifs**, comme la langue utilisée, l'ontologie, le timeOut, l'adresse de réponse...

# L'envoi d'un message

- Pour envoyer un message, il suffit de remplir les champs nécessaires (l'ensemble des récepteur et le contenu du message et l'acte de communication) d'un message JADE, puis d'appeler la méthode send() de la classe Agent. Voici un exemple d'envoi d'un message JADE.

- `ACLMessage message = new ACLMessage(ACLMessage.INFORM);`
- `message.addReceiver(new AID("vendeu1", AID.ISLOCALNAME));`
- `message.setContent("Livre XML");`
- `send(message);`



# Reception d'un message

- La réception d'un message est aussi simple que l'envoi.
- Il suffit d'appeler la méthode `receive()` de la classe `Agent` pour récupérer le premier message non encore lu de l'agent.
- Exemple :

```
• ACLMessage msg = receive();  
• // Pour envoyer la réponse :  
• ACLMessage message = new ACLMessage(ACLMessage.INFORM);  
• message.addReceiver(msg.getSender());  
• message.setContent("400");  
• send(message);
```

# L'attente d'un message

- Il se peut qu'un agent doive effectuer un certain traitement ou lancer quelques tâches après avoir reçu un message d'un autre agent.
- Il est possible de faire une attente active jusqu'à l'arrivée du message en utilisant la méthode block() :
- Exemple :

```
addBehaviour(new CyclicBehaviour() {  
    @Override  
    public void action() {  
        System.out.println("Cyclic behavior");  
        ACLMessage msg=receive();  
        if(msg!=null){  
            System.out.println("Réception du message :" +msg.getContent());  
        }  
        else{  
            block();  
        }  
    }  
});
```

# Exemple de messages

- Envoyer un message à tous les vendeurs pour leur demander de proposer une offre concernant un livre.
- Ici nous utilisons un message dont l'attribut **performative** est **CFP** (Call For Proposal)

```
// Message carrying a request for offer  
ACLMensaje cfp = new ACLMensaje(ACLMensaje.CFP);  
for (int i = 0; i < sellerAgents.length; ++i) {  
    cfp.addReceiver(sellerAgents[i]);  
}  
cfp.setContent(targetBookTitle);  
myAgent.send(cfp);
```



# Filtrer les messages : MessageTemplate

- Plusieurs messages peuvent arriver dans la file d'attente.
- Plusieurs agents tentent d'accéder aux messages qui arrivent,
- il est important de disposer d'un moyen qui permet à un agent de filtrer les messages qui le concernent dans un contexte quelconque.
- Les Templates de messages permettent de résoudre ce problème.



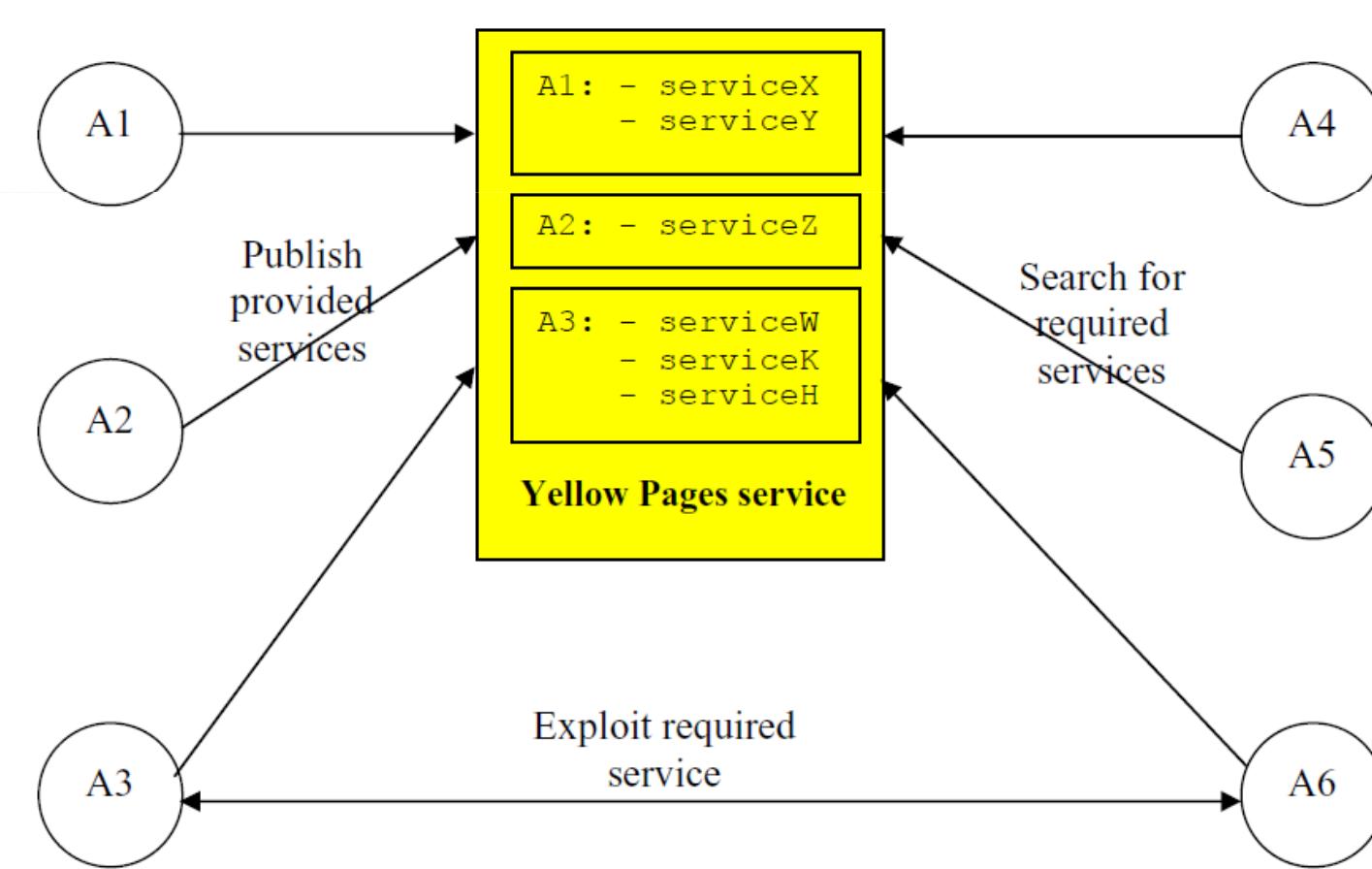
# MessageTemplate

- La classe MessageTemplate dispose d'un ensemble de méthodes statiques de fabrique de d'objet MessageTemplate selon différents critères.
- Exemples :
  - Ignorer tous les messages sauf ceux qui ont l'option performative est CFP

```
addBehaviour(new CyclicBehaviour() {  
    @Override  
    public void action() {  
        MessageTemplate mt =  
            MessageTemplate.MatchPerformativ(ACLM message.CFP);  
        ACLMessage msg = myAgent.receive(mt);  
        if (msg != null) {  
            // CFP Message received. Process it  
  
        }  
        else {  
            block();  
        }  
    }  
}
```

# DF Agent

- Le service des pages jaunes permet aux agents de la plateforme de publier leurs services.
- D'autres agents de la plateforme peuvent découvrir ces services à partir de ce service de pages jaunes.
- Dans JADE, Le service des pages jaunes est assurée par un agent DF (**Directory Facilitator**) en respectant les spécification de la FIPA.





# Interactions avec DF agent

- Les agents peuvent communiquer avec DFAgent en envoyant des messages ACL
- Le contenu de ces messages doivent respecter la norme (SL0 language) et une ontologie propre à ce service ( FIPA-Agent-Management Ontology)
- Pour simplifier ces interactions, JADE a défini la classe **jade.domain.DFService** avec laquelle, il est possible de publier et de chercher des services simplifiant ainsi les interactions avec DFAgent.



# Publication des services

- Un agent qui souhaite publier un service, doit fournir une description qui incluse :
  - Son identifiant AID
  - Liste des langages et des ontologies que les autres agent doivent utiliser pour communiquer avec lui
  - Liste des services publiés
  - Pour chaque service publié, on doit indiquer :
    - Le type du service
    - Le nom du service
    - Les langages et les ontologies à utiliser pour exploiter ce service
    - Et d'autres propriétés spécifiques.
- Les classes
  - **DFAgentDescription**,
  - **ServiceDescription**
  - et **Property**
- du package **jade.domain.FIPAManagement** représentent les trois abstractions mentionnées.

# Publication des services

```
@Override  
protected void setup() {  
    // Register the book-selling service in the yellow pages  
    DFAgentDescription dfd = new DFAgentDescription();  
    dfd.setName(getAID());  
    ServiceDescription sd = new ServiceDescription();  
    sd.setType("book-selling");  
    sd.setName("JADE-book-trading");  
    dfd.addServices(sd);  
    try {  
        DFService.register(this, dfd);  
    }  
    catch (FIPAException fe) {  
        fe.printStackTrace();  
    }  
}
```

# Suppression d'un service publié

- Quand un agent termine son travail, il est important de supprimer ses services des pages jaunes.
- Généralement cette opération est effectuée dans la méthode `takeDown()`

```
@Override  
protected void takeDown() {  
    // Deregister from the yellow pages  
    try {  
        DFService.deregister(this);  
    }  
    catch (FIPAException fe) {  
        fe.printStackTrace();  
    }  
}
```

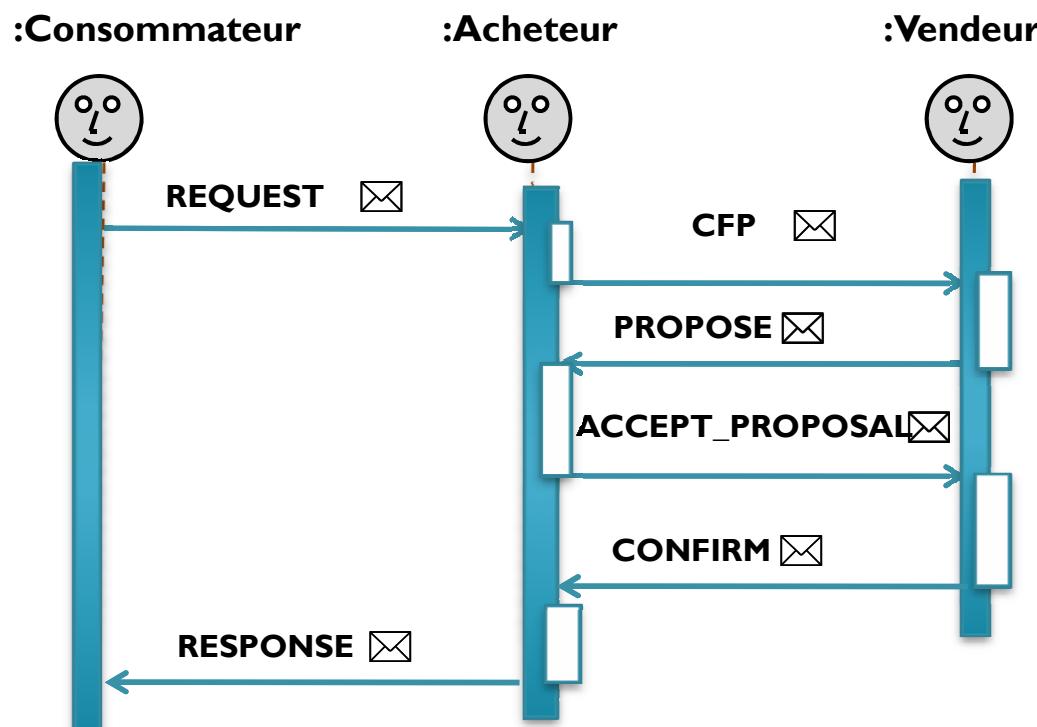
# Chercher des services

- Un agent qui souhaite chercher des services doit fournir au DF, une template de description qui décrit les services souhaités.
- Le résultat de la recherche est une liste de descriptions de services.
- La méthode statique search() de la classe DFService peut être utilisée pour ce fait.

```
addBehaviour(new TickerBehaviour(this, 60000) {  
    protected void onTick() {  
        // Update the list of seller agents  
        DFAgentDescription template = new DFAgentDescription();  
        ServiceDescription sd = new ServiceDescription();  
        sd.setType("book-selling"); template.addServices(sd);  
        try {  
            DFAgentDescription[] result = DFService.search(myAgent, template);  
            AID[] sellerAgents = new AID[result.length];  
            for (int i = 0; i < result.length; ++i) {  
                sellerAgents[i] = result[i].getName();  
            }  
        }  
        catch (FIPAException fe) { fe.printStackTrace(); }  
    }  
}
```

# Première Application

- Un agent consommateur demande à l'agent Acheteur d'acheter un livre
- L'agent acheteur demande à l'agent vendeur de lui donner son offre
- L'agent acheteur accepte l'offre
- Et la transaction est conclue



# BookBuyerAgent

```
package agents; import jade.core.AID; import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour; import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
public class BookBuyerAgent extends Agent {
private int compteur;
@Override
protected void setup() {
System.out.println("Salut je suis l'agent Acheteur mon nom
est:"+this.getAID().getName());
addBehaviour(new CyclicBehaviour() {
private AID requester; private String livre; private double prix;
@Override
public void action() {
try {
MessageTemplate template=
MessageTemplate.or(
MessageTemplate.MatchPerformative(ACLMessage.PROPOSE),
MessageTemplate.or(
MessageTemplate.MatchPerformative(ACLMessage.CONFIRM),
MessageTemplate.MatchPerformative(ACLMessage.REQUEST))
);
});
```

# BookBuyerAgent

```
ACLMessage aclMessage=receive(template);
if(aclMessage!=null){
switch(aclMessage.getPerformative()){
case ACLMessage.REQUEST :
++compteur;
System.out.println("#####");
System.out.println("Requête d'achat de Livre:");
System.out.println("From :" +aclMessage.getSender().getName());
livre=aclMessage.getContent();
requester=aclMessage.getSender();
System.out.println("Livre : "+livre);
System.out.println(".....");
System.out.println("Envoi de la requête....");
ACLMessage msg=new ACLMessage(ACLMessage.CFP);
msg.setContent(livre);
msg.setConversationId(livre+"-"+compteur);
msg.addUserDefinedParameter("compteur", String.valueOf(compteur));
msg.addReceiver(new AID("Vendeur1",AID.ISLOCALNAME));
System.out.println("..... En cours");
Thread.sleep(5000); send(msg);
break;
```



# BookBuyerAgent

```
case ACLMessage.PROPOSE :  
    prix=Double.parseDouble(aclMessage.getContent());  
    System.out.println("*****");  
    System.out.println("Conversation ID:"+aclMessage.getConversationId());  
    System.out.println("Réception de l'offre :");  
    System.out.println("From :" +aclMessage.getSender().getName());  
    System.out.println("Prix="+prix);  
    System.out.println("-----");  
    System.out.println("Conclusion de la transaction.....");  
    ACLMessage aclMessage2=aclMessage.createReply();  
    aclMessage2.setPerformative(ACLMessage.ACCEPT_PROPOSAL);  
    System.out.println("..... En cours");  
    Thread.sleep(5000);  
    send(aclMessage2);  
    break;
```

# BookBuyerAgent

```
case ACLMessage.CONFIRM:  
    System.out.println(".....");  
    System.out.println("Reçu de La confirmation ...");  
    System.out.println("Conversation ID:"+aclMessage.getConversationId());  
    ACLMessage msg3=new ACLMessage(ACLMessage.INFORM);  
    msg3.addReceiver(requester);  
    msg3.setConversationId(aclMessage.getConversationId());  
    msg3.setContent("<transaction>"  
        + "<livre>"+livre+"</livre>"  
        + "<prix>"+prix+"</prix>"  
        + "<fournisseur>"+aclMessage.getSender().getName()+"</fournisseur>"  
        + "</transaction>");  
    send(msg3);  
    break;  
}  
}  
  
else{  
    block();  
}  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

# BookSellerAgent

```
package agents;

import java.util.HashMap; import java.util.Map; import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour; import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class BookSellerAgent extends Agent {
    private Map<String, Double> data=new HashMap<>();
    @Override
    protected void setup() {
        data.put("XML", new Double(230)); data.put("JAVA", new Double(460));
        data.put("IOT", new Double(540));
        System.out.println("..... Vendeur "+this.getAID().getName());
        addBehaviour(new CyclicBehaviour() {
            @Override
            public void action() {
                try {
                    MessageTemplate messageTemplate= MessageTemplate.or(
                        MessageTemplate.MatchPerformativ(ACLMessage.CFP),
                        MessageTemplate.MatchPerformativ(ACLMessage.ACCEPT_PROPOSAL));
                    ACLMessage aclMessage=receive(messageTemplate);
                }
            }
        });
    }
}
```

# BookSellerAgent

```
if(aclMessage!=null){  
    switch(aclMessage.getPerformative()){  
        case ACLMessage.CFP :  
            System.out.println("-----");  
            System.out.println("Conversation ID:"+aclMessage.getConversationId());  
            String livre=aclMessage.getContent();  
            String compteur=aclMessage.getUserDefinedParameter("compteur");  
            System.out.println("Réception d'un message :"+compteur);  
            System.out.println("Expéditeur :"+aclMessage.getSender().getName());  
            System.out.println("Contenu:"+livre);  
            System.out.println("-----");  
            Double prix=data.get(livre);  
            ACLMessage reply=aclMessage.createReply();  
            reply.setPerformative(ACLMessage.PROPOSE);  
            reply.setContent(prix.toString());  
            System.out.println("..... En cours");  
            Thread.sleep(5000);  
            send(reply);  
        break;  
    }  
}
```

# BookSellerAgent

```
case ACLMessage.ACCEPT_PROPOSAL:  
System.out.println("-----");  
System.out.println("Conversation ID:"+aclMessage.getConversationId());  
System.out.println("Validation de la transaction .....");  
ACLMessage reply2=aclMessage.createReply();  
reply2.setPerformative(ACLMessage.CONFIRM);  
System.out.println("..... En cours");  
Thread.sleep(5000);  
send(reply2);  
break;  
} }  
else{  
    System.out.println("Block");  
    block();  
}} catch (Exception e) {e.printStackTrace(); }}});  
}  
@Override  
protected void takeDown() {  
}  
}
```

# MainContainer

```
import jade.core.Profile; import jade.core.ProfileImpl;
import jade.core.Runtime; import jade.util.ExtendedProperties;
import jade.util.leap.Properties; import jade.wrapper.AgentContainer;

public class MainContainner {
    public static void main(String[] args) {
        try{
            Runtime runtime=Runtime.instance();
            Properties properties=new ExtendedProperties();
            properties.setProperty(Profile.GUI, "true");
            ProfileImpl profileImpl=new ProfileImpl(properties);
            AgentContainer mainContainer=runtime.createMainContainer(profileImpl);
            mainContainer.start();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

# BookBuyerContainer

```
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
public class BookBuyerContainer {
    public static void main(String[] args) {
        try {
            Runtime runtime=Runtime.instance();
            ProfileImpl profileImpl=new ProfileImpl(false);
            profileImpl.setParameter(ProfileImpl.MAIN_HOST, "localhost");
            AgentContainer agentContainer=runtime.createAgentContainer(profileImpl);
            AgentController agentController=agentContainer.createNewAgent("acheteur1",
                "agents.BookBuyerAgent", new Object[]{});
            agentController.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

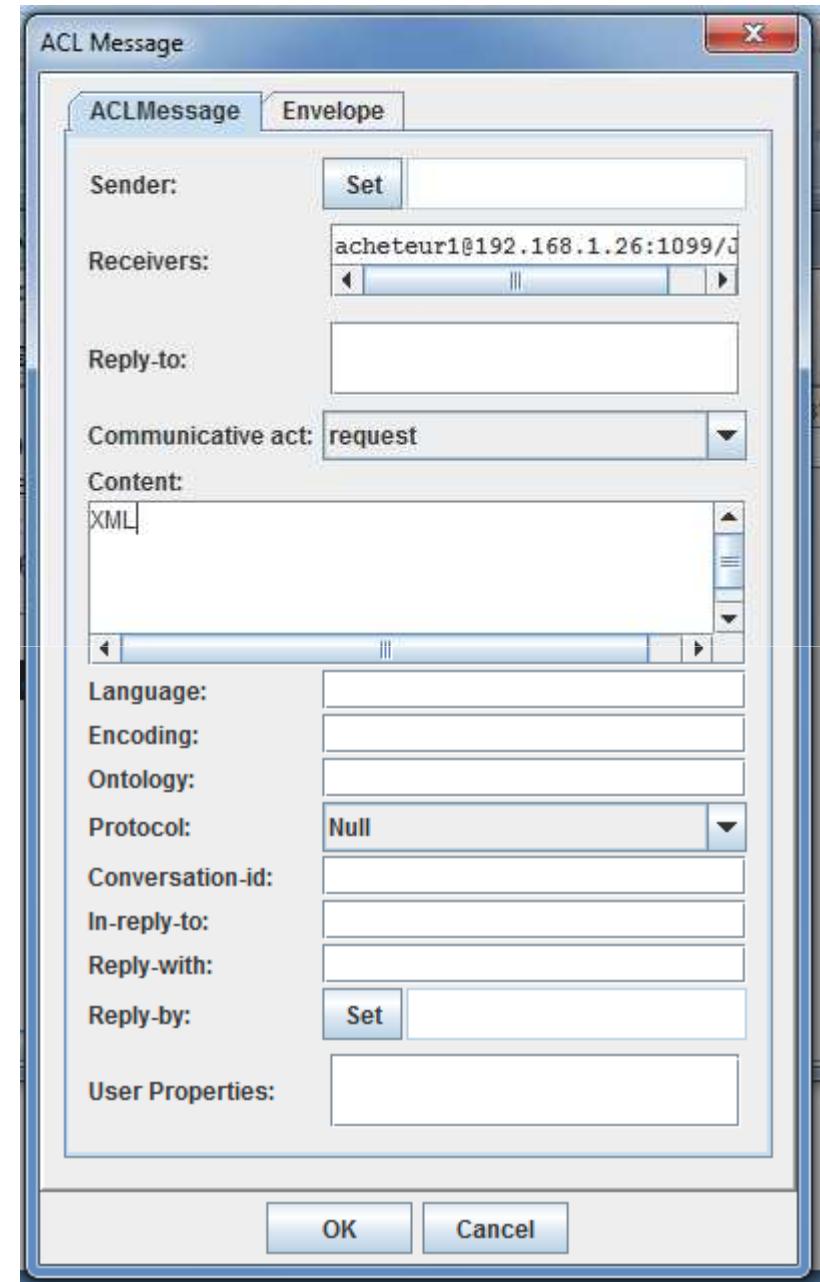
# BookSellerContainer

```
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
public class BookSellerContainer {
    public static void main(String[] args) {
        try {
            Runtime runtime=Runtime.instance();
            ProfileImpl profileImpl=new ProfileImpl(false);
            profileImpl.setParameter(ProfileImpl.MAIN_HOST, "localhost");
            AgentContainer agentContainer=runtime.createAgentContainer(profileImpl);
            AgentController
            agentController=agentContainer.createNewAgent("Vendeur1",
                "agents.BookSellerAgent", new Object[]{});
            agentController.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

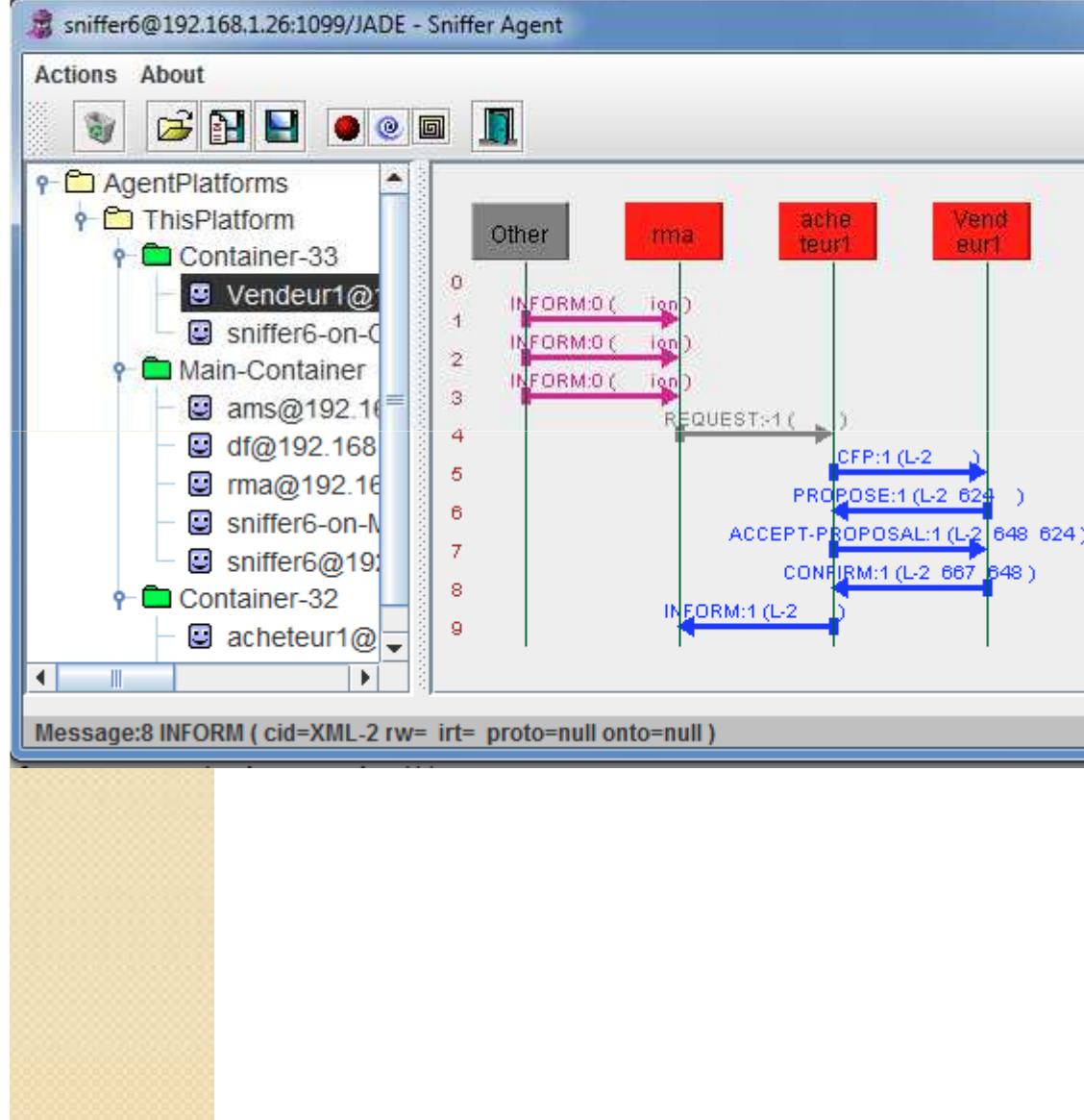
# Déploiement

```
c:\Windows\system32\cmd.exe - java BookSellerContainer
.... Vendeur Vendeur1@192.168.1.26:1099/JADE
Block
-----
Conversation ID:XML-1
RÉception d'un message :1
ExpÜditeur :acheteur1@192.168.1.26:1099/JADE
Contenu:XML
-----
... En cours
Block
-----
Conversation ID:XML-1
Validation de la transaction .....
... En cours
Block
-----
Conversation ID:XML-2
RÉception d'un message :2
ExpÜditeur :acheteur1@192.168.1.26:1099/JADE
Contenu:XML
-----
... En cours
Block
-----
Conversation ID:XML-2
Validation de la transaction .....
... En cours
Block

c:\Windows\system32\cmd.exe - java BookBuyerContainer
Conversation ID:XML-1
#####
RequÔte d'achat de livre:
From :rma@192.168.1.26:1099/JADE
Livre : XML
Envoi de la requÔte....
... En cours
#####
Conversation ID:XML-2
RÉception de l'offre :
From :Vendeur1@192.168.1.26:1099/JADE
Prix=230.0
-----
Conclusion de la transaction.....
... En cours
Répu de la confirmation...
Conversation ID:XML-2
```



# Image de la conversation



ACL Message

ACLMessages Envelope

Sender:  acheteur1@192.168.1.26:1099/JADE

Receivers:

Reply-to:

Communicative act:

Content:  
<transaction><livre>XML</livre><prix>XML</prix><fournis>

Language:

Encoding:

Ontology:

Protocol:

Conversation-id: XML-2

In-reply-to:

Reply-with:

Reply-by:

User Properties:

OK



# Application 2

- Créer un SMA composés des agents suivants :
  - Des agents de type BookSellerAgent qui ont pour objectif de vendre des livres aux agents qui souhaitent acheter.
  - Un agent de type BookBuyerAgent qui a pour objectif d'acheter un livre avec le meilleur prix du marché.
- Le SMA fonctionne de la manière suivante :
- **BookSeller** :
  - À sa création un agent BookSeller doit publier un service dans les pages jaunes. Ce service concerne la vente des livres.
  - Chaque agent BookSeller possède une collection de type `HashMap` dont la clé représente le titre du livre et dont la valeur représente le prix du livre. Les prix seront choisi aléatoirement

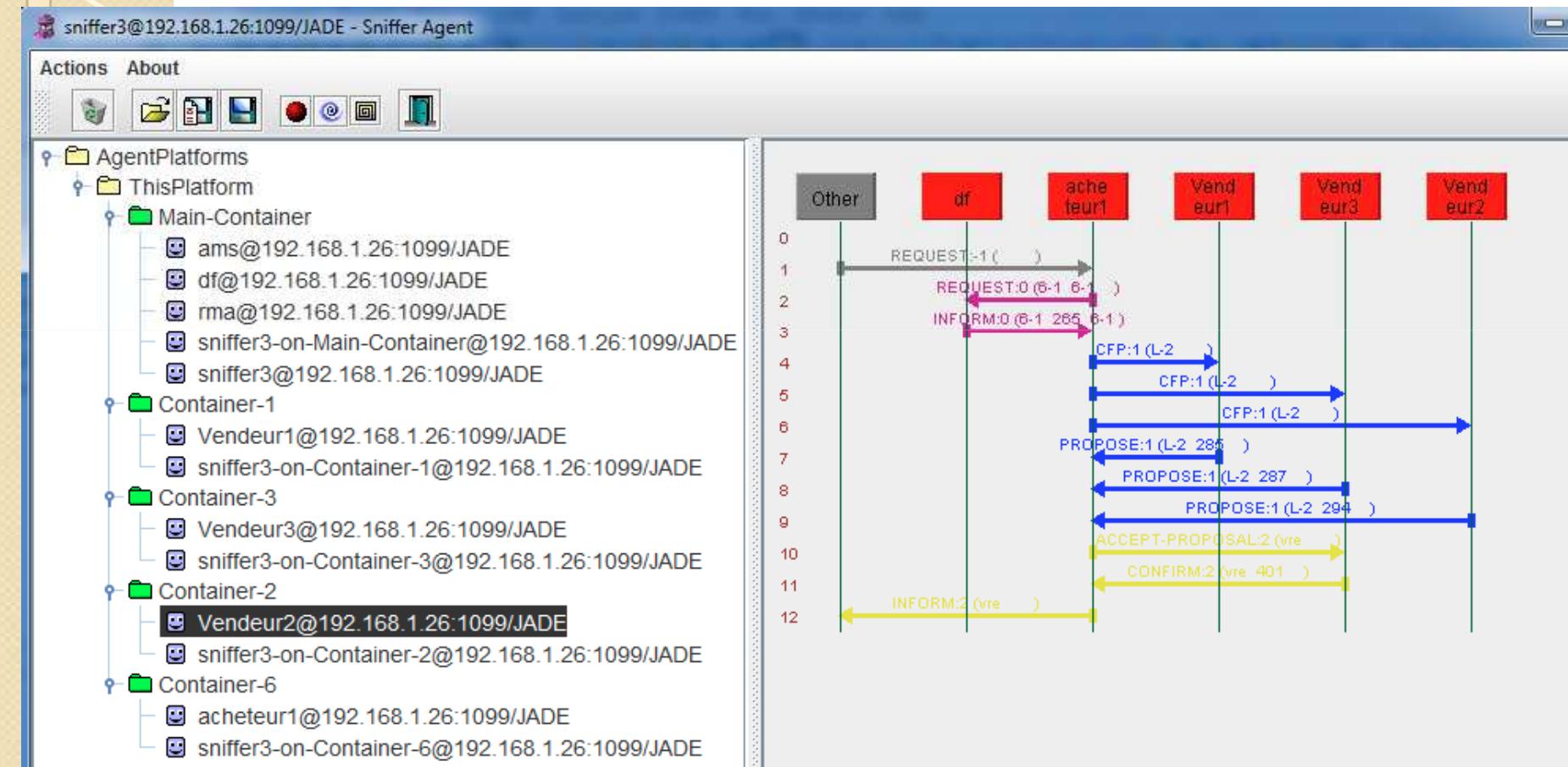


# application

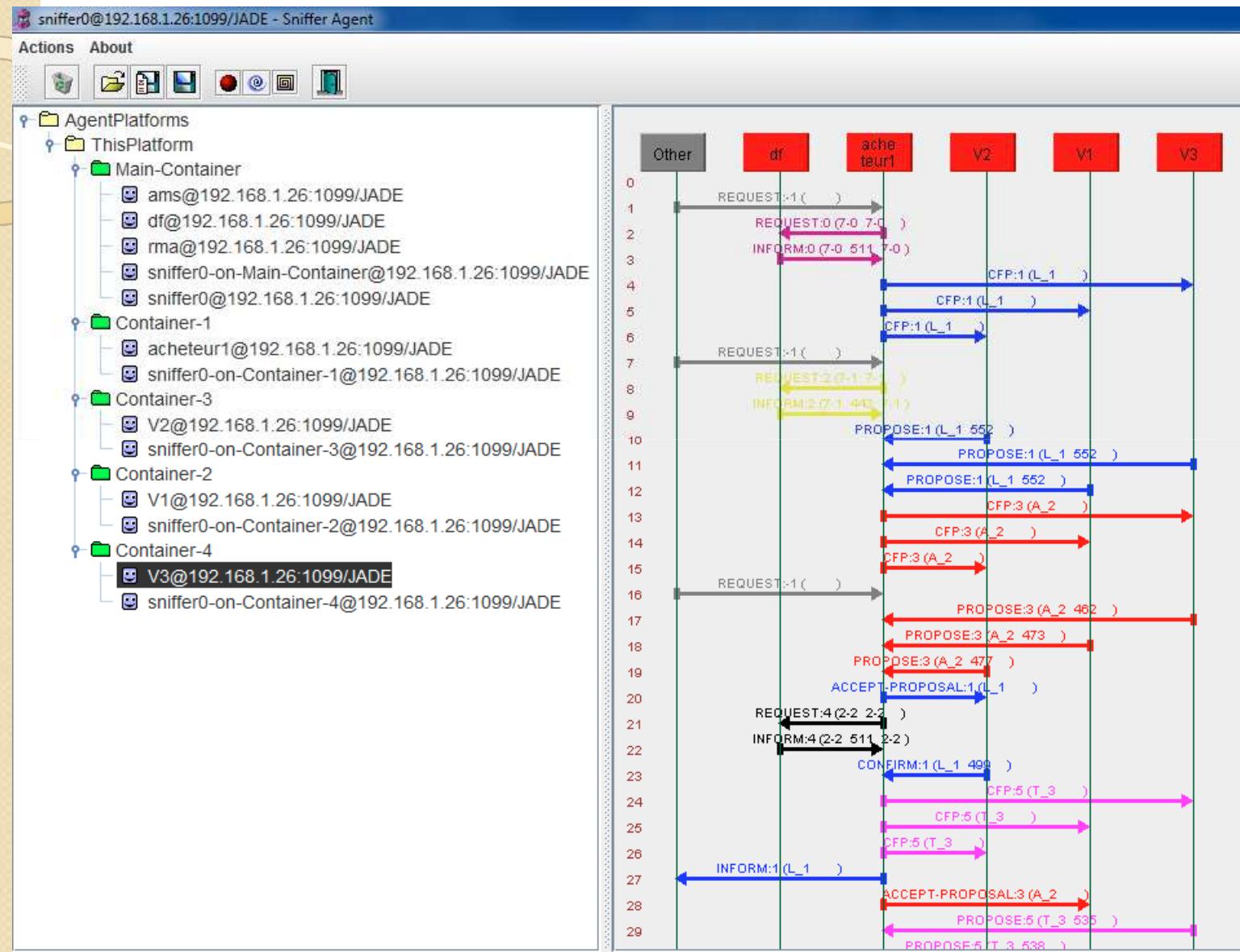
- **BookBuyer :**

- À la demande d'un agent consommateur, l'agent de type BookBuyer commence par récupérer le titre du livre à acheter, passé comme argument au moment du déploiement
- Ensuite, chaque minute,
  - il tente de chercher dans le service des pages jaunes DF, les agents qui offrent le service de vente de livres.
- Ensuite, il envoie un message de type CFP (Create For Proposal ) à tous ces agents sellers, pour leurs demander de lui fournir la proposition du prix du livre souhaité.
- Si les propositions arrivent, il doit déterminer l'agent qui offre le meilleur prix.
- Ensuite il envoie un message à cet agent pour finaliser la vente.

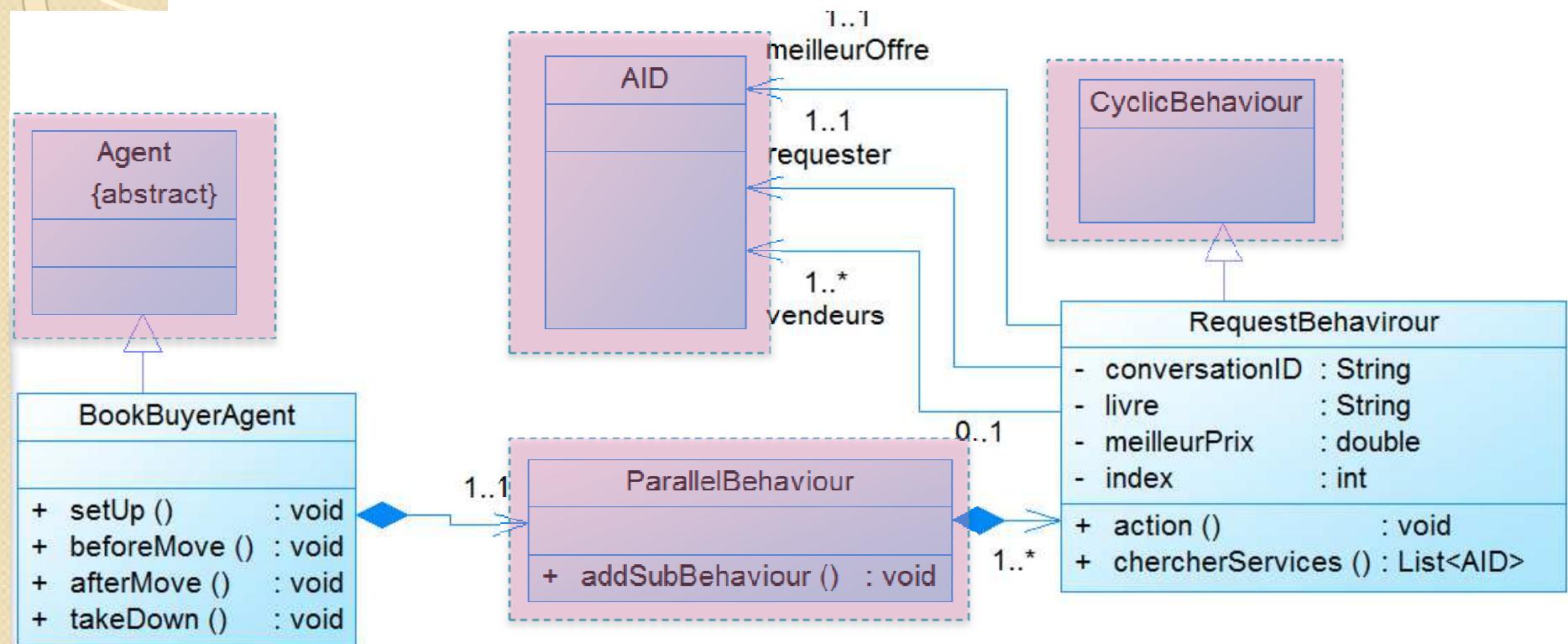
# Diagramme d'interactions



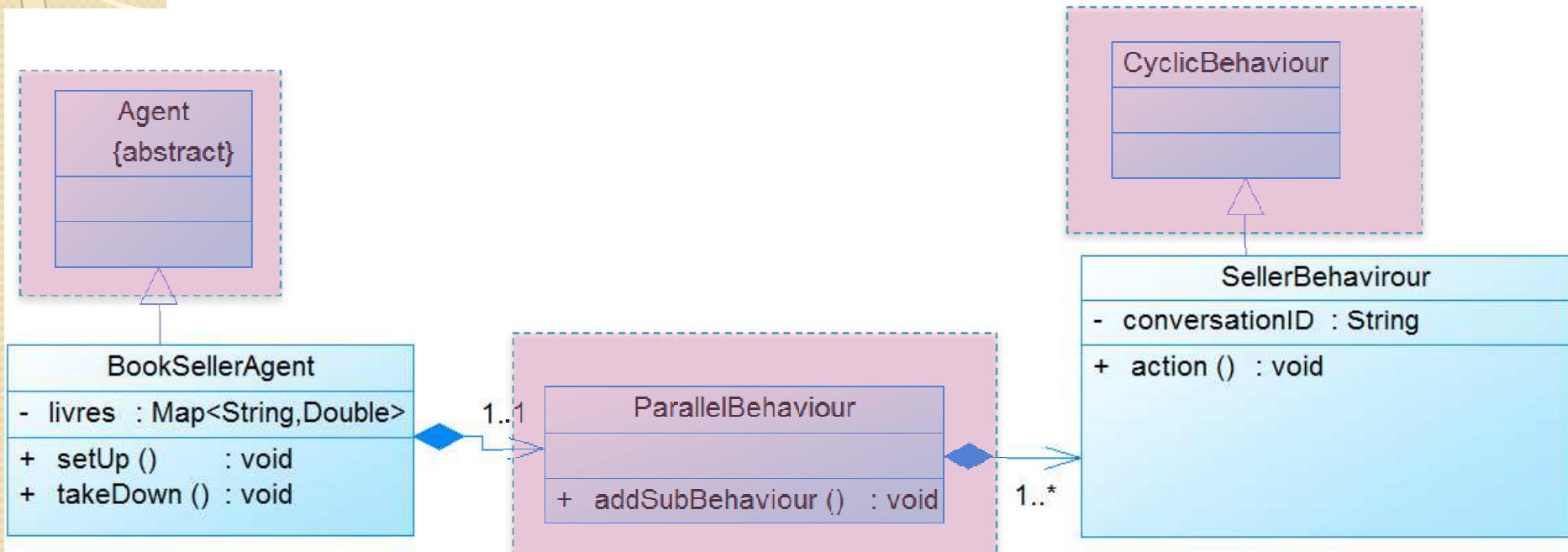
# 3 demandes simultanée d'achat de livres



# Diagramme de classes : Acheteur



# Diagramme de classes :Vendeur



# BookSellerAgent

```
package agents; import java.util.HashMap;
import java.util.Map;import jade.core.*;
import jade.core.behaviours.*; import jade.domain.*;
import jade.domain.FIPAService.*;
import jade.lang.acl.*;

public class BookSellerAgent extends Agent {
private Map<String, Double> data=new HashMap<>();
private ParallelBehaviour parallelBehaviour;

@Override
protected void setup() {
    data.put("XML", new Double(230+Math.random()*200));
    data.put("JAVA", new Double(460+Math.random()*200));
    data.put("IOT", new Double(540+Math.random()*200));
    System.out.println("..... Vendeur "+this.getAID().getName());
    System.out.println("-----");
```

# BookSellerAgent

```
System.out.println("Publication du service dans Directory Facilitator...");  
DFAgentDescription agentDescription=new DFAgentDescription();  
agentDescription.setName(this.getAID());  
ServiceDescription serviceDescription=new ServiceDescription();  
serviceDescription.setType("book-selling");  
serviceDescription.setName("book-trading");  
agentDescription.addServices(serviceDescription);  
try {  
    DFService.register(this, agentDescription);  
} catch (FIPAException e1) {  
e1.printStackTrace();  
}
```



# BookSellerAgent

```
parallelBehaviour=new ParallelBehaviour();
addBehaviour(parallelBehaviour);
parallelBehaviour.addSubBehaviour(new CyclicBehaviour() {
    @Override
    public void action() {
        try {
            MessageTemplate messageTemplate=
                MessageTemplate.MatchPerformativ(ACLMessage.CFP);
            ACLMessage aclMessage=receive(messageTemplate);
            if(aclMessage!=null){
                System.out.println("Conversation ID:"+aclMessage.getConversationId());
                String livre=aclMessage.getContent();
                Double prix=data.get(livre);
                ACLMessage reply=aclMessage.createReply();
                reply.setPerformativ(ACLMessage.PROPOSE);
                reply.setContent(prix.toString());
                System.out.println("..... En cours");
                Thread.sleep(5000);send(reply);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```



# BookSellerAgent

```
parallelBehaviour.addSubBehaviour(new
    SellerBehaviour(myAgent,aclMessage.getConversationId())));
} else{ block(); }
} catch (Exception e) { e.printStackTrace(); }
});    }
@Override
protected void takeDown() {
try {
    DFService.deregister(this);
} catch (FIPAException e) {
    e.printStackTrace();
}
}
}
```



# SellerBehaviour

```
package agents;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
public class SellerBehaviour extends CyclicBehaviour {
    private String conversationID;
    public SellerBehaviour(Agent agent, String conversationID) {
        super(agent);
        this.conversationID=conversationID;
    }
    @Override
    public void action() {
        try {
            MessageTemplate messageTemplate= MessageTemplate.and(
                MessageTemplate.MatchConversationId(conversationID),
                MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL));
            ACLMessage aclMessage=myAgent.receive(messageTemplate);
```



# SellerBehaviour

```
if(aclMessage!=null){  
    System.out.println("-----");  
    System.out.println("Conversation ID:"+aclMessage.getConversationId());  
    System.out.println("Validation de la transaction .....");  
    ACLMessage reply2=aclMessage.createReply();  
    reply2.setPerformative(ACLMessage.CONFIRM);  
    System.out.println("..... En cours");  
    Thread.sleep(5000);  
    myAgent.send(reply2);  
}  
else{  
    block();  
}  
} catch (Exception e) {  
e.printStackTrace();  
}}}
```

# BookBuyerAgent

```
package agents; import jade.core.AID; import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour; import
jade.core.behaviours.ParallelBehaviour;
import jade.lang.acl.ACLMessage; import jade.lang.acl.MessageTemplate;
public class BookBuyerAgent extends Agent {
    ParallelBehaviour parallelBehaviour;
    int requesterCount;
    @Override
    protected void setup() {
        System.out.println("Salut je suis l'agent Acheteur mon nom
est:"+this.getAID().getName());
        parallelBehaviour=new ParallelBehaviour();
        addBehaviour(parallelBehaviour);
```

# BookBuyerAgent

```
parallelBehaviour.addSubBehaviour(new CyclicBehaviour() {  
    @Override  
    public void action() {  
        MessageTemplate  
        template=MessageTemplate.MatchPerformativ(ACLMessage.REQUEST);  
        ACLMessage aclMessage=receive(template);  
        if(aclMessage!=null){  
            String livre=aclMessage.getContent();  
            AID requester=aclMessage.getSender();  
            ++requesterCount;  
            String conversationID="transaction_"+livre+"_"+requesterCount;  
            parallelBehaviour.addSubBehaviour(  
                new RequestBehaviour(myAgent,livre,requester,conversationID));  
        }  
        else block();  
    }  
});  
}
```

# BookBuyerAgent

```
@Override  
protected void beforeMove() {  
System.out.println("Avant de migrer vers une nouvelle location  
.....");  
}  
  
@Override  
protected void afterMove() {  
System.out.println("Je viens d'arriver à une nouvelle location  
.....");  
}  
  
@Override  
protected void takeDown() {  
System.out.println("avant de mourir .....");  
}  
  
}
```

# RequestBehaviour

```
package agents; import jade.core.AID; import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour; import jade.domain.DFService;
import jade.domain.FIPAException; import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription; import jade.lang.acl.ACMessage;
import jade.lang.acl.MessageTemplate; import java.util.ArrayList; import java.util.List;
class RequestBehaviour extends CyclicBehaviour{
    private String conversationID;
    private AID requester; private String livre;
    private double prix; private int compteur;
    private List<AID> vendeurs=new ArrayList<>();
    private AID meilleureOffre;
    private double meilleurPrix;
    private int index;
```

# RequestBehaviour

```
public RequestBehaviour(Agent agent, String livre, AID requester, String conversationID) {  
    super(agent);  
    this.livre=livre; this.requester=requester;  
    this.conversationID=conversationID;  
    System.out.println("Recherche des services...");  
    vendeurs=chercherServices(myAgent, "book-selling");  
    System.out.println("Liste des vendeurs trouvés :");  
    try {  
        for(AID aid:vendeurs){  
            System.out.println("===="+aid.getName());  
        }  
        ++compteur;  
        System.out.println("#####");  
        System.out.println("Requête d'achat de livre:");  
        System.out.println("From : "+requester.getName());  
        System.out.println("Livre : "+livre);  
        System.out.println(".....");  
    }  
}
```

# RequestBehaviour

```
System.out.println("Envoi de la requête....");
ACLMensaje msg=new ACLMensaje(ACLMensaje.CFP);
msg.setContent(livre);
msg.setConversationId(conversationID);
msg.addUserDefinedParameter("compteur", String.valueOf(compteur));
for(AID aid:vendeurs){
    msg.addReceiver(aid);
}
System.out.println("..... En cours");
Thread.sleep(5000);
index=0;
myAgent.send(msg);
} catch (Exception e) {
e.printStackTrace();
}
}
```

# RequestBehaviour

```
@Override  
public void action() {  
    try {  
        MessageTemplate template=MessageTemplate.and(  
            MessageTemplate.MatchConversationId(conversationID),  
            MessageTemplate.or(  
                MessageTemplate.MatchPerformative(ACLMessage.PROPOSE),  
                MessageTemplate.MatchPerformative(ACLMessage.CONFIRM)));  
        ACLMessage aclMessage=myAgent.receive(template);  
        if(aclMessage!=null){  
            switch(aclMessage.getPerformative()){  
                case ACLMessage.PROPOSE :  
                    prix=Double.parseDouble(aclMessage.getContent());  
                    System.out.println("*****");  
                    System.out.println("Conversation ID:"+aclMessage.getConversationId());  
                    System.out.println("Réception de l'offre :");  
                    System.out.println("From :"+aclMessage.getSender().getName());  
                    System.out.println("Prix="+prix);  
            }  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

# RequestBehaviour

```
if(index==0){  
    meilleurPrix=prix; meilleureOffre=aclMessage.getSender();  
}  
else{  
    if(prix<meilleurPrix){  
        meilleurPrix=prix;  
        meilleureOffre=aclMessage.getSender();  
    } }  
++index;  
if(index==vendeurs.size()){  
    index=0; System.out.println("-----");  
    System.out.println("Conclusion de la transaction.....");  
    ACLMessage aclMessage2=new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);  
    aclMessage2.addReceiver(meilleureOffre);  
    aclMessage2.setConversationId(conversationID);  
    System.out.println("..... En cours"); Thread.sleep(5000);  
    myAgent.send(aclMessage2);  
}  
break;
```

# RequestBehaviour

```
case ACLMessage.CONFIRM:  
    System.out.println(".....");  
    System.out.println("Reçu de la confirmation ...");  
    System.out.println("Conversation ID:" + aclMessage.getConversationId());  
    ACLMessage msg3 = new ACLMessage(ACLMessage.INFORM);  
    msg3.addReceiver(requester);  
    msg3.setConversationId(conversationID);  
    msg3.setContent("<transaction>"  
        + "<livre>" + livre + "</livre>"  
        + "<prix>" + meilleurPrix + "</prix>"  
        + "<fournisseur>" + aclMessage.getSender().getName() + "</fournisseur>"  
        + "</transaction>");  
    myAgent.send(msg3);  
    break;  
} }  
else{ block(); }  
} catch (Exception e) { e.printStackTrace(); }  
}
```

# RequestBehaviour

```
public List<AID> chercherServices(Agent agent, String type){  
    List<AID> vendeurs=new ArrayList<>();  
    DFAgentDescription agentDescription=new DFAgentDescription();  
    ServiceDescription serviceDescription=new ServiceDescription();  
    serviceDescription.setType(type);  
    agentDescription.addServices(serviceDescription);  
    try {  
        DFAgentDescription[] descriptions=DFService.search(agent, agentDescription);  
        for(DFAgentDescription dfad:descriptions){  
            vendeurs.add(dfad.getName());  
        }  
    } catch (FIPAException e) {  
        e.printStackTrace();  
    }  
    return vendeurs;  
}
```

# BookSellerContainer

```
import java.util.Scanner;
import jade.core.ProfileImpl;import jade.core.Runtime;
import jade.wrapper.AgentContainer; import jade.wrapper.AgentController;
public class BookSellerContainer {
    public static void main(String[] args) {
        try {
            Scanner clavier=new Scanner(System.in);
            System.out.print("Nom du vendeur:");
            String name=clavier.next();
            Runtime runtime=Runtime.instance();
            ProfileImpl profileImpl=new ProfileImpl(false);
            profileImpl.setParameter(ProfileImpl.MAIN_HOST, "localhost");
            AgentContainer agentContainer=runtime.createAgentContainer(profileImpl);
            AgentController agentController=agentContainer.createNewAgent(name,
                "agents.BookSellerAgent", new Object[]{});
            agentController.start();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```