Bases de l'Intelligence Artificielle Distribuée

Environnements de développement des SMA

Mr. KHEBBACHE Mohib Eddine

2ème année SDIA

2016/2017

kmohibeddine@yahoo.fr

Plan

Plan

Plan

Plan

Plan

- **Langage Machine** : Chaque famille de CPU possède son propre jeu d'instructions
- **Assembleur** : 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 $\Longrightarrow$ SOAP et REST
- **Programmation Orientée Agents** : Service + Intelligence + Apprentissage

- La programmation orientée agents a été proposée par Yoav Shoham en 1993 comme un nouveau paradigme de programmation,
- voir comme une spécialisation de la programmation orientée objets, où les agents sont les éléments centraux du langage
- mis l'accent sur la dimension sociale des agents dont les programmes développés, plusieurs agents peuvent interagissent
- Le langage de programmation proposé par Shoham comme démonstration de ce nouveau paradigme s'appelle AGENT0.

	POO	POA
Unité de base	objet	agent
Paramètres définissant l'état de l'unité de base	pas de contraintes	croyances, décisions, obligations, habiletés
Processus de calcul	envoi de messages et méthodes pour la réponse	envoi de messages et méthodes pour la réponse
Types de messages	pas de contraintes	informer, demander, offrir, promettre,accepter, rejeter
Contraintes sur les méthodes	pas de contraintes	consistance, vérité, ...

- une fois L'architecture de l'agent est décidée, on peut utiliser n'importe quel langage de programmation pour implémenter l'agent
- vue l'introduction de l'approche de la programmation orientée agents
- Les chercheurs ont proposé de nombreux langages pour la programmation des agents
- chaque langage favorisant une vision particulière de la notion d'agent intelligent.
- On ne peut pas dire qu'il y ait actuellement un tel langage généralement accepté.
- un langage de programmation agent peut être intégré dans une plate-forme multi-agent

- Les langages concurrents orientés objets peuvent être considérés comme un premier pas vers la programmation des agents : Actors, ABCL. . . .
- Langages pour les agents mobiles : TELESCRIPT, Agent-Tcl, Aglets

Plan

plates-formes SMA

- Pour construire un SMA il est préférable d'utiliser une plate-forme multi-agent
- qui offre un ensemble d'outils utilisé pour la construction, l'analyse, le test et la mise en service d'agents au sein d'un environnement spécifique.
  - Ces outils peuvent être sous la forme d'environnement de programmation (API) et d'applications
  - permettant d'aider le développeur à la programmation d'un SMA ainsi que son débogage.
- Il existe de nombreuses plates-formes pour le développement de SMA : JACK, Jadex, Madkit, JADE...etc.

Caractéristiques générales

- **Nom de la plate-forme** :
- **Auteurs** :
- **Etat de la plate-forme** :
  - Maquette – Prototype – Produit fini
  - Logiciel libre – Produit Commercial
- **Nombre d'applications développées**
  - des petits exemples – une application réelle – plusieurs applications réelles
- **Type d'application privilégié** :
  - Simulation – Résolution de problème
- **Configuration logicielle et matérielle**
- **Langages utilisés**
- **Systèmes d'exploitation**

- **CORMAS : (COmmon Resources Multi-Agent System)** est un framework de développement de systèmes multiagents, 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.
- **AnyLogic** : Logiciel de simulation multi-agents et multiméthode
- **NetLogo** : Logiciel de simulation multi-agents
- **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.
- **Jason** : est un environnement open source de développement d'agents dans le formalisme AgentSpeak, et développé en Java.
- **MadKit** : est une plate-forme multi-agents modulaire écrite en Java et construite autour du modèle organisationnel Agent/Gro C'est une plateforme libre basée sur la licence GPL/LGPL développée au sein du LIRMM.

Plan

plate forme JADE :*http ://jade.tilab.com/*

caractéristiques

- JADE est une plate-forme multi-agent crée par le laboratoire TILAB.
- C'est un framework qui permet le développement de systèmes multiagents et d'applications conformes aux normes FIPA (Foundation for Intelligent Physical Agents).
- Elle mis en œuvre dans le langage Java et fonctionne sous tous les systèmes d'exploitation
- JADE contient :
  - **Runtime Environment** : environnement d'exécution, où les agents peuvent vivre, qui doit être actif avant qu'un agent(s) peuvent être exécutées sur un hôte donné.
  - **librairie de classes** : que les développeurs utilisent pour écrire leurs agents
  - **suite d'outils graphiques** : qui facilitent la gestion et la supervision (le dépoilement et le débogage) de la plateforme des agents : rma (Remote agent Management) GUI

plate forme JADE :*http ://jade.tilab.com/*

rma (Remote agent Management) GUI

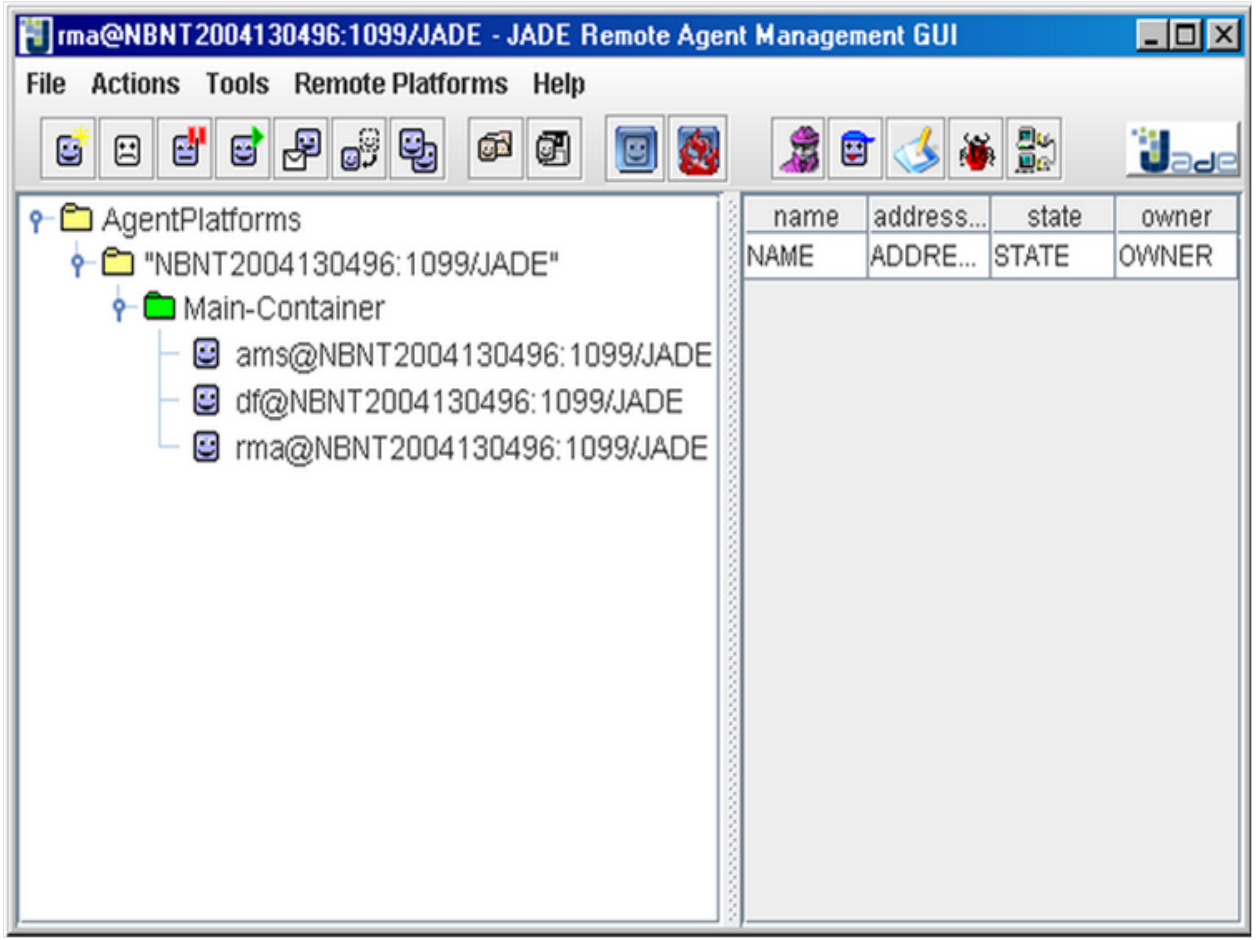
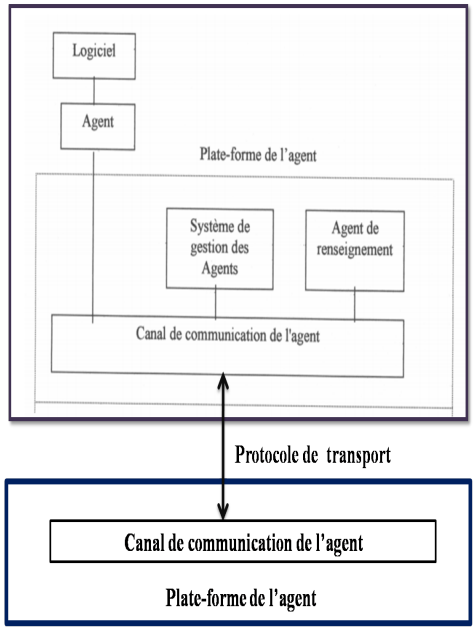


plate forme JADE :*http ://jade.tilab.com/*

caractéristiques

- Elle inclut tous les composants (agents de base) qui contrôlent un SMA (nécessaire aux spécifications FIPA).
  - **DF « Directory Facilitator »** agent de renseignement qui fournit un service de « pages jaunes» à la plate-forme ;
  - **ACC «Agent Communication Channel »** gère la communication entre les agents (situés dans la plate-forme ou résidant sur d'autres plates-formes).
  - **AMS « Agent Management System »** agent de gestion du système qui supervise l'enregistrement des agents (créé ou migré), leur authentification, leur accès et l'utilisation du système.

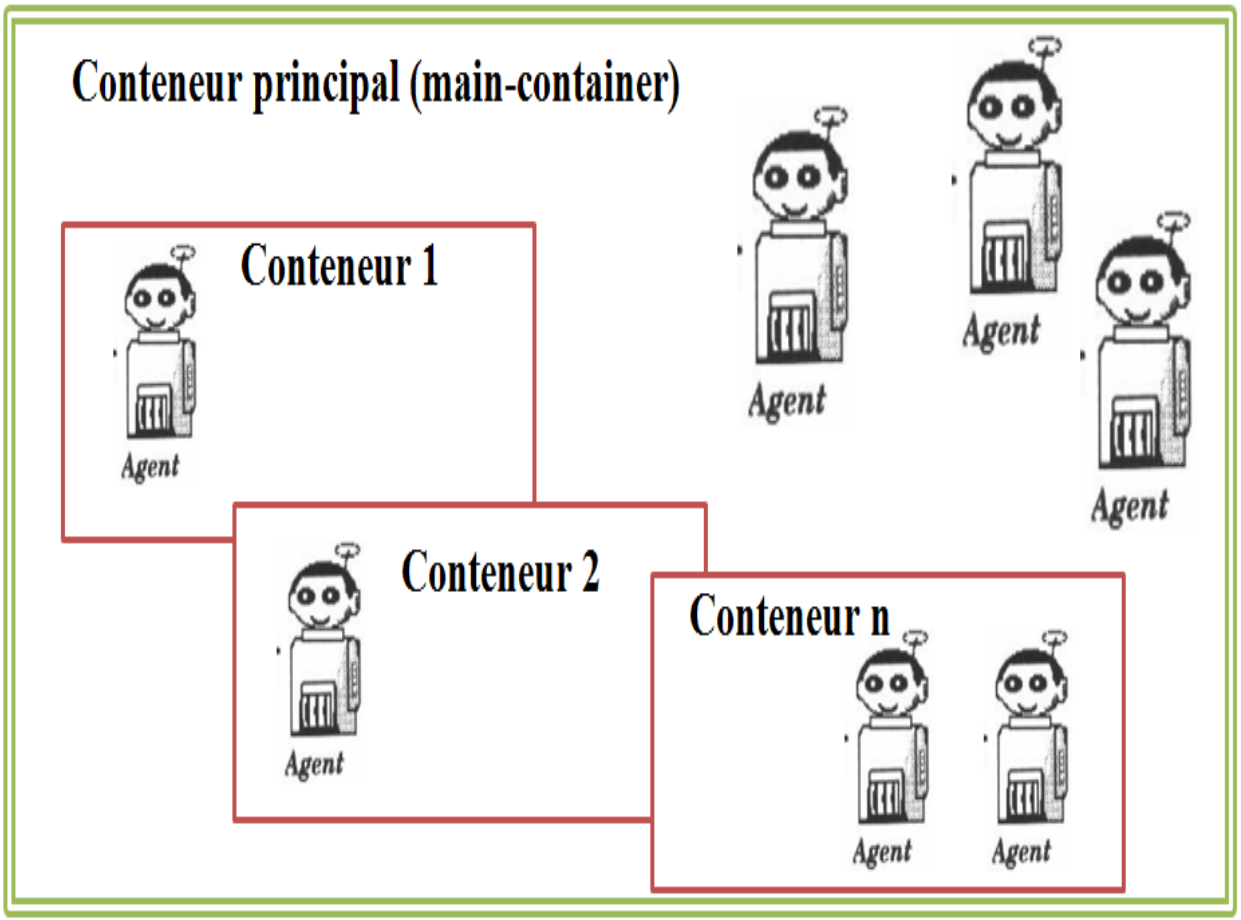


Ces trois modules sont activés automatiquement à chaque démarrage de la plate-forme.

- une plateforme est constituée par un ensemble de conteneurs
  - un conteneur spécial appelé main-container
    - Le conteneur principal comprend les agents AMS et le DF
  - les autres conteneurs s'enregistrent auprès de celui-là dés leur lancement via son URL(hôte et port).
  - chaque conteneur peut contenir plusieurs agents.
    - 1 thread per agent

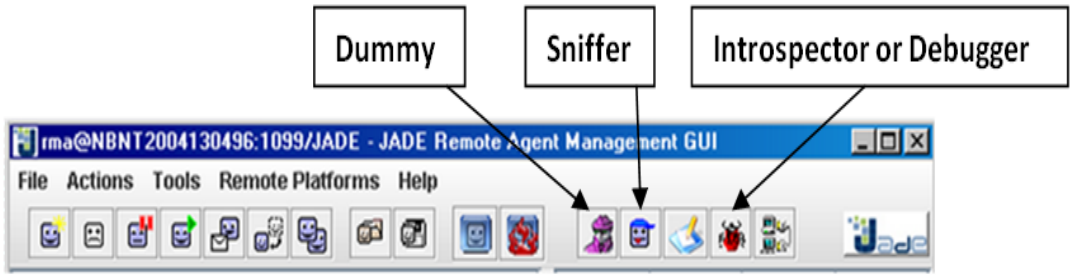
plate forme JADE :*http ://jade.tilab.com/*

caractéristiques



En plus la plateforme JADE contient des agents outils prés-implantés tels que :

- **l'agent** ≪ **Dummy** ≫ qui permet d'envoyer / recevoir des messages personnalisés, pour les stimuler.
- **L'agent** ≪ **Sniffer** ≫ qui permet l'inspection des conversations entre agents,
- **L'agent** ≪ **Introspector or Debugger** ≫ ou Agent débogueur qui permet d'inspecter le fonctionnement interne d'un agent (par exemple, les tâches qu'il est en cours d'exécution)
- **L'agent** ≪**Log Manager**≫ le gestionnaire de journaux qui permet de changer les niveaux de classes de journaux à l'exécution



- JADE fournit une couche homogène qui cache complètement la complexité et la diversité du réseau des agents
- Un système basé JADE peut être distribué sur plusieurs machines indépendamment de OS de celles-ci.
  - 1 JVM (machine virtuelle) par machine
  - plusieurs conteneurs principaux dans le réseau $\Longrightarrow$ plates-formes différentes
  - la configuration peut être
    - contrôlée via une interface graphique à distance $\ll$ remote GUI $\gg$.
    - changée au moment de l'exécution par la mobilité des agents d'une machine à une autre

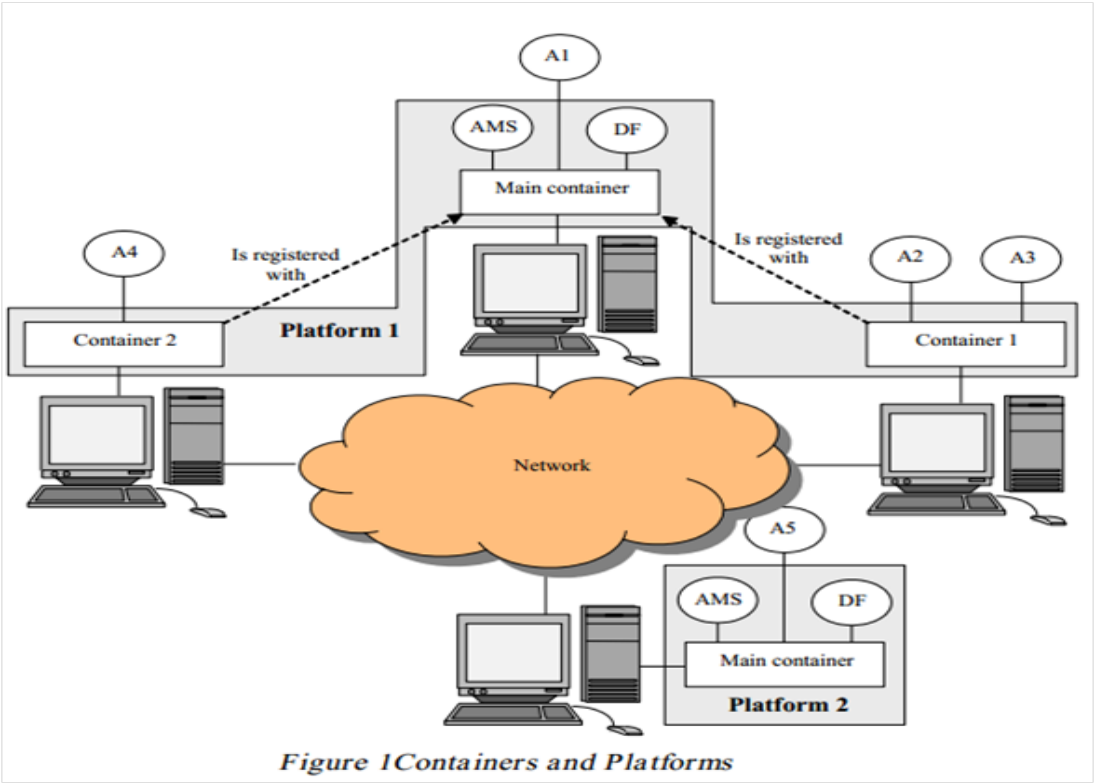
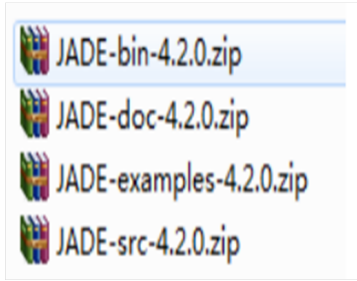


Figure 1Containers and Platforms

plate forme JADE :*http ://jade.tilab.com/*

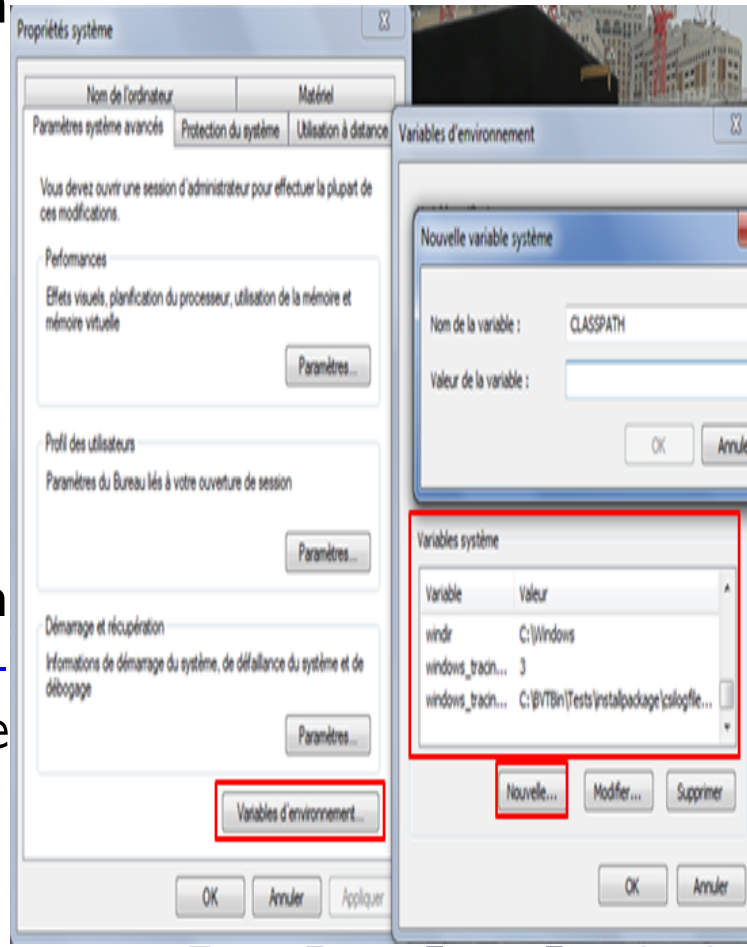
Configuration

1. Installation de JDK : *http ://java.sun.com/*
2. Installation de JADE : *http ://jade.tilab.com/*
   - Extraction du package JADE-ALL-x.x.x.ZIP téléchargé
   - Extraction de sous fichiers :
     - *JADE-doc* : Document
     - *JADE-src* : Code Source (api)
     - *JADE-bin* : Binary Code
     - *JADE-example* : Code source d'Examples
3. Réglage de classpath.
4. Installation de l'IDE : eclipse or NetBeans



Pour manipuler JADE sur ligne de commande, vous aurez besoin d'ajouter la variable d'environnement "classpath".

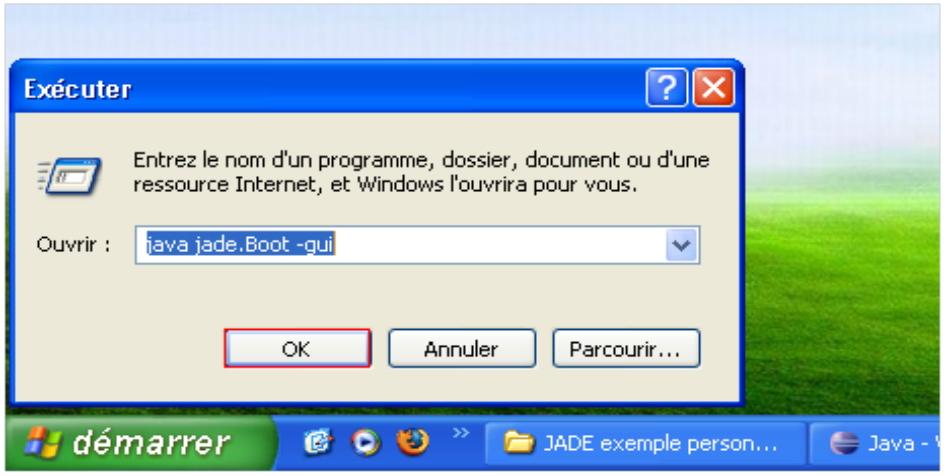
- Dans la zone variables système, essayez de trouver ou créer la variable d'environnement qui porte le nom CLASSPATH
  - sa valeur est la concaténation des chemins des quatre fichiers jar : http.jar, iiop.jar, jade.jar, jadeTools.jar
    - **Attention** : les chemins doivent être séparés par **des points Virgule(;)**
- Dans la même zone, vérifier la variable d'environnement "JAVA-HOME" ayant comme valeur le chemin de jdk
  - Exp :"C : Program Files



Lancer Jade avec la ligne de commandes

java jade.Boot [options] [agentlist]

- options
  - -container
  - -host HostName (default is localhost)
  - -port PortNumber (default is 1099)
  - -gui (Remote Monitoring Agent)
- agentlist : semicolon (;) separated agent identifiers
  - identifier = name :class-name(comma-sep-arguments)



Lancer Jade et la GUI

java jade.Boot –gui

Lancer un agent au démarrage

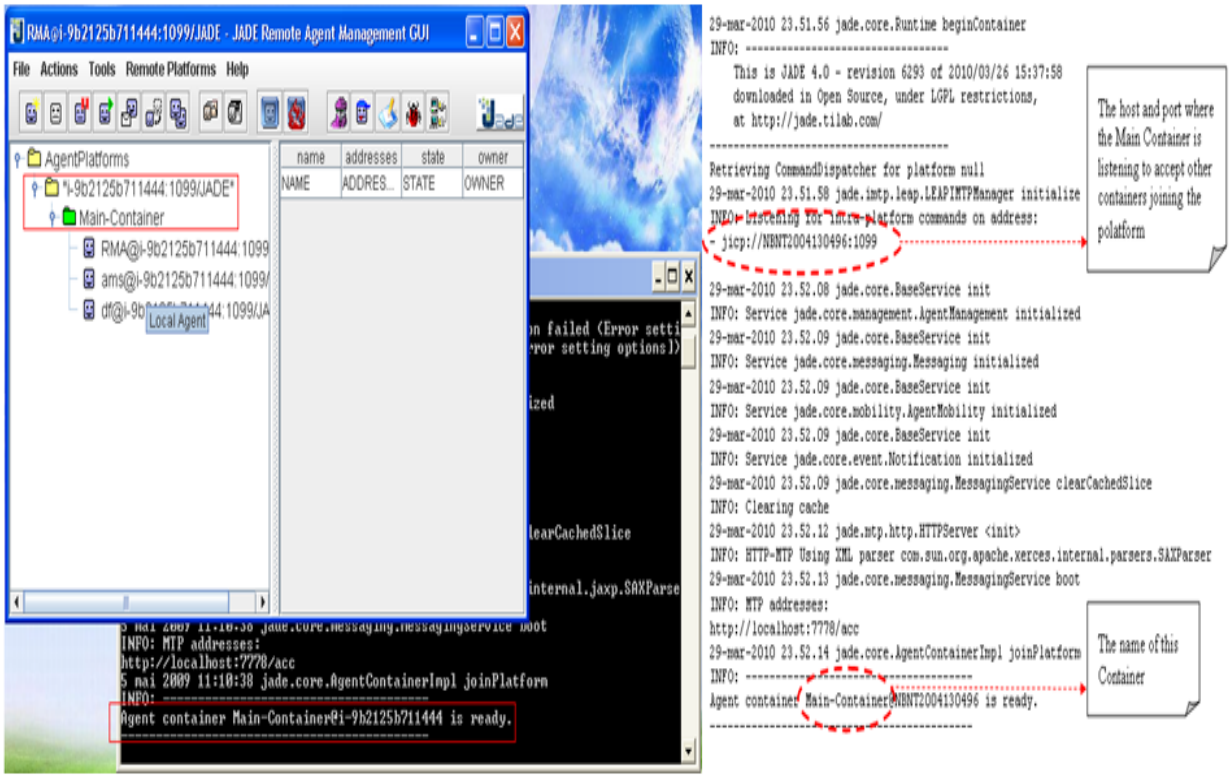
java jade.Boot –gui $< nomdel'agent >:< classedel'agent >$

Lancer un agent avec des paramètres

java jade.Boot –gui
$< nomdel'agent >:< classeagent > (< Param >)$

plate forme JADE :*http ://jade.tilab.com/*

Démarrage de la plateforme Jade



Création d'un agent

- Etendre la classe**jade.core.Agent**
  - *import jade.core.agent ;*
  - *public class monAgent extends Agent . . . ;*
- Chaque agent est identifie par un **AID**
  - Methode *getAID()* //pour récupérer l'AID
- Dans la méthode **setup()** (Obligatoire)
  - Enregistrer les langages de contenu
  - Enregistrer les Ontologies
  - Enregistrer les Services auprès du DF
  - Démarrer les Comportements (behaviors)

setup() est la première méthode qui sera appelée après instanciation de l'agent

```
protected void setup() {. . . . . ;}
```

Identification des agents

- Le nom d'un agent :
  - enregistré dans la plateforme auprès de l'AMS
  - de la forme :
    $< nom - local - agent > @ < nom - plate - forme >$
  - Doit être globalement unique
  - objet de la classe **jade.core.AID**
- Plate-forme par défaut :
  - $< main - host >:< main - port > /JADE$
  - $< main - host >$: nom de domaine ou adresse IP

Exemple de nom complet

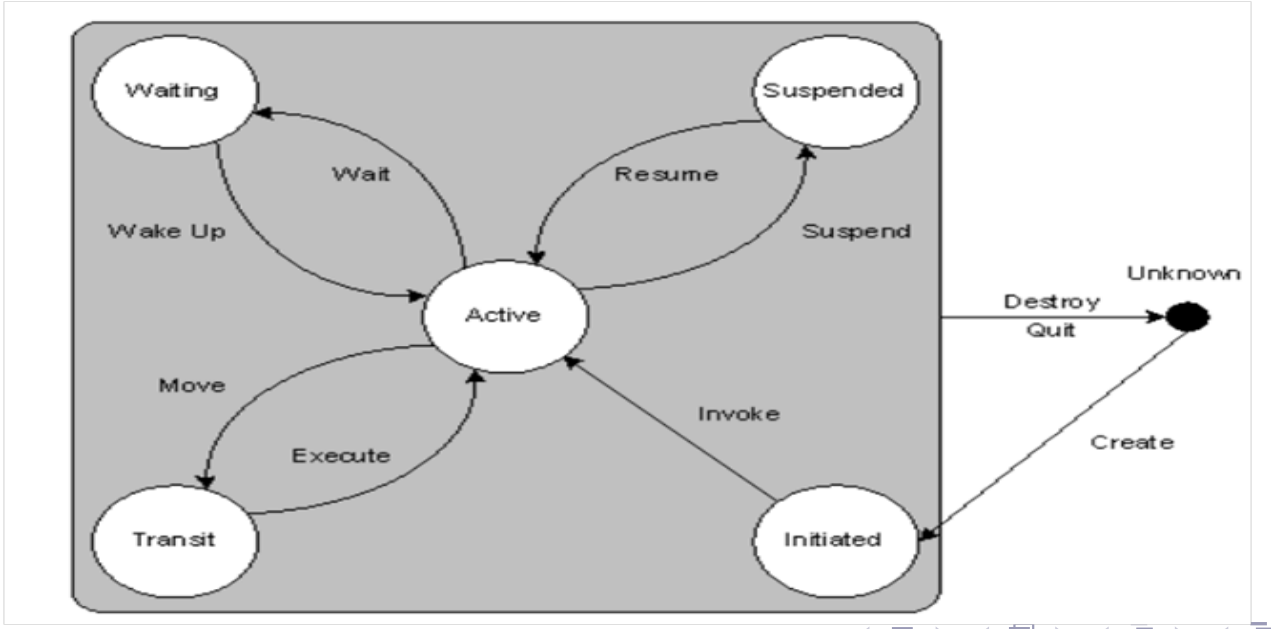
$acheteur1@192.168.30.12 : 1099/JADE$

Méthodes de la classe Agent

- Méthode **getArguments()** pour obtenir les arguments d'un agent
- Méthode **doDelete()** pour tuer (détruire) un agent
- Methode **takeDown()** appelée Avant que l'agent soit détruit

Note

la redéfinition des méthodes, définit le cycle de vie de l'agent

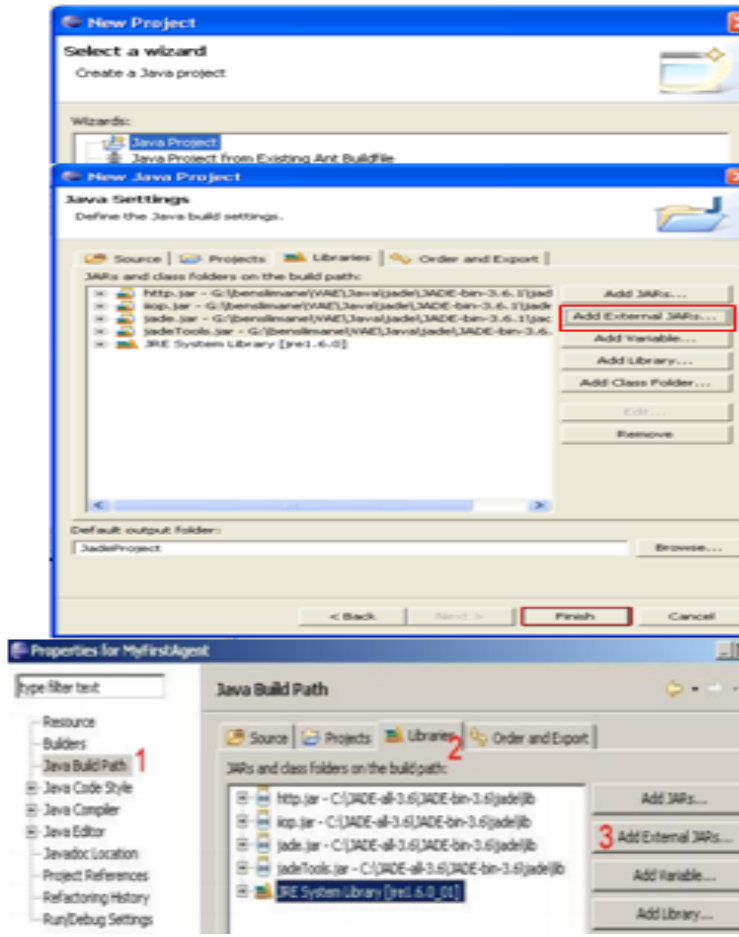


- Durant l'exécution d'un agent, son état peut être modifié
  - INITIATED : l'agent est lancé mais non enregistré auprès de l'AMS, aucun nom, aucune adresse
  - ACTIVE : l'agent est répertorié auprès de l'AMS et peut accéder aux services.
  - SUSPENDED : tous les behaviours de l'agent sont suspendus.
  - TRANSIT : l'agent migre vers une autre plateforme.
  - WAITING : tous les behaviours de l'agent sont temporairement interrompus.
  - DELETED : l'exécution de l'agent est terminée et n'est plus répertorié au sein de l'AMS
- JADE permet de contrôler le passage d'un agent d'un état à l'autre avec les méthodes doXXX

setup()	*INITIATED*
doSuspend() ↔ doActivate()	*SUSPENDED ↔ACTIVE*
doMove()	*ACTIVE ↔ TRANSIT*
doWait() ↔ doWake()	*WAITING ↔ ACTIVE*

Plan

- Ouvrir eclipse et créer un nouveau projet (MyFirstAgent par exemple)
- dans l'onglet **Libraries**, cliquer sur : **add external JARs**
- Ajoutez les quatre jar : "http.jar" "iiop.jar" "jade.jar" "jadeTools.jar" ;
- on peut aussi ajouter ces fichiers comme suit :
  1. Effectuer un clic droit sur le nom du projet. Puis choisisser **propriétés**.
  2. Cliquer sur **java build path** >> **Libraries** >> **add external JARs**



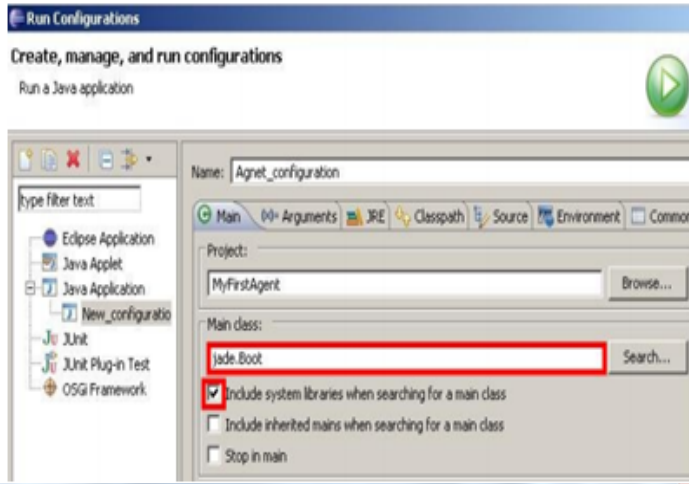
- dans le projet créé :
  - ajouter un package (firstAgent)
  - dans ce package, créer une nouvelle classe appelée HelloWorldAgent.

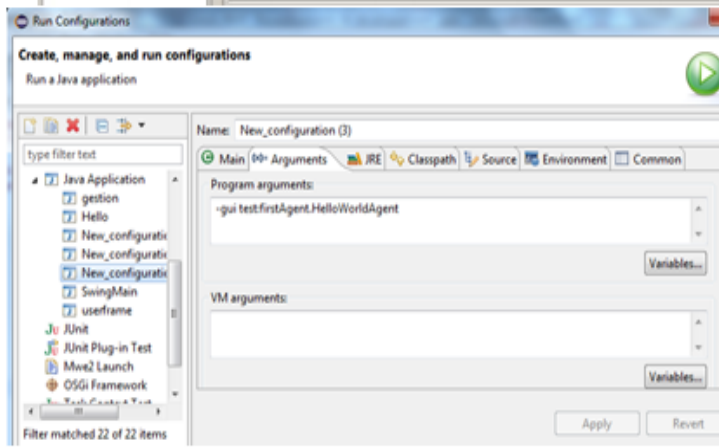
```
Package firstAgent;
Import jade.core.Agent;
public class HelloWorldAgent extends Agent {
protected void setup() {
   System.out.println("Hello! My name is" + getLocalName());
}
protected void takeDown() {
  System.out.println(« avant de mourir.....");
  // l'agent va se termine
   doDelete();
}
}
```

JADE sous Eclipse

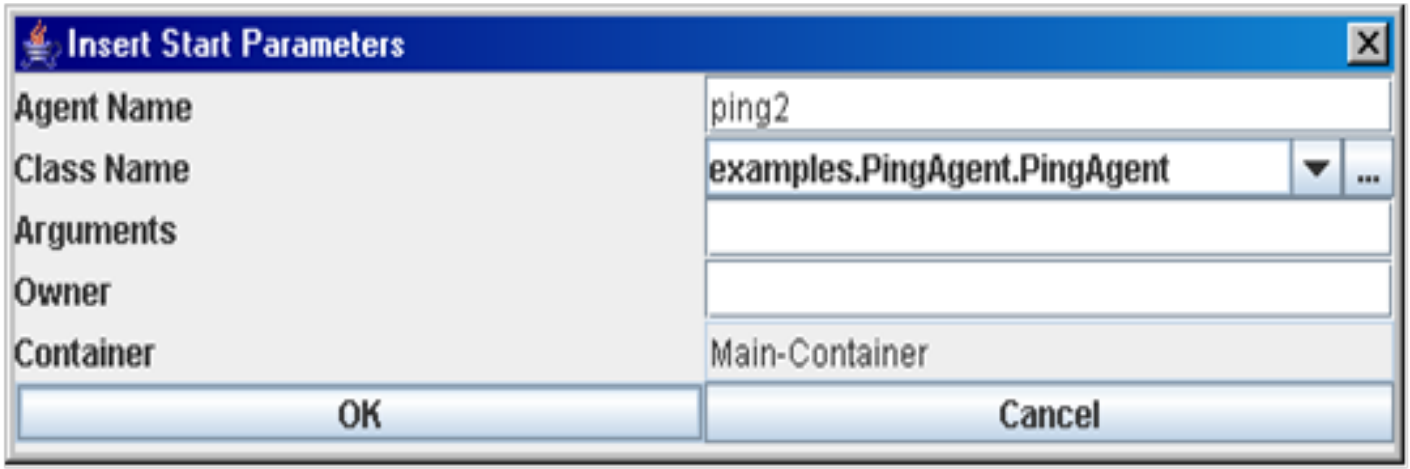
Mise en place de l'application sous Eclipse(Compilation et exécution de l'agent)

- Allez dans **run** >> **Run configuration** puis double-clique sur **java application** pour créer une nouvelle configuration .
- Dans l'onglet **main** et dans la zone de saisie **Main class**
  - taper le code suivant : jade.Boot
  - puis cocher la case : " Include librairies when searching for a main class "
- dans l'onglet **arguments** : tapez le code suivant : -gui Nomagent :NomPackage.NomClasse
- Dans notre exemple on tape le code suivant : -gui test :firstAgent.HelloWorldAgent
- cliquer sur **apply** puis sur **run** pour





- On peut aussi créer et lancer un agent en utilisant la console graphique (GUI) de gestion.
  - Faites un clic droit sur le conteneur principal.
  - Dans le menu contextuel sélectionnez l'option « Démarrer un nouveau Agent(Start New Agent) ».
  - Tapez le nom de l'agent et le nom de classe entièrement qualifié



- Dans Java, une interface graphique fonctionne sur son propre thread
  - gérer et réagir rapidement à des événements générés lorsque un utilisateur interagit avec l'interface graphique via un composant
- d'autre part, un programme d'agent fonctionne sur son propre fil d'exécution qui lui permet de gérer ses comportements.
- JADE a fourni un mécanisme approprié pour gérer les interactions entre les deux threads (interface graphique avec un agent).
- Le mécanisme est simplement basé sur le passage de l'événement.

- **interaction agent et l'interface graphique (GUI)** :
  - GUI implémente l'interface **ActionListener**
  - puis enregistre tous les composants interactifs de votre interface graphique via la méthode **addActionListener()**
  - Lorsque votre programme d'agent interagit avec l'interface graphique
    - une **ActionEvent** est générée par le composant de source, qui appelle la méthode actionPerformed()
    - l'implémentation de cette méthode permet à l'interface graphique de traiter l'événement

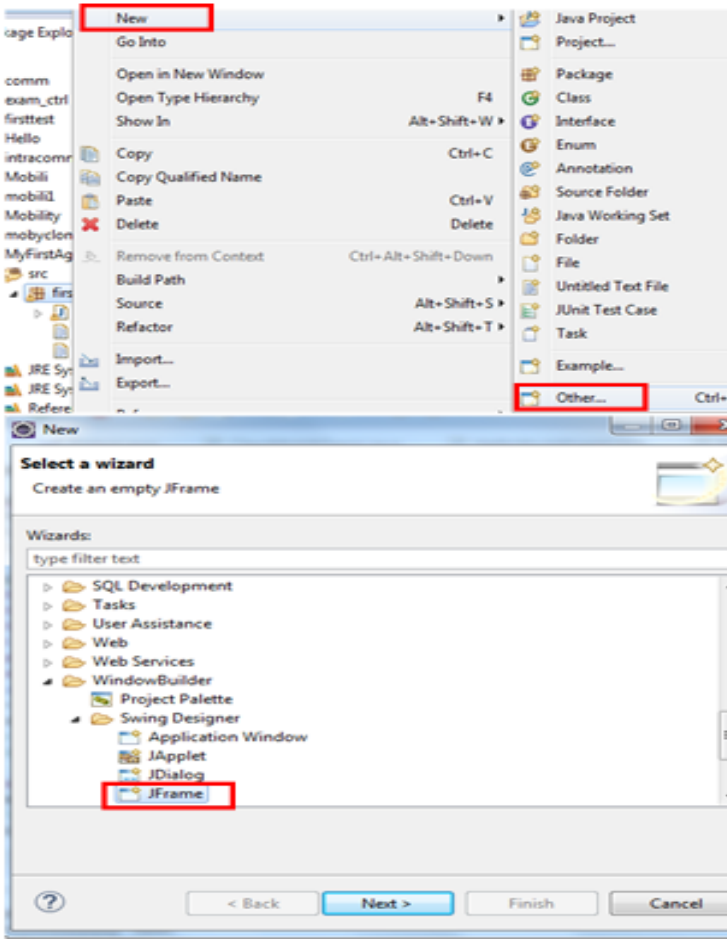
Puisque cette méthode appartient à la classe de GuiAgent, vous devez fournir à votre

- **interaction interface graphique (GUI) et agent** :
  - JADE a fourni classe abstraite **"GuiAgent"** qui étend la classe "Agent"
  - pour gérer les interactions entre une interface utilisateur et un programme de l'agent
    - le programme d'agent doit étendre la classe GuiAgent
    - pour utiliser les deux méthodes **postGuiEvent()** et **onGuiEvent()**
  - la méthode postGuiEvent() utilisée par l'interface graphique pour envoyer et poster les événements capturés vers l'agent
  - la méthode onGuiEvent() utilisée par l'agent pour recevoir et traiter les événements qui sont affichés par la GUI.
  - Pour poster un événement à l'agent, l'interface graphique crée simplement un objet GuiEvent, ajoute les paramètres requis et passe en argument à la méthode postGuiEvent ().
  - la GUI doit avoir une référence à la classe d'agent sur lequel le GUI peut invoquer cette méthode.

- comme illustrer dans l'exemple précédent, créer une classe d'agent "HelloWorldAgent"

```
public class HelloWorldAgent extends GuiAgent{
        public static final int OKEY = 0;
        public static final int ANNULER = 1;
        public static final int FERMER = 2;
        private int command;
        transient protected HelloWorldAgentGui myGui; // Reference to the gui
        protected void setup() {
                System.out.println("Hello World! My name is "+getLocalName());
                myGui = new HelloWorldAgentGui(this);
                myGui.setVisible(true);
                // Make this agent terminate
                //doDelete();
          }
        protected void onGuiEvent(GuiEvent ev) {
                // Process the event according to it's type
                command = ev.getType();
                if (command == OKEY) {
                System.out.println("Okeyyyyyyyyyy "+getLocalName());
                }
                else if (command == 1) {
                System.out.println("Annulerrrrrr "+getLocalName());
                }
                else if(command == 2) {
                        System.out.println("Fermerrrrrr "+getLocalName());
                }
                }
}
```

- puis créer la classe de l'interface graphique "HelloWorldAgentGui"
  - dans le même package cliquer droit, sélectionner **new** >> **other**
  - Aller dans **windowbuilder** >> **swing desginer** >> **JFrame** pour créer une nouvelle classe graphique.



- implémenter la classe d'interface graphique d'agent "HelloWorldAgentGui"

```
public class HelloWorldAgentGui extends JFrame implements ActionListener{

	private JPanel contentPane;
	private HelloWorldAgent myAgent; // Reference to the agent class
	private JButton btnNewButton;
	private JButton btnNewButton_1;
	private JButton btnNewButton_2;
	public HelloWorldAgentGui(HelloWorldAgent a) {
	super("HelloWorld");
	this.myAgent = a; // provide the value of the reference of Agent class
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 450, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		btnNewButton = new JButton("ok");
		btnNewButton.addActionListener(this);
		btnNewButton.setBounds(0, 18, 89, 23);
		contentPane.add(btnNewButton);

		btnNewButton_1 = new JButton("cancel");
		btnNewButton_1.setBounds(99, 18, 89, 23);
		contentPane.add(btnNewButton_1);
		btnNewButton_1.addActionListener(this);

		btnNewButton_2 = new JButton("quit");
		btnNewButton_2.setBounds(200, 18, 89, 23);
		contentPane.add(btnNewButton_2);
		btnNewButton_2.addActionListener(this);

	}
```

JADE sous Eclipse

conception d'un agent avec une interface graphique(Exemple)


```
public void actionPerformed(ActionEvent ae) {
        // TODO Auto-generated method stub
         //Process the event according to it's source
        if (ae.getSource() == btnNewButton ) {
          System.out.print("okkkkkkkkkkk sent");
          GuiEvent ge = new GuiEvent(this, myAgent.OKEY);
          ge.addParameter("");
          ge.addParameter("");
          myAgent.postGuiEvent(ge);
      }
        else if (ae.getSource() == btnNewButton_1) {
        System.out.print("CANCELLLLLLL sent");
  GuiEvent ge = new GuiEvent(this, myAgent.ANNULER);
  ge.addParameter("");
  ge.addParameter("");
  myAgent.postGuiEvent(ge);
        }
        else if (ae.getSource() == btnNewButton_2){
                System.out.print("Quitttttttt sent");
                GuiEvent ge = new GuiEvent(this, myAgent.FERMER);
                myAgent.postGuiEvent(ge);
        }}}
```

JADE sous Eclipse

conception d'un agent avec une interface graphique(résultat)