



JADE : Java Agent DEvelopment framework

•Laboratoire IBISC & Départ. GEII
Université & IUT d'Evry
nadia.abchiche@ibisc.univ-evry.fr



A large, thick, grey brushstroke graphic that starts as a circle on the left and curves downwards and to the right, framing the text.

Introduction à la plateforme JADE

- 1) Modèle d'agent
- 2) Services
- 3) Norme FIPA
- 4) Environnement et Architecture

JADE

généralités

- Plate-forme Multi-agents en Java developpee par Gruppo Telecom Italia
- Respecte la Norme FIPA
- But : Réaliser simplement des Systèmes Multi-Agents interoperables
- Principalement de agents de type “cognitifs”

JADE

conception et implantation

- Un agent :
 - possède un état
 - a un cycle de vie
- Classe Comportement dérivée de la super-classe Agent
- File de comportements d'agent : ajout ou suppression d'un comportement à tout moment

JADE

conception et implantation

- Programmation concurrente :
 - ✓ Un thread par agent
 - ✓ Plutôt qu'un thread par comportement pour éviter une surcharge en nombre de threads
- Planificateur de tâches
- Un comportement peut se bloquer lui-même pour éviter de gaspiller du CPU
 - ✓ Ex.: pendant qu'il attend des messages

JADE

conception et implantation

- Les services

- ✓ Action enregistrée et dispensée par la plateforme
- ✓ Comportements d'un ou plusieurs agents répondant à une demande.
- ✓ Annuaire des services = Pages Jaunes : Directory Facilitator (DF)

JADE

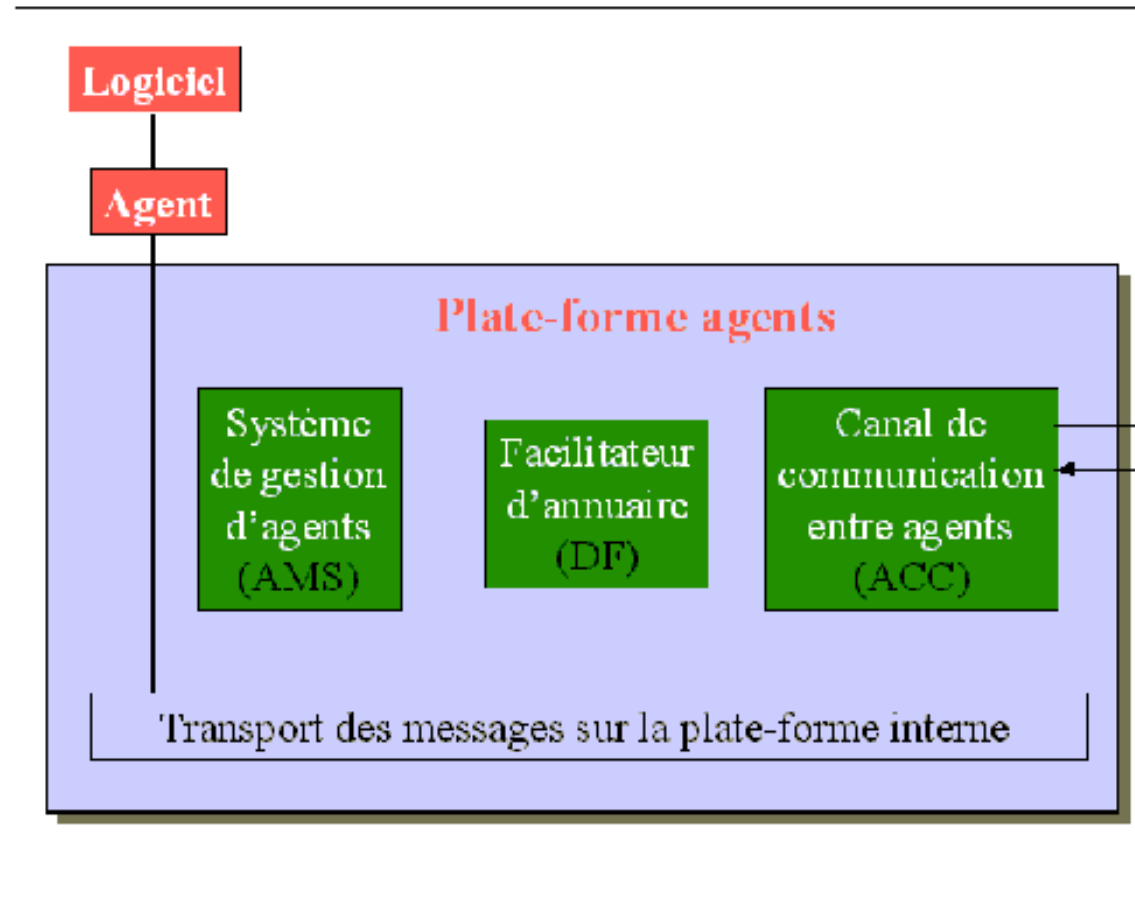
Conception et implantation

- Les Pages Jaunes (DF)
 - ✓ Enregistre les descriptions des agents ainsi que les services qu'ils offrent
 - ✓ Les agents peuvent :
 - ✓ Enregistrer leurs services auprès d'un DF
 - ✓ Demander au DF de découvrir les services offerts par d'autres agents

JADE

Conception et implantation

Norme FIPA (1997)



JADE

Conception et implantation

Spécificités de la plate-forme

- **Systeme de Gestion d'Agents (AMS)**
Agent central qui supervise les autres agents et les accès à la plate-forme (Pages Blanches)
- **Canal de Communication entre Agents (ACC)**
Agent qui fournit la route pour les interactions entre agents dans et en dehors de la plateforme
- **Facilitateur d'Annuaire (DF)**
Agent qui fournit un service de Pages Jaunes (recensement des services disponibles)

JADE

Conception et implantation

Environnement JADE

- AMS, ACC, DF
- Plate-forme Multi-agents distribuee :
 - 1 Machine Virtuelle JAVA par Hote
 - 1 agent = 1 thread
 - JADE planifie les tâches d'un agent plus efficacement que la machine virtuelle JAVA
- Messages ACL codés en objets JAVA
- Interface graphique

JADE

Conception et implantation

Architecture de la plateforme

- Un receptacle d'agents :
 - Est un objet serveur RMI qui gère localement un ensemble d'agents + Machine Virtuelle JAVA
 - Gère le cycle de vie des agents (création, mort, suspension, reprise,...)
 - Répartit les messages ACL dans les files d'attente des agents
- Interface graphique implémentée comme un agent

JADE

Conception et implantation

Architecture de la plateforme (suite)

- Communication de plusieurs machines virtuelles (VM) JAVA par la méthode RMI
- Chaque VM est un receptacle d'agents et un environnement multi-threads compose :
 - D'un thread d'exécution pour chaque agent
 - De threads de RMI pour échanger les messages
- Plusieurs VM sur le même hôte possible, mais déconseillé

JADE

Conception et implantation

Programmer avec JADE

- 1) Démarrer JADE
- 2) Programmer des agents
- 3) Service de Pages Jaunes
- 4) Interface
- 5) Intégration

JADE

Conception et implantation

Installation de JADE

- Installer Java (jdk 1.2 ou plus)
 - ✓ Attention au « path » et au « classpath »
- Installer le package Jade
 - ✓ Copier les classes
 - ✓ Mettre à jour le « classpath » :
 - \jade
 - \jade\Lib
 - \jade\Lib\iiop.jar
 - \jade\Lib\jade.jar
 - \jade\Lib\jadeTools.jar
- Installer les agents comme des classes java classiques

JADE

Conception et implantation

Démarrage de JADE

- Lancer Jade avec la ligne de commandes :
`java jade.Boot`
- Lancer Jade et la GUI
`java jade.Boot -gui`
- Lancer un agent au démarrage
`java jade.Boot -gui <nom de l'agent>:<classe de l'agent>`
- Lancer un agent avec des paramètres
`java jade.Boot -gui <nom de l'agent>:<classe agent> (<Param>)`

JADE

Conception et implantation

Création d'un agent

- Etendre la classe `jade.core.Agent`

```
import jade.core.agent;  
public class monAgent extends Agent ...;
```

- Chaque agent est identifié par un AID

- Méthode `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)

JADE

Conception et implantation

Identification des agents

- Le nom d'un agent :

- ✓ Est de la forme :

- ✓ `<nom-agent>@<nom-plate-forme>`

- ✓ Doit être globalement unique

- Plate-forme par défaut :

- `<main-host>:<main-port>/JADE`

- Nom de la plate-forme défini avec `-name`

JADE

Conception et implantation

Méthodes de la classe Agent

- Méthode getArguments() pour obtenir les arguments d'un agent
- Méthode doDelete() pour tuer un agent
- Méthode takeDown() pour lancer l'“agent garbage collector”

JADE

Conception et implantation

Exemple

```
public class HelloWorldAgent extends Agent {  
  
    protected void setup() {  
        // creation de l'agent il se décrit lui même  
        ServiceDescription sd = new ServiceDescription();  
        sd.setType( "Hello example" );  
        sd.setName( "HelloServiceDescription" );  
        DFAgentDescription dfd = new DFAgentDescription();  
        dfd.setName( getAID() );  
        dfd.addServices( sd );  
        // enregistrer sa description auprès du DF  
        DFService.register( this, dfd);  
        System.out.println("Hello! My name is"+ getLocalName());  
        //afficher son premier argument  
        Object[] args = getArguments();  
        if (args != null && args.length > 0) {  
            System.out.println("mon premier argument : "+args[0]);  
        }  
        // l'agent se termine  
        doDelete(); } }  
}
```

JADE

Conception et implantation

Création d'un comportement

- Créer (étendre la classe « behaviour »)

```
public class myBehaviour extends Behaviour
```

- Créer le constructeur avec la super classe

```
public myBehaviour(Agent agent) {  
super(agent); }
```

- Créer la méthode « action » (Obligatoire) qui correspond à l'exécution du behaviour

```
public void action() {<code du behaviour>}
```

JADE

Conception et implantation

Comportements (1)

- Pour faire faire une tâche à un agent :

- Créer une instance de la sous-classe Behaviour
- Appeler la méthode `MonAgent.addBehaviour()`

- Chaque sous-classe Behaviour doit implémenter les méthodes :

- `Public void action()` : Ce que fait le behaviour
- `Public boolean done()` : Si le behaviour est fini (rend "true") ou non (rend "false")

JADE

Conception et implantation

Comportements (2)

- Plusieurs types de comportements :
 - One Shot
 - Complexe : combinaison de comportements simples
 - SequentialBehaviour
 - ParallelBehaviour
 - FSMBehaviour
 - Cyclique
- Nouveau comportement mis en place quand la méthode done() du comportement courant retourne vrai

JADE

Conception et implantation

Comportements (exemples)

```
public class MyOneShotBehaviour extends OneShotBehaviour {
public void action() {
    // exécuter les opérations souhaitées une fois } }
public class MyCyclicBehaviour extends CyclicBehaviour {
public void action() {
    // exécuter les opérations sans arrêt} }
public class ThreeStepBehaviour extends Behaviour {
private int step = 0;
    // exécuter X, puis Y, puis Z, puis arrêt
public void action() {
    switch (step) {
    case 0:        // perform operation X
        step++;   break;
    case 1:        // perform operation Y
        step++;   break;
    case 2:        // perform operation Z
        step++;   break;    } }
public boolean done() {return step == 3; } }
```

JADE

Conception et implantation

Comportements (exemples)

```
public class MyStepAgent extends Agent {  
    protected void setup() {  
        System.out.println("Agent "+getLocalName()+"  
        démarre.");  
        // lancer le comportement générique  
        addBehaviour(new ThreeStepBehaviour());  
        // lancer un autre comportement (cyclic)  
        addBehaviour(new CyclicBehaviour(this));  
    }  
}
```

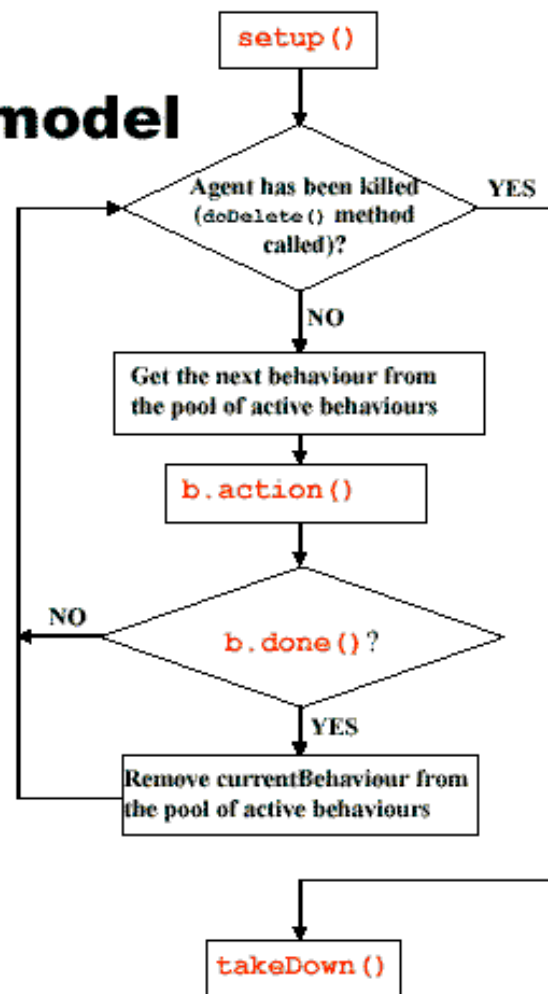

JADE

Conception et implantation

Résumé du fonctionnement d'un agent

The agent execution model

Highlighted in red the methods that programmers have to/can implement



- Initializations
- Addition of initial behaviours

- Agent "life" (execution of behaviours)


- Clean-up operations



JADE

Conception et implantation

Communication entre agents

- Messages codés en langage ACL
 - Transmission asynchrone
 - Messages instanciés par la classe `jade.lang.acl.ACLMessage`
- 

JADE

Conception et implantation

Exemples de messages

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM) ;  
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME)) ;  
msg.setLanguage("English") ;  
msg.setOntology("Weather-Forecast-Ontology") ;  
msg.setContent("Today it's raining") ;  
send(msg) ;
```

```
ACLMessage msg = receive() ;  
if (msg != null) {  
    // Process the message  
}
```

```
else{ block(); } //bloque le comportement
```

JADE

Conception et implantation

Exemples de messages (filtrer les messages)

```
MessageTemplate mt;  
mt= MessageTemplate.MatchPerformative(ACLMessage.CFP);  
ACLMessage msg = myAgent.receive(mt);  
if (msg != null) {  
    // un message de type CFP est reçu, récupérer son contenu  
    String titre = msg.getContent();  
    ... }  
}
```

```
MessageTemplate tpl = MessageTemplate.MatchOntology("Test-Ontology");  
  
public void action() {  
    ACLMessage msg = myAgent.receive(tpl);  
    if (msg != null) {  
        // Process the message  
    }  
    else {  
        block();  
    }  
}
```

JADE

Conception et implantation

Service de Pages Jaunes

- DF = Agent comme les autres qui communique avec des messages ACL
- Classe `jade.domain.DFService`
 - `Search()`
 - `Register()`
 - `Deregister()`
 - `Modify()`

JADE

Conception et implantation

Recherche dans un DF

- Pour s'inscrire dans un DF, il faut une description, classe `DFAgentDescription` :
 - AID de l'agent
 - Nom du service
 - Type de service
 - Langages, Ontologies, ...
- Pour rechercher dans un DF, créer une autre instance de la classe `DFAgentDescription`



JADE

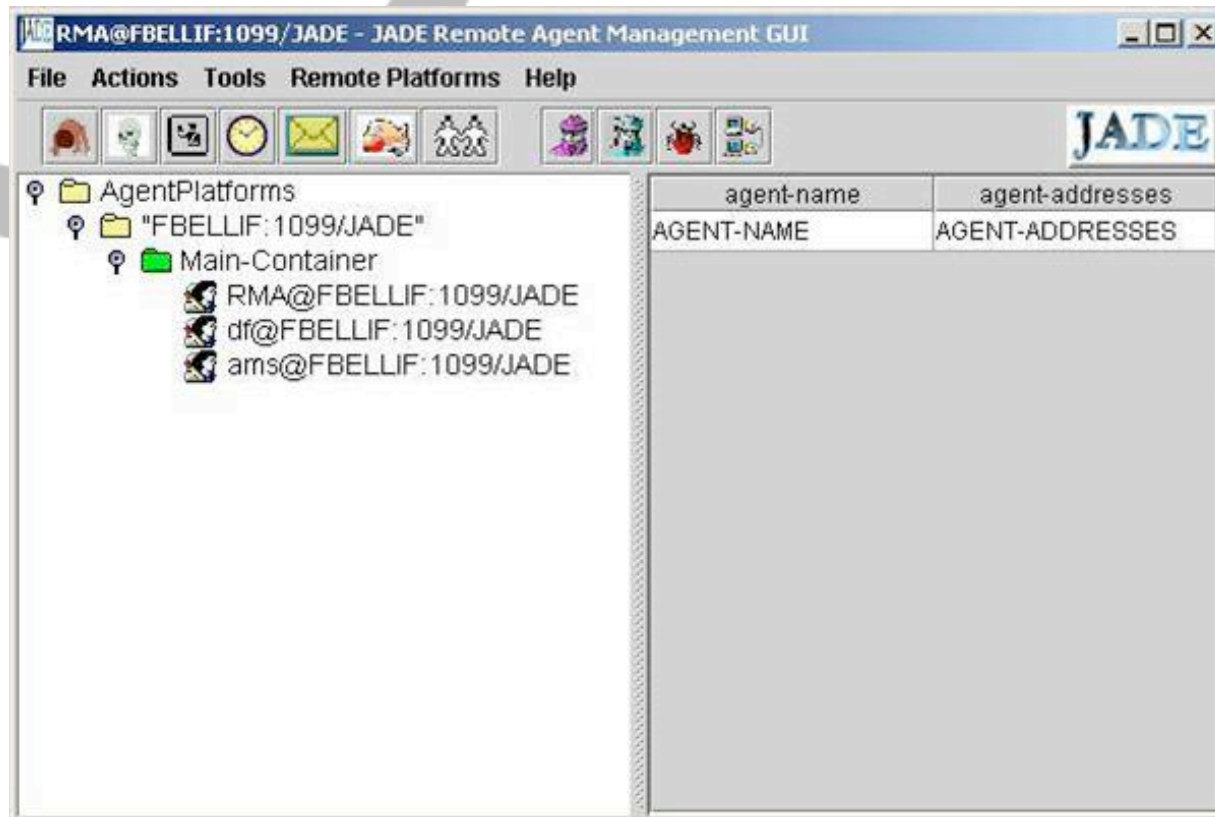
Conception et implantation

Interface

- Jade GUI
 - DF Agent GUI
 - Dummy Agent
 - Sniffer Agent
 - Introspector Agent
- 

Jade GUI

Permet de contrôler les agents

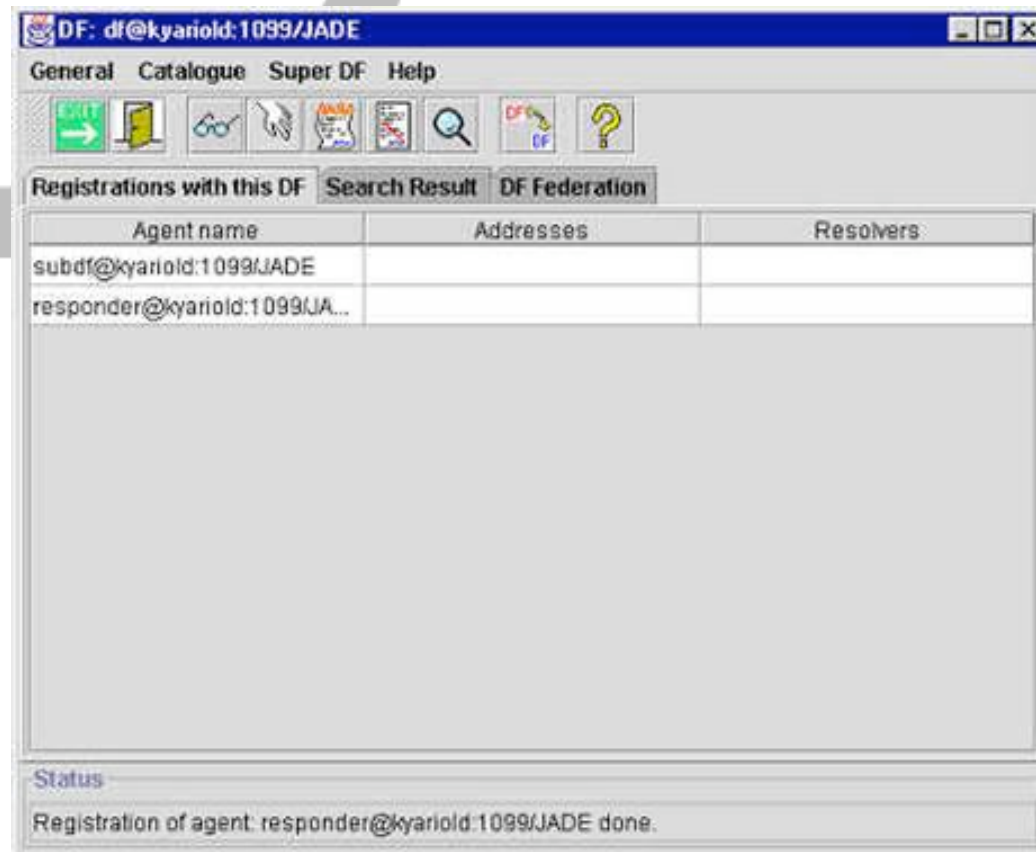


JADE

Conception et implantation

DF Agent GUI

Permet de consulter les Pages Jaunes

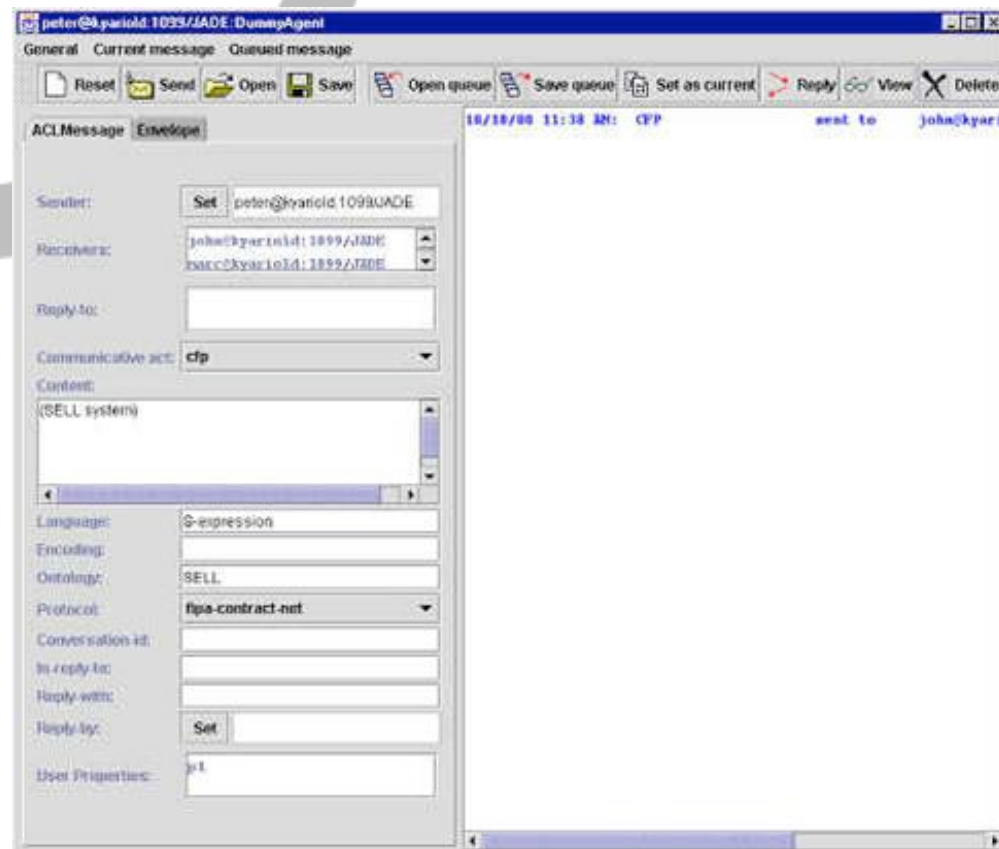


JADE

Conception et implantation

Dummy Agent

Pour envoyer et recevoir des messages

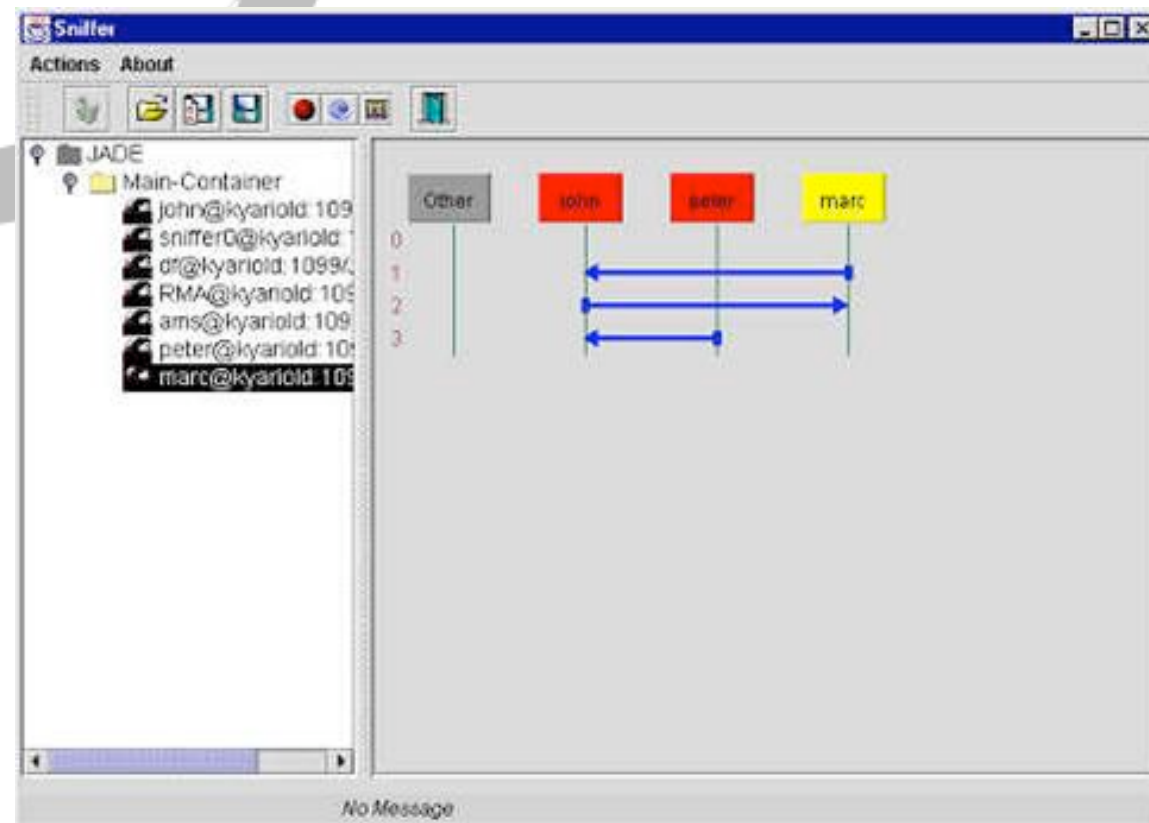


JADE

Conception et implantation

Sniffer Agent

Surveille les échanges de messages dans une plate-forme

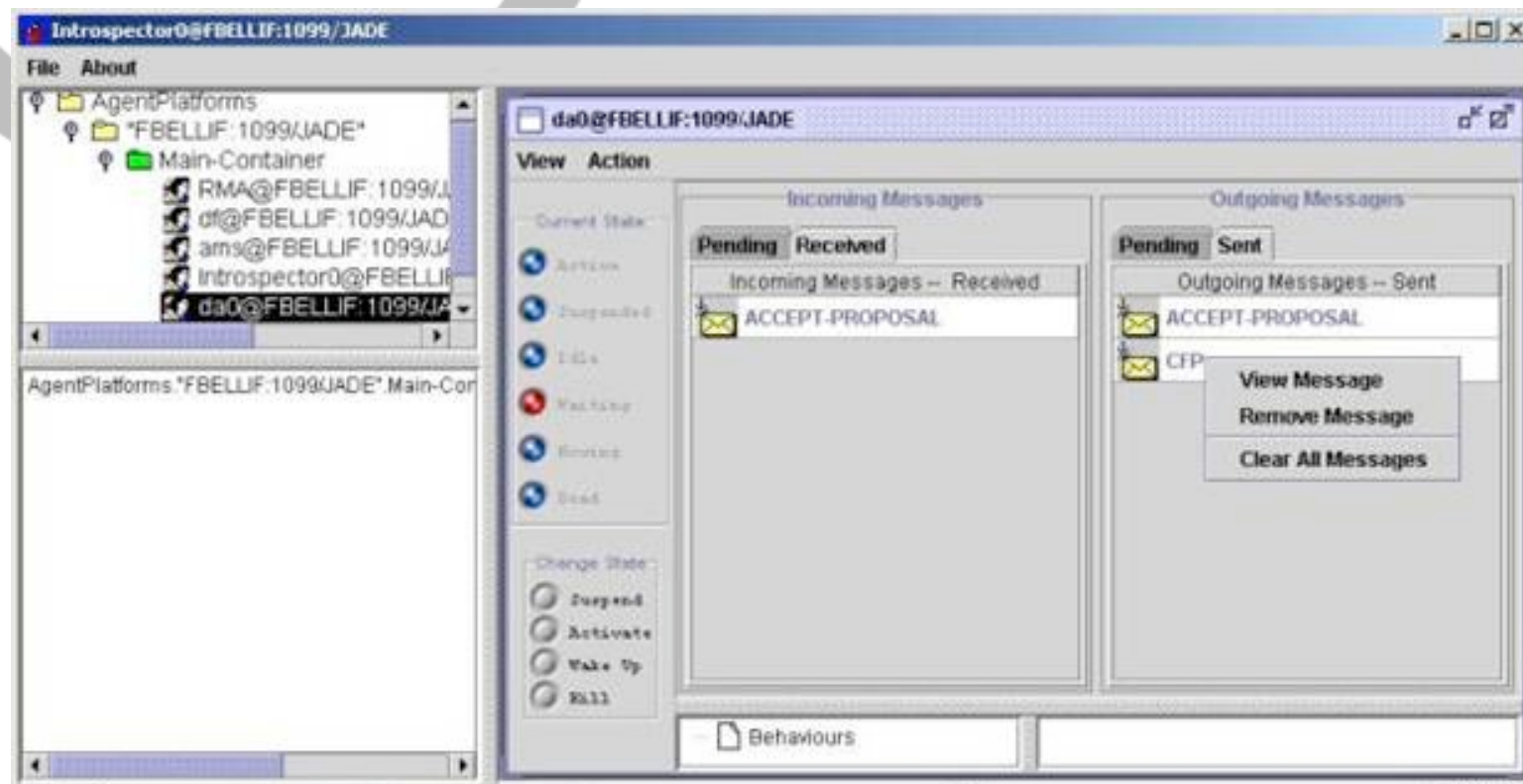


JADE

Conception et implantation

Introspector Agent

Pour surveiller l'activité d'un agent
(messages, cycle de vie)



JADE

Conception et implantation

- Integration avec JESS :
 - Communication avec les messages de JESS
 - Un programme JESS peut contrôler les messages et les comportements des agents de JADE
 - Classe JessBehavior

JADE

Conception et implantation

Conclusion

- Communauté de plus en plus importante
- Exécution distribuée sur plusieurs hôtes et types de machines (PC, mobile, ...)
- Exécution concurrente des agents
- Communication transparente par message (ACL)
- Open Source
- Norme FIPA

JADE

Conception et implantation

Approfondir en TP ?

- Encapsuler le CBR et le RBR « stimulation cognitive » avec Jess dans deux agents en vue de les faire coopérer ;
- Dialogue dans un domaine de connaissance (à choisir) entre 2 plusieurs agents. Faire Communiquer les agents avec : tentative de compréhension en employant la notion ontologie ;
- Négociation via Contract Net protocol (exemple d'application d'achat/vente) ;
- Résolution d'un problème par décomposition en tâches et sous tâches ayant des caractéristiques/ performances hétérogènes (coopération). Cycle communication-délibération-action.

JADE

Conception et implantation

Bibliographie

- FIPA: Foundation for Intelligent Physical Agents. Specifications. 1997.
<http://www.fipa.org>
- Plate-forme JADE: Java Agent Development Framework, 2000. <http://jade.tilab.com/>
- Java Expert System Shell (JESS)
<http://herzberg.ca.sandia.gov/jess/>
- AUML. The Agent Unified Modelling language, <http://www.auml.org/>