La Programmation Orientée Agent

FIPA - Jade

Emmanuel ADAM

Université Polytechnique des Hauts-De-France

UPHF/ISTV-LAMIH

- Protocoles de communication
  - Protocole Subscribe
  - Protocole AchieveRE
  - Protocole AchieveRE Simplifié
- Cas d'étude
- Cas d'étude - Solution sans protocole
- Cas d'étude - Solution avec protocole

Généralités sur FIPA (Foundation of Intelligent Physical Agents)

Architecture abstraite

- But : favoriser l'interopérabilité et la réutilisabilité
- Identifier les entités architecturales abstraites et leurs relations
  - Transport de messages, langage de communication, services de renseignements, langages de contenu
- Identifier les entités difficilement définissable de façon abstraite
  - Gestion et mobilité des agents

Transports de messages

- But : gérer la livraison et la représentation des messages, des protocoles de communications inter-réseaux et inter-plateformes
  - Message = enveloppe + corps
  - 1 MTS sur chaque plateforme

Généralités sur FIPA (Foundation of Intelligent Physical Agents)

Gestion des agents

- But : fournir la structure permettant l'existence et le fonctionnement des agents
- Modèle de référence logique pour la création, l'enregistrement, la localisation, la communication, la migration et le retrait d'agents
- Décrit les primitives et ontologies nécessaires pour les services
  - *Pages Blanches* : localisation, désignation et contrôle de l'accès aux services (AMS)
  - *Pages Jaunes* : localisation, et enregistrement des services (DF)
  - Service de transport de messages

Généralités sur FIPA (Foundation of Intelligent Physical Agents)

Communication entre agents

- Techniques de communication pour structurer les interactions dans le système
- Spécifications du langage de communication + protocoles d'interaction + bibliothèques d'actes communicatifs prédéfinis + protocoles d'interaction + langages de contenu

Plateforme FIPA

- Plateformes dites "FIPA-compliant" :
  - JADE (Java Agent DEvelopment)
  - JACK (Agent Oriented Software Group)

Généralités sur JADE

JADE : Java Agent DEvelopment framework

- But : développement et exécution de systèmes multi-agent conformes aux normes FIPA
  - service de nommage
  - service de pages jaunes
  - transport de messages
  - service d'analyse
  - bibliothèque des protocoles d'interaction de FIPA
- Les agents sont des coquilles auxquelles il faut ajouter des comportements implémentant des services/fonctionnalités
- Les communications utilisent le standard ACL
- Possibilité de communications entre plateformes JADE

Généralités sur JADE

Plateforme

- Comprend 3 composantes de base :
  - un environnement d'exécution dans lequel les agents "existent"
  - une librairie de classes java utilisable directement ou par extension pour développer les agents
  - une suite d'outils graphiques permettant l'administration et la gestion des agents actifs

Caractéristiques de JADE

- Entièrement implémentée en JAVA
- Conforme aux spécifications FIPA
- Open Source et distribué avec licence LGPL
- Distribution possible sur différents serveurs
- Modifiable en cours d'exécution (mobilité des agents)

Architecture de JADE

Architecture

- Chaque instance de JADE est un Conteneur (Container)
- Plateforme = ensemble de Conteneurs
  - obligation d'avoir un Conteneur Principal (Main Container) actif
  - d'autres conteneurs peuvent s'y enregistrer
- Le conteneur principal possède 2 agents spéciaux
  - AMS (Agent Management System) : Système de gestion d'agents
  - DF (Directory Facilitator) : Service de pages jaunes

Architecture de JADE

Conteneur

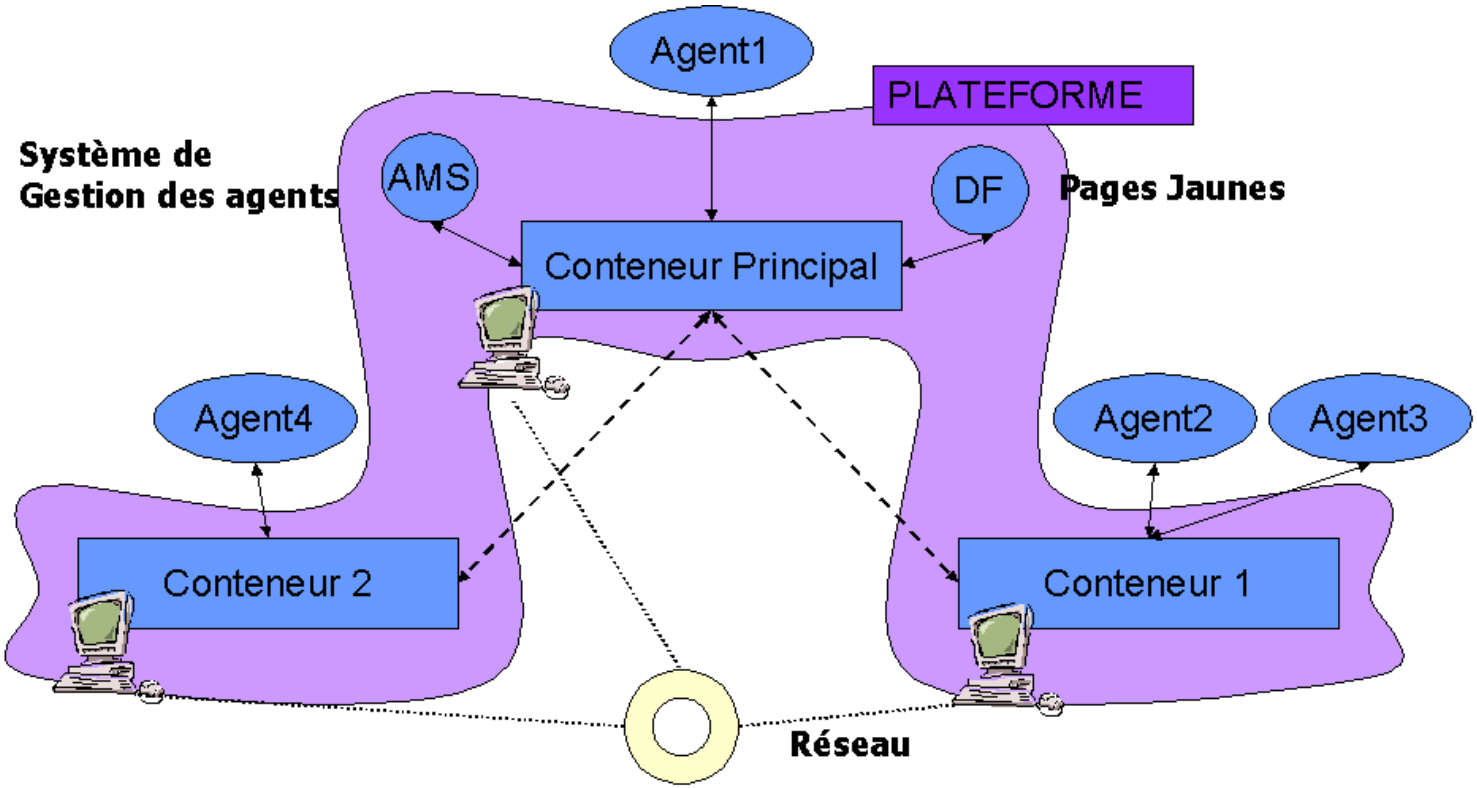
Chaque conteneur d'agents :

- environnement multi-threads composé d'un thread d'exécution pour chaque agent
- gère localement un ensemble d'agents
- règle le cycle de vie des agents (création, attente et destruction)
- assure le traitement des communications :
  - répartition des messages ACL reçus
  - routage des messages
  - dépôt des messages dans les boîtes privées de chaque agent
  - gestion des messages vers l'extérieur

Architecture de JADE

Exemple d'architecture

UNE plateforme avec 3 conteneurs et 4 agents

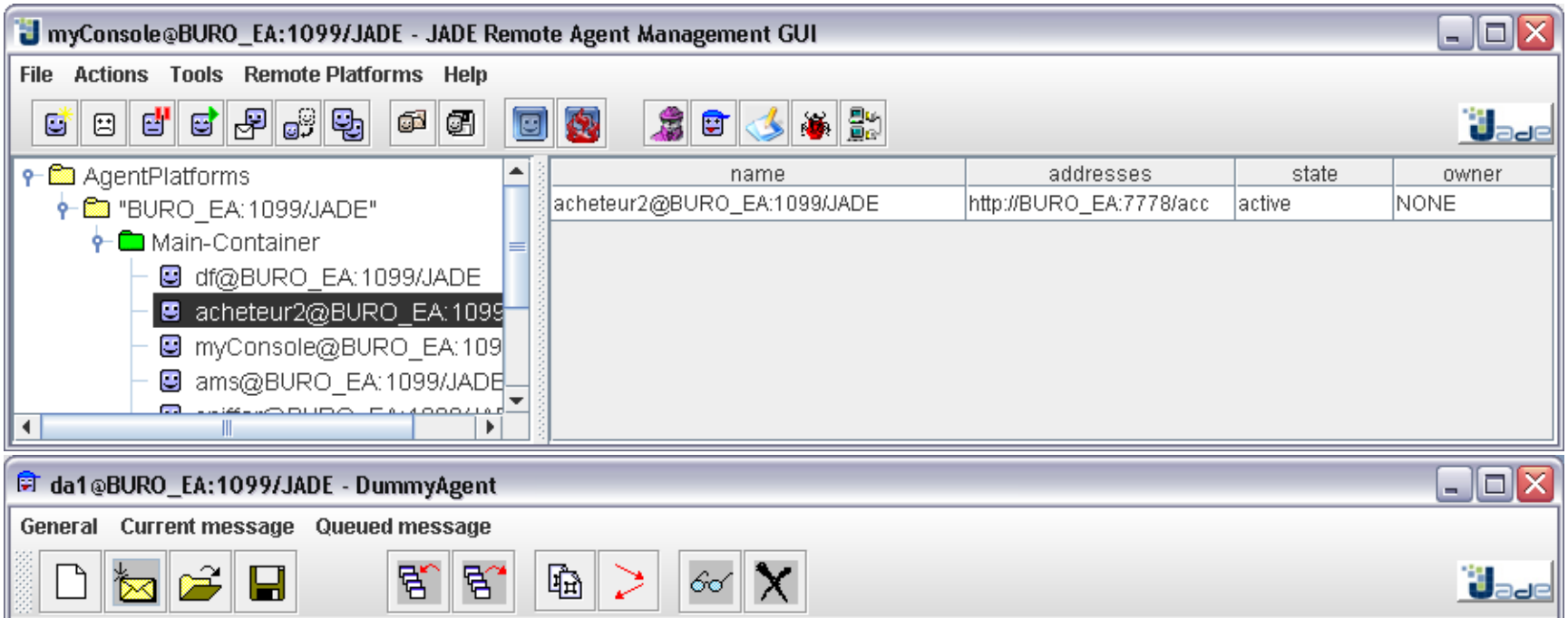


Outils de JADE

Outils principaux

JADE propose quelques outils de gestion dont :

- JADE RMA (Remote Agent Management) : gestion des agents d'une plateforme
- JADE Dummy Agent : permet d'envoyer des messages aux agents
- JADE Sniffer : analyse des messages échangés entre agents
- JADE Introspector : affiche le détail du cycle de vie d'un agent



Agents en JADE

Agent JADE

- coquille à laquelle il faut ajouter des comportements implémentant des services/fonctionnalités
- Suit son cycle de vie

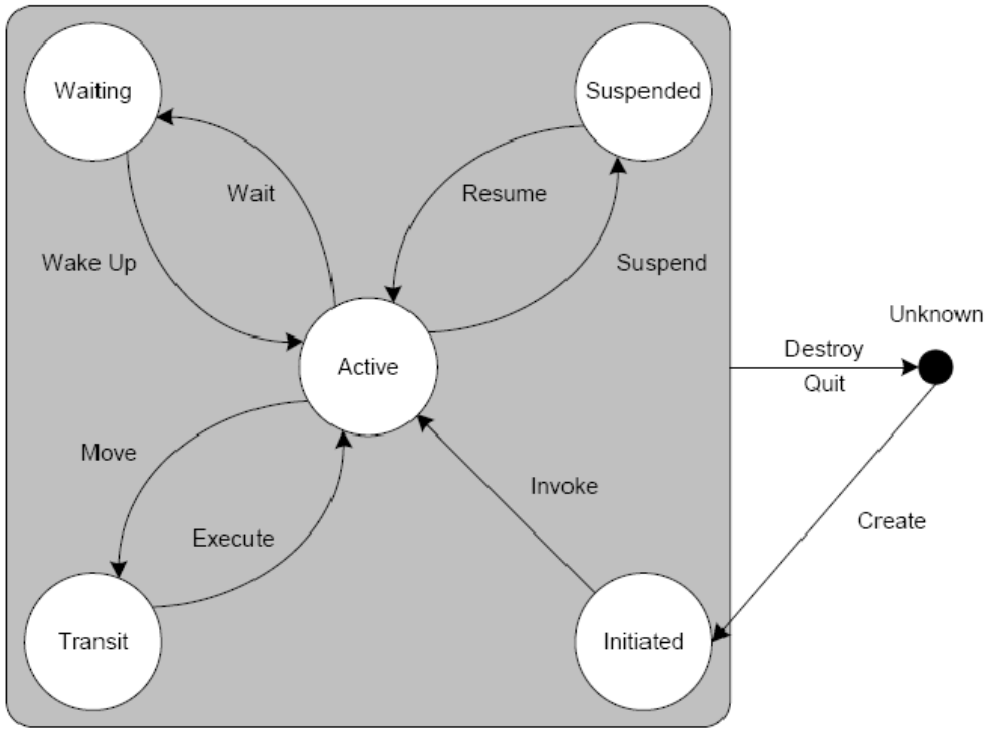


Figure 3 - Agent life-cycle as defined by FIPA.

Agents en JADE

Agent JADE - Elément de programmation

- hérite de la classe Agent
- Thread
- Possibilité de déclarer/rechercher un *service*
  - créer des groupes d'agents répondant à une demande
  - facilite la recherche d'agents répondant à un besoin
  - Gérés par Pages Jaunes
  - Proche de la notion de Web Services
- Méthode *setup*() invoquée dès la création de l'agent
- Méthode *takedown*() invoquée avant qu'un agent ne quitte la plateforme (soit détruit)

Exemple de l'agent HelloWorld I

```
import jade.core.Agent;
import static java.lang.System.out;

public class AgentHello extends Agent {
 private String texteHello;
 /** Initialisation de l'agent */
 protected void setup() {
  texteHello = "Bonjour a toutes et a tous";
  out.println("De l'agent "+ getLocalName() + " : " + texteHello);
  out.println("Mon adresse est "+ getAID());
  //tuer l'agent
  doDelete();
  }
 }
 /** Adieu */
 protected void takeDown() {
  out.println("Agent " + getLocalName() + " part.");
 }
}
```

Exécuter des agents HelloWorld

Lancer la plateforme et les agents

- Il faut premièrement demander le chargement de la plateforme
- puis demander le chargement des agents :
  `java jade.Boot -gui a1:AgentHello;a2:AgentHello`

Résultat de l'exécution

```
oct. 01, 2018 4:30:38 PM jade.core.Runtime beginContainer
...
Agent container Main-Container@10.4.151.26 is ready.
...
De l'agent a1 : Bonjour à toutes et à tous
De l'agent a2 : Bonjour à toutes et à tous
Mon adresse est ( agent-identifier :name a2@10.4.151.26:1099/JADE  :address
Agent a2 quitte la plateforme.
Mon adresse est ( agent-identifier :name a1@10.4.151.26:1099/JADE  :address
Agent a1 quitte la plateforme.
```

Comportements d'agents en JADE

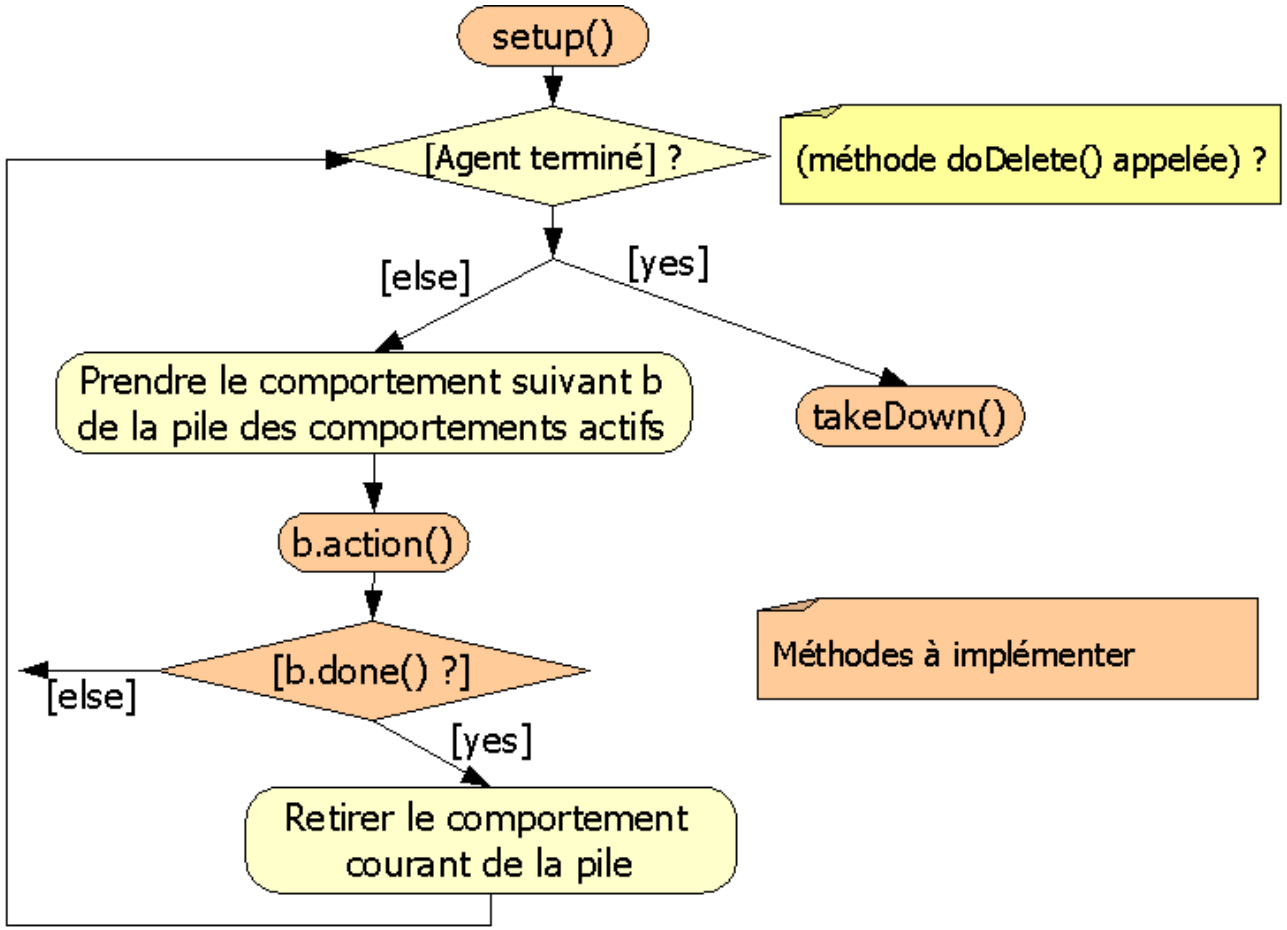
Comportement JADE - Elément de programmation

- hérite de la classe Behaviour ou d'une de ses sous-classes.
- possède deux méthodes.
  - Méthode *action*() définit les actions à exécuter par l'agent.
  - Méthode *done*() retourne un booléen spécifiant si le comportement doit être retiré de la file des comportements de l'agent.
- possibilité d'ajouter/retirer des comportements à un agent en cours d'exécution.
- un agent "dort" s'il n'a pas de comportement à exécuter.

Comportements d'agents en JADE

Cycle de vie d'un comportement

La gestion des comportements d'un agent utilise ce cycle de vie



Comportements d'agents en JADE

Eléments de programmation

- Un comportement peut être bloqué par `block()`
  - prend effet dès la fin de `action()`
  - jusqu'à
    - l'arrivée d'un message ACL,
    - la date limite de blocage si définie,
    - le lancement de la méthode `restart()`,
- `onStart()` est lancé à l'initialisation du comportement
- `onEnd()` est lancé à la fin du comportement et après son retrait de la file des comportements

Comportement simple

Comportement classique

Le comportement se termine lorsque la méthode done() retourne vrai

```
public class MonComportementATroisEtape extends Behaviour {
  private int step = 0;
  public void action() {
    switch (step) {
      case 0: tache1(); step++; break;
      case 1: tache2();step++; break;
      case 2: tache3();step++; break;
    }
  }
  public boolean done() { return (step == 3); }
}
```

Exemple de comportement simple

```
import jade.core.behaviours.Behaviour;

public class TroisEtapes extends Behaviour {
  private int step = 0;
  public void action() {
    switch (step) {
      case 0: affiche("etape 1"); step++; break;
      case 1: affiche("etape 2"); step++; break;
      case 2: affiche("etape 3"); step++; break;
    }
  }

  private void affiche(String texte)
  { System.out.println(myAgent.getLocalName()+": " + texte); }

  public boolean done() { boolean end=(step == 3);
    if(end)myAgent.doDelete(); return end;}
}
```

Exemple d'agent recevant un comportement

```
import jade.core.Agent;

public class AgentTroisEtapes extends Agent {

 /** Initialisation de l'agent */
 protected void setup() {

   System.out.println("De l'agent "+ getLocalName() + ": Hello !");
   TroisEtapes comp3 = new TroisEtapes();
   addBehaviour(comp3);
  }

 // 'Nettoyage' de l'agent
 protected void takeDown() {
  System.out.println("Agent " + getLocalName() + " quitte la
      plateforme. ");
        }
}
```

Exécuter des agents à 3 étapes

Lancer la plateforme et les agents

- Il faut premièrement demander le chargement de la plateforme puis demander le chargement des agents :
  `java jade.Boot -gui a1:AgentTroisEtapes;a2:AgentTroisEtapes`

Résultat de l'exécution

```
De l'agent a1 : Hello !
a1: etape 1
De l'agent a2 : Hello !
a2: etape 1
a1: etape 2
a2: etape 2
a2: etape 3
a1: etape 3
Agent a1 quitte la plateforme.
Agent a2 quitte la plateforme.
```

Comportement éphémère : One-shoot

Comportement éphémère

Le comportement "`One-shoot`" se termine immédiatement, `action()` est exécutée une seule fois.
La classe `jade.core.behaviours.OneShotBehaviour` implémente la méthode `done()` qui retourne `true`.

```
public class MonComportementUneFois extends OneShotBehaviour {
public void action() { /* effectue les taches */ }
}
```

Comportement perpétuel : Cyclic

Comportement cyclique

Le comportement "`Cyclic`" ne se termine jamais, `action()` est exécutée à chaque appel du comportement.
La classe `jade.core.behaviours.CyclicBehaviour` implémente la méthode `done()` qui retourne `false`.

```
public class MonComportementCyclique extends CyclicBehaviour {
public void action() { /* effectue les taches */ }
}
```

Les comportements différés

Comportements actionnables selon un temps

WakerBehavior : effectue une tâche après un délai

TickerBehavior : effectue une tâche périodiquement

```
protected void setup() {
 addBehaviour(new WakerBehaviour(this, 5000) {
  public void onWake() {
  System.out.println(myAgent.getLocalName() + "␣je␣meurs.");
  myAgent.doDelete();}
 });

 addBehaviour(new TickerBehaviour(this, 500) {
  public void onTick() {
  System.out.println(myAgent.getLocalName() + "␣tictac␣");}
 });
}
```

Les comportements pour communiquer

Comportements prédéfinis

MsgReceiver : traite la réception d'un message

SimpleAchieveREInitiator : lance un message et gère les réponses

SimpleAchieveREResponder : attend un message et répond

Les comportements complexes

Comportements composés

- comportements incluant de sous-comportements :

  SequentialBehavior : enchaînement de comportements
  ParallelBehavior : exécution de comportements en concurrence
  FSMBehavior : exécution de comportements selon une Machine d'Etats Finis (Finished State Machine)

Communication entre agents en JADE

Communication JADE - Élément de programmation

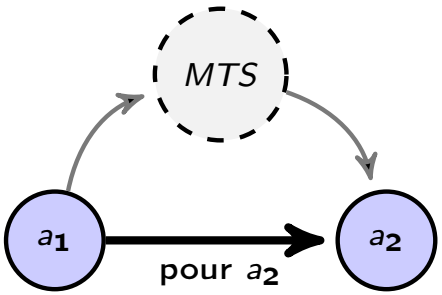
- Communication asynchrone par protocole ACL (Agent Communication Language)
- Chaque agent :
  - possède une "boîte aux lettre" (agent messages queue)
  - est averti dès qu'un nouveau message est arrivé dans sa boîte

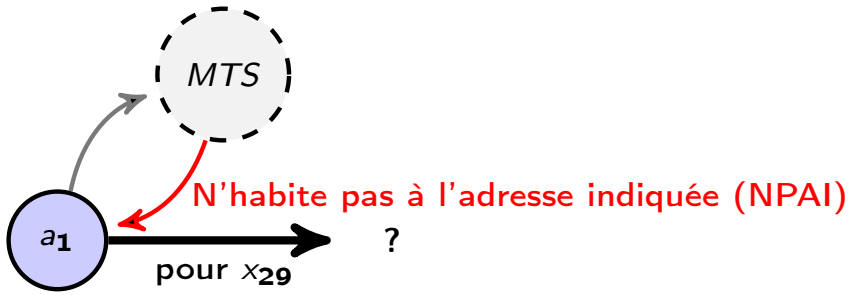
Structure d'un message FIPA-ACL

Emetteur	(sender)
Destinataires	(receivers)
Performatifs	(REQUEST, INFORM, QUERY_IF, CFP (Call For Proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, ...)
Contenu	(content)
Langage	(language) : syntaxe
Ontologie	(ontology) : vocabulaire

Gestion concurrence : conversation-id, reply-with, in-reply-to, reply-by...

Communication directe entre agents en JADE (1/3)





Communication directe

Envoi d'un message : Utilisation de la méthode send()

- le message est envoyé au service de communication (MTS) en contact avec les pages blanches (AMS)
- si le destinataire existe, le message est déposé dans sa boite aux lettres
- si le destinataire n'existe pas, l'emetteur reçoit un accusé de non réception (qu'il peut traiter ou non (ci-dessus, il ne le traite pas))

Communication entre agents en JADE (1/3)

Communication directe

Envoi d'un message : Utilisation de la méthode send()

```
//rqe code en java 10, utilisation de var et typage des variables a la compilation
// Message transmettant une demande d'offre
var helloMsg = new ACLMessage(ACLMessage.INFORM);
//ajout de l'adresse (AID) de l'agent de nom "voisin" declare sur la machine locale
helloMsg.addReceiver(new AID("voisin", AID.ISLOCALNAME));
//ajout du contenu du message
helloMsg.setContent("Salut voisin");
//envoi par la poste
send(helloMsg);
```

- le message est envoyé au service de communication en contact avec les pages blanches (AMS)
- si le destinataire existe, le message est déposé dans sa boite aux lettres
- si le destinataire n'existe pas, l'emetteur reçoit un accusé de non réception (qu'il peut traiter ou non (ci-dessus, il ne le traite pas))

Communication entre agents en JADE (2/3)

Communication directe

Lecture d'un message Utilisation de la méthode receive()

Méthode non bloquante, retourne le premier message de la boîte, ou null s'il n'en existe pas

```
// prendre un message de la file
ACLMessage msg = receive();
if (msg != null) {
// recuperation du contenu :
String contenu  = msg.getContent();
// recuperation de l'adresse de l'emetteur
AID adresseEmetteur = msg.getSender();
// recuperation du nom declare en local de l'emetteur
String nomEmetteur  = adresseEmetteur.getLocalName();

System.out.println("recu " + contenu +" , de "+nomEmetteur);}
```

Un comportement qui affiche chaque message

Boucle de retraits de messages

- `receive()` est non bloquante → à placer dans une boucle
- `block()` met en pause un comportement tant qu'un message n'est pas reçu par l'agent

```
protected void setup() {
// comportement cyclique d'affichage de messages
 addBehaviour(new CyclicBehaviour(this) {
  public void action() {
   ACLMessage msg = receive();
   if (msg != null) {
    System.out.println("recu " + msg.getContent()+" , de "+msg.
        getSender().getLocalName());}
   //pause, action() sera relancee si un message est recu
   block();
 });
}
```

Un Ping-Pong (le hello world de la communication) I

```
//rqe code en java 10, utilisation de var et typage des variables a la compilation
public class AgentDirectPingPong {
 String joueur;
 /** agent joueur de pingpong<br>
  * le joueur doit porter le nom ping ou pong */
 protected void setup() {
  joueur = getLocalName();
  println("Joueur " +  joueur + " est pret !");

  if(joueur.equals("ping")) {
    //lancement de la partie dans 15 secondes
    var debute = new WakerBehaviour(this, 15000){
      public void onWake(){
        var msg = new ACLMessage(ACLMessage.INFORM);
        msg.setContent("ping");
        msg.addReceiver(new AID("pong", AID.ISLOCALNAME));
        println("j'envoie le premier ping")
        send(msg); }};
    addBehaviour(debute);
  }
```

Un Ping-Pong (le hello world de la communication) II

```
//relance de la balle (50 fois)
addBehaviour(new Behaviour(this) {
   int i=0;
   public void action() {
     ACLMessage msg = receive();
     if (msg != null){
        println ("recu " + msg.getContent()) ;
        //retour a l'envoyeur, l'emetteur devient destinataire
        ACLMessage retour = msg.createReply();
        retour.setContent(Joueur + "-"+ i);
        send(retour);
        i++;
     }
     block () ; }
   public boolean done() {return i>50;}
       });

 void println(String msg) {
   System.out.println(this.getLocalName() + " -> " + msg);
 }
}
```

Exécuter les agents PingPong

Lancer la plateforme et les agents

- Créer deux agents du même type :

```
java jade.Boot -gui ping:agents.AgentDirectPingPong;
pong:agents.AgentDirectPingPong
```

Résultat de l'exécution

```
pong --> Joueur pong est pret !
ping --> Joueur ping est pret !
ping --> J'envoie le premier ping
pong --> recu ping , de ping
ping --> recu pong-0 , de pong
pong --> recu ping-0 , de ping
ping --> recu pong-1 , de pong
pong --> recu ping-1 , de ping
...
pong --> recu ping-49 , de ping
ping --> recu pong-50 , de pong
```

Modèle de messages

Tri du courrier

- `receive()` dépile un message de la pile → il n'est plus accessible. **S'il n'est pas celui attendu, il est perdu pour les autres comportements du même agent**
- possibilité de ne retirer que le message répondant à un modèle
- exemple de création de modèles :

```
//rqe code en java 10, utilisation de var et typage des variables a la compilation
// modele filtrant les messages identifie VENTE
var model1 = MessageTemplate.MatchConversationId("VENTE");
// modele filtrant les messages d'id VENTE et de type INFORM
var model2 = MessageTemplate.and(model1, MessageTemplate.
    MatchPerformative(ACLMessage.INFORM));
// ne retirer que les messages du type model2 :
receive(model2);
```

Modèles de messages

Possibilités de filtrage

- possibilité de filtrer sur l'émetteur, les destinataires, le performatif, l'identifiant, la date, le contenu ...
- possibilité de filtrer
  - les messages répondant à toutes les caractéristiques
  - les messages répondant à une caractéristique parmi plusieurs
  - les messages excluant une caractéristique

Un comportement dédié à la réception message

MsgReceiver

- Jade propose le comportement simple `MsgReceiver`
- sa fonction `handleMessage(ACLMessage msg)` est activée dès qu'un message répondant au modèle précisé est reçu, ou lorsque la date butoire est dépassée
- le constructeur est `MsgReceiver(Agent a, MessageTemplate mt, long deadline, DataStore s, java.lang.Object msgKey)`
  - le `DataStore` ainsi que sa clé peuvent être omis (remplacés par `null`)
  - le modèle peut être précisé ou remplacé également par `null`
  - la `deadline` est un temps en ms à compter de la date courante, elle peut être initialisée à `MsgReceiver.INFINITE` pour une attente indeterminée

Utilisation du MsgReceiver

Attente d'un message dans les 5 secondes

```
// comportement dans le setup de l'agent destinataire
// fonctionne en parallele
// se desactive dans les 5 secondes ou des reception de message
long finAttente = System.currentTimeMillis()+5000;
 addBehaviour(new MsgReceiver(this, null, finAttente, null, null) {

   protected void handleMessage(ACLMessage msg) {
     if (msg != null)
       System.out.println("j'ai recu : " + msg.getContent());
     else
       System.out.println("je n'ai rien recu ... " );
  }});
```

Utilisation du MsgReceiver

Attente d'un message INFORM sans date butoire

```
// comportement dans le setup de l'agent destinataire
// active des reception d'un message de type INFORM
// ne se desactive qu'en cas de reception de message
//rqe code en java 10, utilisation de var et typage des variables a la compilation
 var model = MessageTemplate.MatchPerformative(ACLMessage.INFORM);

 addBehaviour(new MsgReceiver(this, model, MsgReceiver.INFINITE,
     null, null) {

   protected void handleMessage(ACLMessage msg) {
   if (msg != null)
     System.out.println("j'ai recu : " + msg.getContent());
 }});
```

Utilisation du MsgReceiver

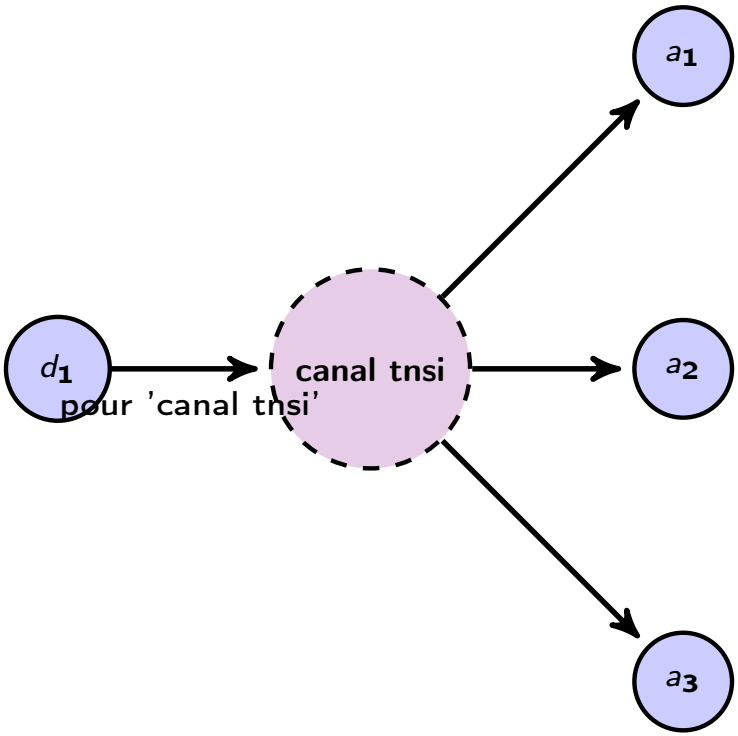
Relance du comportement pour recevoir DES messages

```
// comportement dans le setup de l'agent destinataire
// active des reception d'un message de type INFORM
// se relance en fin de traitement
//rqe code en java 10, utilisation de var et typage des variables a la compilation
 var model = MessageTemplate.MatchPerformative(ACLMessage.INFORM);

 addBehaviour(new MsgReceiver(this, model, MsgReceiver.INFINITE,
     null, null) {

   protected void handleMessage(ACLMessage msg) {
   if (msg != null)
     System.out.println("j'ai recu : " + msg.getContent());
   //relance du comportement pour un autre message
   reset(model, MsgReceiver.INFINITE, null, null);
 }});
```

Diffusion de messages en JADE



Diffusion sans nommer les destinataires

Diffusion : Utilisation d'un **topic** : sujet de diffusion

- le message est envoyé à l'adresse d'un canal
- les agents inscrits sur ce canal reçoivent le message

Diffusion de messages en JADE

Diffusion de messages sur un sujet

Emission sur un 'canal radio'

Definition du sujet : Utilisation du service "**topic**"

```
// Enregistrement du canal "TNSI NEWS" aupres du service topic
//rqe code en java 10, utilisation de var et typage des variables a la compilation
var topicHelper = (TopicManagementHelper) getHelper(
        TopicManagementHelper.SERVICE_NAME);
AID topicTNSI = topicHelper.createTopic("TNSI NEWS");
```

Diffusion de message à l'adresse du topic

```
var msg = new ACLMessage(ACLMessage.INFORM);
//les destinataires sont les auditeurs du canal "TNSI NEWS"
msg.addReceiver(topicTNSI);
msg.setContent(''nouvelles du Master'');
myAgent.send(msg);
```

Ecoute de messages diffusés

Ecoute de messages diffusés à propos d'un sujet

Reglage du canal : Création de topic (identique au cas de l'émetteur)

```
// Recuperation du canal "TNSI NEWS" aupres du service topic
var topicHelper = (TopicManagementHelper) getHelper(
        TopicManagementHelper.SERVICE_NAME);
AID topicTNSI = topicHelper.createTopic("TNSI NEWS");
//demander la reception dans la boite aux lettres des messages diffuses sur le canal
topicHelper.register(this.getAID(), topic);
```

Récupérer les messages d'un topic : utiliser un **modèle**

```
var msg= myAgent.receive(MessageTemplate.MatchTopic(topic));
//boucle obligatoire pour depiler tous les messages recu du topic
while (msg != null) {
  System.out.println(" msg recupere : " + msg.getContent());
  msg= myAgent.receive(MessageTemplate.MatchTopic(topic));
}
```

Ping-Pong à sens unique par canal radio I

```
//rqe code en java 10, utilisation de var et typage des variables a la compilation
public class DiffuseurAgent extends Agent {
  protected void setup() {
    try {
      // declarer le canal radio "PingPongSansPong"
      var topicHelper = (TopicManagementHelper) getHelper(
          TopicManagementHelper.SERVICE_NAME);
      AID topic = topicHelper.createTopic("PingPongSansPong");
      //envoyer un ping sur le canal toutes les 0.5 seconde
      addBehaviour(new TickerBehaviour(this, 500) {
        int i=0;
        public void onTick() {
          var msg = new ACLMessage(ACLMessage.INFORM);
          msg.addReceiver(topic);
          msg.setContent("ping"+i);
          i++;
          myAgent.send(msg);
      } } );
    } catch (Exception e) {e.printStackTrace();}
} }
```

Ping-Pong à sens unique par canal radio II

```
public class AuditeurAgent extends Agent {
 protected void setup() {
 //en argument un entier indiquant quand lancer le comportement de lecture
 int delai= Integer.parseInt(this.getArguments()[0].toString());
 try {
  // demander l'acces au canal "PingPongSansPong"
  var topicHelper = (TopicManagementHelper) getHelper(
      TopicManagementHelper.SERVICE_NAME);
  AID topic = topicHelper.createTopic("PingPongSansPong");
  topicHelper.register(this.getAID(), topic);

  //dans qq secondes, ajouter un comportement de lecture du canal
  addBehaviour(new WakerBehaviour(this, delai*1000) {
   protected void onWake() {
    //comportement cyclique de lecture des msg du canal
    myAgent.addBehaviour(new CyclicBehaviour(myAgent) {
    public void action() {
     ACLMessage msg = myAgent.receive(MessageTemplate.MatchTopic(
         topic));
     //ici on depile tous les messages recus depuis la creation du canal
     while (msg != null) {
```

Ping-Pong à sens unique par canal radio III

```
println("recu " + msg.getContent() + " du canal " + topic.
    getLocalName());
msg = myAgent.receive(MessageTemplate.MatchTopic(topic));
}
block();
} } ); } } );
}
catch (Exception e) { e.printStackTrace(); }
}

private void println(String msg) {
System.out.println("Agent "+ getLocalName()+" : "+ msg); }
//lorsque l'agent part, il se desenregistre aupres du canal
protected void takeDown() {
try {
var topicHelper = (TopicManagementHelper) getHelper(
    TopicManagementHelper.SERVICE_NAME);
AID topic = topicHelper.createTopic("PingPongSansPong");
topicHelper.deregister(this.getAID(), topic);
} catch (ServiceException e) { e.printStackTrace(); }
}
}
```

Ping-Pong à sens unique par canal radio IV

```
public class LaunchTopicAgents {
 public static void main(String... args) {
  // services et agents a lancer par JADE
  Properties pp = new ExtendedProperties();
  // autoriser la fenetre de controle
  pp.setProperty(Profile.GUI, "true");
  // activer le 'Topic Management Service'
  pp.setProperty(Profile.SERVICES, "jade.core.messaging.
      TopicManagementService;jade.core.event.NotificationService");
  //definir les agents a lancer
  var lesAgents = new StringBuilder();
  lesAgents.append("diffuseur:topic.DiffuseurAgent;");
  lesAgents.append("auditeur1:topic.AuditeurAgent(1);");
  lesAgents.append("auditeur2:topic.AuditeurAgent(2)");
  pp.setProperty(Profile.AGENTS, lesAgents.toString());
  // definir un profil base sur ces proprietes
  var pMain = new ProfileImpl(pp);
  // lancer le conteneur principal de JADE avec ce profil
  Runtime.instance().createMainContainer(pMain);
  }
}
```

Exécuter les agents PingPong sans Pong

Lancer la plateforme et les agents

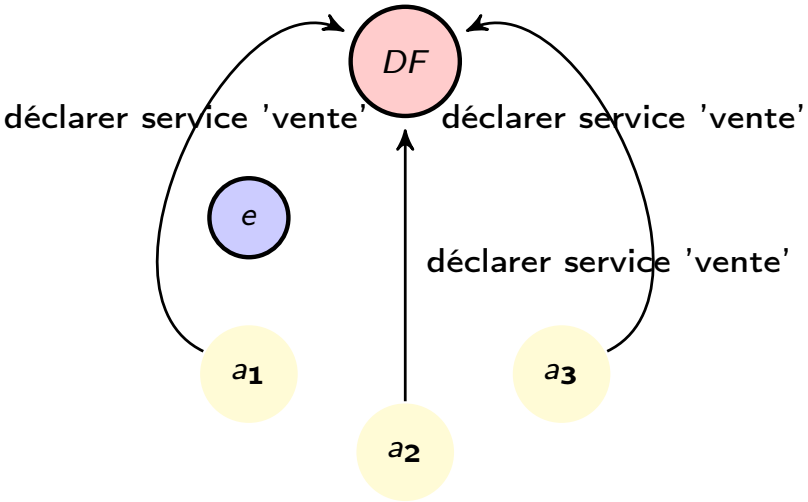
- La classe LaunchTopicAgents va se charger de définir les services et les agents, il suffit de lancer le main de la classe :

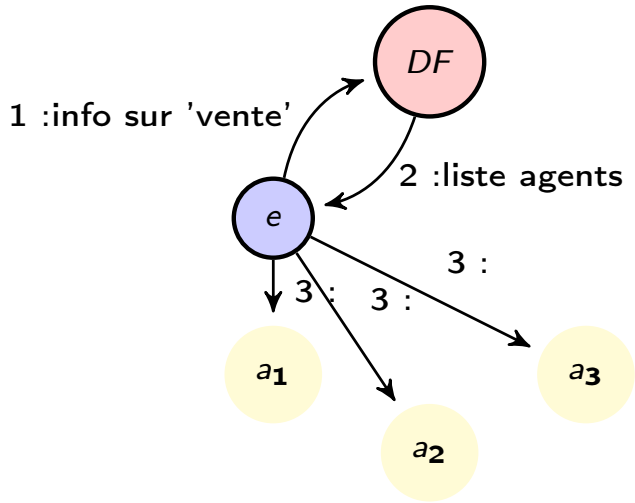
```
java LaunchTopicAgents
```

Résultat de l'exécution

```
Agent auditeur1 : recu ping0 du canal PingPongSansPong, emis par diffuseur
Agent auditeur1 : recu ping1 du canal PingPongSansPong, emis par diffuseur
Agent auditeur1 : recu ping2 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping0 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping1 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping2 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping3 du canal PingPongSansPong, emis par diffuseur
Agent auditeur1 : recu ping3 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping4 du canal PingPongSansPong, emis par diffuseur
Agent auditeur1 : recu ping4 du canal PingPongSansPong, emis par diffuseur
Agent auditeur2 : recu ping5 du canal PingPongSansPong, emis par diffuseur
Agent auditeur1 : recu ping5 du canal PingPongSansPong, emis par diffuseur
```

Communication à un groupe d'agents précis

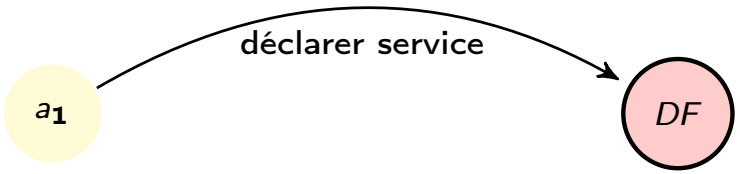




Groupement selon services rendus

- un agent déclare son/ses service/s auprès de l'agent pages jaunes (DF = Directory Facilitator Agent)
- possibilité de communiquer aux agents jouant le même service
  - par récupération auprès du DF des adresses des agents inscrits sous un service donné

Groupe d'agents par services



Enregistrer un agent sur les pages jaunes

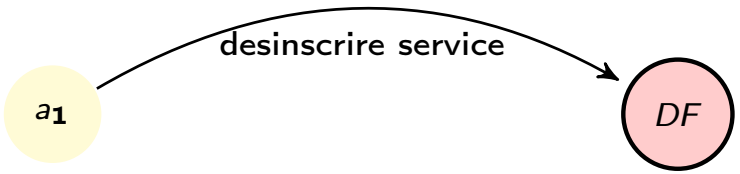
- Définir un carnet contenant les fiches de services
- un service possède un type et un nom (facultatif)

```
//definir le carnet de l'agent
DFAgentDescription carnet = new DFAgentDescription();

//definir une fiche pour chaque service
ServiceDescription service = new ServiceDescription();
service.setType("vente");
service.setName("gestionDeStock");
carnet.addServices(service);

//demander l'enregistrement sur les pages jaunes
try { DFService.register(this, carnet);}
catch (FIPAException fe) { fe.printStackTrace();}
```

Se désinscrire des pages jaunes



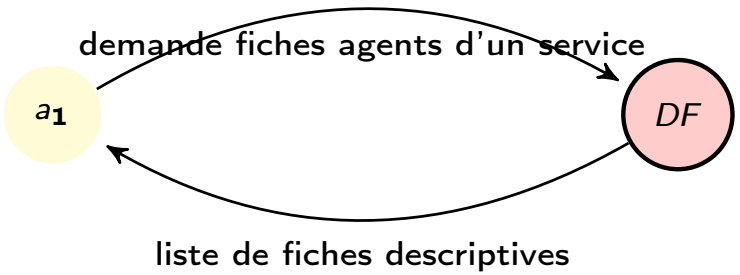
Se désinscrire en quittant la plateforme

- Surcharger la méthode `takeDown()` lancée automatiquement à la destruction de l'agent

```
protected void takeDown() {
 // S'effacer du service pages jaunes
 try { DFService.deregister(this); }
 catch (FIPAException fe) { fe.printStackTrace();}

 System.err.println("Agent : " + getLocalName() + " quitte la
     plateforme.");
}
```

Trouver les agents par services



Rechercher dans les pages jaunes

- Rechercher les agents offrant un même service

```
// NB. reprendre le carnet de l'agent defini precedemment (p. 52)
// result contiendra les descriptions des agents correspondant au carnet
List<AID> result = new ArrayList<>();
try {
 DFAgentDescription[] fiches=DFService.search(myAgent, carnet);
 if (fiches != null)
  for(DFAgentDescription fiche:fiches)
  //oter sa fiche de celles recuperees (si on cherche les agents de son service)
   if (!fiche.getName().equals(myAgent.getAID()))
     result.add(fiche.getName());
 } catch (FIPAException fe) { fe.printStackTrace(); }
```

Trouver les agents par services (écriture possible depuis Java 10)

Rechercher dans les pages jaunes

- Rechercher les agents offrant un même service

```
// NB. reprendre le carnet de l'agent defini precedemment (p. 52)
// result contiendra les adresses des agents correspondant au carnet
AID[] result = null;
try {
 var fiches = DFService.search(myAgent, carnet);
 if (fiches != null) {
  AID monAID = getAID();
  Stream<AID> fluxAID = Arrays.stream(fiches).
   map(DFAgentDescription::getName).
   filter(aid -> !aid.equals(monAID));
  result = fluxAID.toArray(AID[]::new);
 }
}
catch (FIPAException fe) { fe.printStackTrace(); }
```

Protocoles de communication en JADE

Protocoles de communication

BUT : fournir un cadre à l'échanges de messages dans un but précis

AchieveRE (Achieve Rational Effect) : un Initiateur envoie un message, le receveur **doit** répondre par `not-understood`, `refuse` ou `agree`. Suite à l'accord (`agree`), le receveur effectue l'action et retourne un message de type `inform` (réponse) ou `failure`.

SimpleAchieveRE : version simplifée de AchieveRE, un Initiateur envoie un message, le receveur **doit** répondre par `not-understood`, `refuse` ou `inform`.

Protocoles de communication en JADE

Protocoles de communication

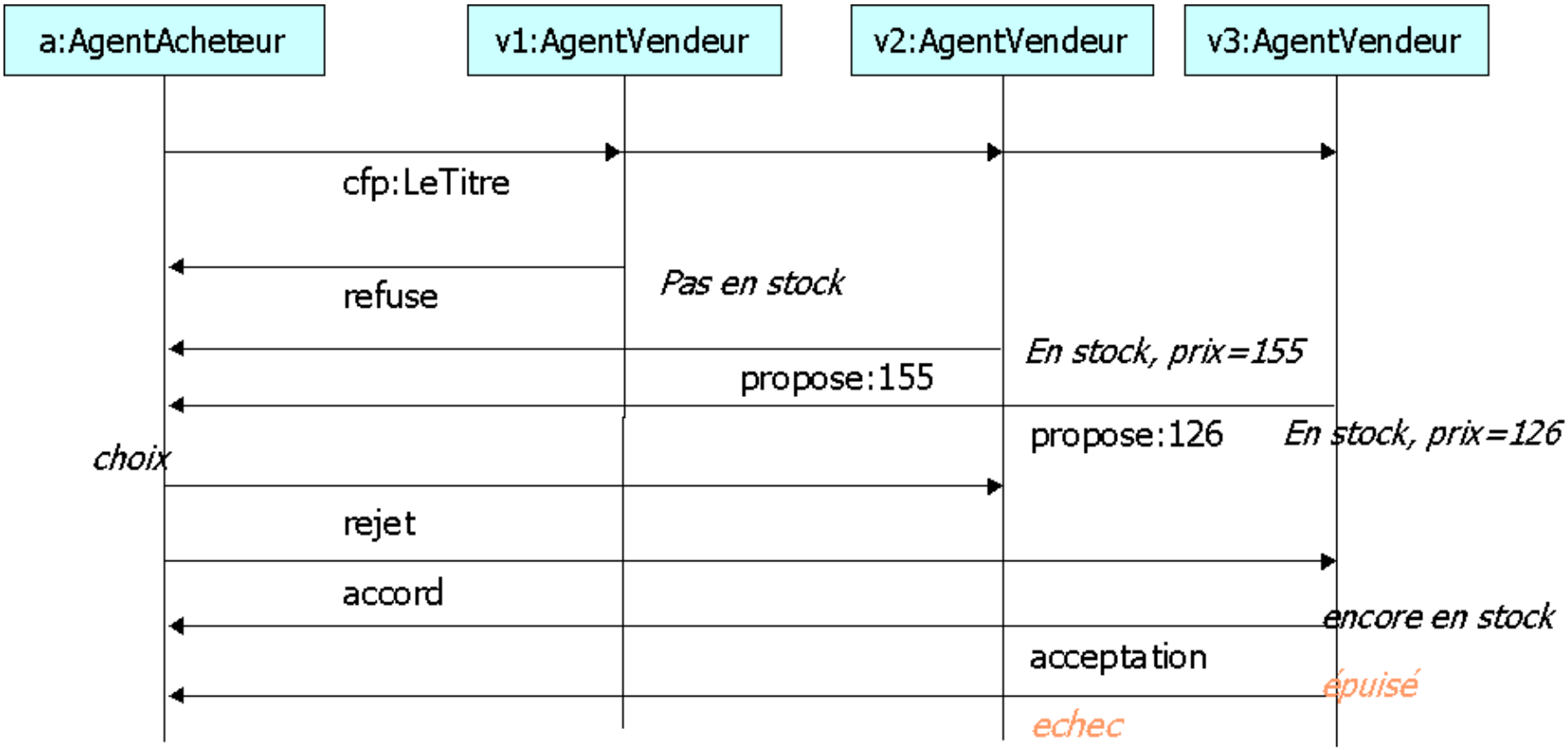
BUT : fournir un cadre à l'échanges de messages dans un but précis

FIPA - Contract NET : Un initiateur sollicite d'autres agents par un CFP, les receveurs **doivent** répondre par une proposition (`PROPOSE`), ou un message de type `REFUSE` ou `NOT-UNDERSTOOD`.
L'émetteur choisit parmi toutes les propositions reçues et envoie un message de type `ACCEPT_PROPOSAL` au candidat retenu.
Ce dernier retourne un message de type `INFORM` (accord), `FAILURE` ou `CANCEL` (annulation de l'offre)

Exemple de protocole CFP en JADE

Achat de livres par CFP

L'initiateur est l'acheteur, les participants (répondeurs) sont les vendeurs.



Protocoles de communication en JADE

Protocoles de communication

FIPA - Propose : un Initiateur envoie un message à un Participant lui indiquant qu'il effectuera une action si le Participant est d'accord (`agree`).
Le Participant répond par un refus ou un accord.
Lorsque l'accord est reçu, l'Initiateur doit effectuer l'action et retourner un résultat.

FIPA - Subscribe : un Initiateur envoie un message à un Participant lui demande s'il peut souscrire (`subscribe`).
Le participant répond par un accord (`agree`) ou un refus (`refuse`).
En cas d'accord, le Participant informe l'Initiateur à chaque fois que la condition apparaît.

Exemple d'utilisation de Subscribe I

```
public class AgentSearchService extends Agent{

protected void setup() {

 DFAgentDescription carnet = new DFAgentDescription();
 ServiceDescription fiche = new ServiceDescription();
 fiche.setType("vente"); fiche.setName("gestionDeStock");
 carnet.addServices(templateSd);

  //cree un message de souscription aux pages jaunes
  // chaque fois qu'un agent s'enregistre au service vente, gestionDe Stock
  // l'agent de type AgentSearchService sera averti

 ACLMessage msg = DFService.createSubscriptionMessage(this,
     getDefaultDF(), carnet, null);
```

Exemple d'utilisation de Subscribe II

```
List<AID> liste = new ArrayList<>();

addBehaviour(new SubscriptionInitiator(this, msg) {

 protected void handleInform(ACLMessage inform) {
  try {
   DFAgentDescription[] fiches = DFService.decodeNotification(
       inform.getContent());
   if (results.length > 0) {
    for (DFAgentDescription fiche:fiches) {
      AID nvellnscrit = fiche.getName(); liste.add(nvellnscrit);
      System.out.println(nvellnscrit.getName() + "␣s'est␣inscrit␣au
          ␣service.");
 } }
  } catch (FIPAException fe) { fe.printStackTrace(); }
 } } );
}
}
```

Initiateur d'un dialogue AchieveRE

méthodes principales d'un AchieveREInitiator

`void handleAgree(java.util.Vector responses)` pour chaque message de type 'AGREE' reçu

`void handleAllResponses(ACLMessage agree)` méthode appelée à la réception de toutes les réponses, ou après le délai imparti

`void handleFailure(ACLMessage failure)` pour chaque message de type 'FAILURE' reçu

`void handleInform(ACLMessage inform)` pour chaque message de type 'INFORM' reçu

`void handleNotUnderstood(ACLMessage notUnderstood)` pour chaque message de type 'NOT_UNDERSTOOD' reçu

`void handleOutOfSequence(ACLMessage msg)` pour chaque message tardif reçu

`void handleRefuse(ACLMessage refuse)` pour chaque message de refus, 'REFUSE' reçu

Exemple d'initiateur AchieveRE

```
public class AgentC extends Agent{
 protected void setup() {
  ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
  msg.setConversationId("123");
  msg.setContent("compute 4*5+3");
  msg.addReceiver(new AID("agentD", false));

  AchieveREInitiator init=new AchieveREInitiator(this, msg){
   protected void handleAgree(ACLMessage agree){
    System.out.println("recu un accord de " +
                       agree.getSender().getLocalName()); }

   protected void handleInform(ACLMessage inform){
    System.out.println("recu  de " +
      inform.getSender().getLocalName() + ", ce resultat " +
      inform.getContent());
   }
  };
  addBehaviour(init);
  }
}
```

Destinaire d'un dialogue AchieveRE

méthodes principales d'un AchieveREResponder

- **ACLMessage handleRequest(ACLMessage request)** méthode appelée à la réception d'un message. La fonction retourne la réponse.
- **ACLMessage prepareResultNotification(ACLMessage request, ACLMessage answer)** méthode appelée après handleRequest si la réponse envoyée a été AGREE, ou si aucune réponse n'a été renvoyé à l'initiateur. Doit retourner un message de type INFORM

Exemple de répondant AchieveRE I

```
public class AgentD extends Agent{

 protected void setup() {
  MessageTemplate model =
          MessageTemplate.MatchConversationId("123");

  AchieveREResponder respond = new AchieveREResponder(this, model){

   protected ACLMessage handleRequest(ACLMessage request){
    System.out.println("recu␣␣" + request.getContent() );
    ACLMessage answer = request.createReply();
    answer.setPerformative(ACLMessage.AGREE);
    return answer;
   }
```

Exemple de répondant AchieveRE II

```
    protected ACLMessage prepareResultNotification(ACLMessage
        request, ACLMessage response){
      String content = request.getContent();
      String[] terms = content.split(" ");
      String expression = terms[1];
      double result = new ExpressionBuilder(expression).build().
          evaluate();

      ACLMessage answer = request.createReply();
      answer.setPerformative(ACLMessage.INFORM);
      answer.setContent("resultat = " + result );
      System.out.println("j envoie " + result);
      return answer;
    } };

    addBehaviour(respond);
  }

}
```

Initiateur d'un dialogue SimpleAchieveRE

méthodes principales d'un SimpleAchieveREInitiator

Dialogue 1-1, l'emetteur n'attends pas de message AGREE

void handleFailure(ACLMessage failure) pour chaque message de type 'FAILURE' reçu

void handleInform(ACLMessage inform) pour chaque message de type 'INFORM' reçu

void handleNotUnderstood(ACLMessage notUnderstood) pour chaque message de type 'NOT_UNDERSTOOD' reçu

void handleOutOfSequence(ACLMessage msg) pour chaque message tardif reçu

void handleRefuse(ACLMessage refuse) pour chaque message de refus, 'REFUSE' reçu

Exemple d'initiateur AchieveRE

```
public class AgentEmissionSimpleARE extends AgentWindowed{
 protected void setup() {
   ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
   msg.setConversationId("123");
   msg.setContent("4*5+3");
   msg.addReceiver(new AID("b", false));

   SimpleAchieveREInitiator init = new SimpleAchieveREInitiator(
       this, msg){
    protected void handleInform(ACLMessage inform){
     System.out.println("recu  de " + inform.getSender().
         getLocalName() +  " ce contenu : " + inform.getContent());
   }};

   addBehaviour(new WakerBehaviour(this, 5000) {addBehaviour(init)
       ;}});
        }
```

Destinaire d'un dialogue SimpleAchieveRE

méthodes principales d'un SimpleAchieveREResponder

- **ACLMessage handleRequest(ACLMessage request)** méthode appelée à la réception d'un message. La fonction retourne la réponse de type `INFORM` (ou `NOT_UNDERSTOOD`)

Exemple de répondant SimpleAchieveRE I

```
public class AgentD extends Agent{
 protected void setup() {
  MessageTemplate model =
          MessageTemplate.MatchConversationId("123");

  SimpleAchieveREResponder respond = new SimpleAchieveREResponder(
      this, model){
   protected ACLMessage prepareResponse(ACLMessage request){
    String expression = request.getContent();
    double result = new ExpressionBuilder(expression).build().
        evaluate();
    ACLMessage answer = request.createReply();
    answer.setPerformative(ACLMessage.INFORM);
    answer.setContent("resultat = " + result );
    return answer;
   }
  };
  addBehaviour(respond);
 } }
```

Cas d'étude : achat de livres

Enoncé du cas d'étude (inspiré de la documentation Jade)

- Dans cet exemple, des agents vendent des livres et d'autres achètent des livres...
- Chaque agent acheteur reçoit le titre du livre à acheter (le livre cible) en argument et invite périodiquement tous les agents vendeur connus à fournir une offre.
- Dès qu'une offre est reçue, l'agent acheteur l'accepte et publie un ordre d'achat.
- Si plus d'un agent vendeur fournit une offre, l'agent acheteur accepte la meilleure (le plus bas prix).
- L'agent acheteur stoppe après avoir acheté le livre cible.
- Chaque agent vendeur a une GUI minimale permettant à l'utilisateur d'insérer de nouveaux titres (et les prix associés) dans le catalogue local des livres à vendre.
- Les agents vendeur attendent sans interruption des demandes des agents acheteur.
- Lorsque les agents acheteur sont invités à fournir une offre pour un livre, ils vérifient si le livre demandé est dans leur catalogue et répondent avec le prix. Autrement ils refusent.
- Quand ils reçoivent un ordre d'achat, ils l'exécutent et enlèvent le livre demandé de leur catalogue.

Définition d'un agent acheteur I

```
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;

public class AgentAcheteur extends Agent {
 /** titre du livre a acheter*/
 private String titreLivreCible;

 /** Initialisation de l'agent */
 protected void setup() {

  System.out.println("acheteur "+getAID().getName()+" est pret. ");

  Object[] args = getArguments(); // Recuperation des arguments
  if (args != null && args.length > 0) {
    titreLivreCible = (String) args[0];
```

Définition d'un agent acheteur II

```
        // Toutes les 1.5 sec, le comportement de demande d'achat (EffectuerDemande) est
            ajoute a la pile de comportement
        addBehaviour(new TickerBehaviour(this, 15000) {
         protected void onTick() {
          myAgent.addBehaviour(new EffectuerDemande( titreLivreCible));
         }
        } );
      } else {
          System.out.println("Aucun livre cible trouve...");
          // detruire l'agent
          doDelete();
      }
    }

    // 'Nettoyage' de l'agent
    protected void takeDown() {
     System.out.println("Agent_acheteur "+getAID().getName()+
     " quitte la plateforme. ");
            }
}
```

Définition d'un agent vendeur I

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import java.util.Hashtable;

public class AgentVendeur extends Agent {

 //Catalogue des livres a vendre (titre - prix)
 private Hashtable catalogue;
 //Interface Utilisateur pour lui permettre d'ajouter des livres
 private VendeurGui myGui;

 //Initialisation de l'agent
 protected void setup() {
  catalogue = new Hashtable();
  myGui = new VendeurGui(this);
  myGui.affiche();
```

Définition d'un agent vendeur II

```
// Enregistrer le service associe a l'agent aux pages jaunes
//decrire une description sur soi
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
//decrire un service
ServiceDescription sd = new ServiceDescription();
//le type de service
sd.setType("vente-livre");
//le nom (sous-type) du service
sd.setName("JADE-vente-livre");
dfd.addServices(sd);
// tenter l'enrefisgrement dans les pages jaunes
try { DFService.register(this, dfd); }
 catch (FIPAException fe) { fe.printStackTrace(); }

// Ajout du comportement de reponse a une demande d'offre
addBehaviour(new ReponseDemandeOffre(catalogue));
// Ajout du comportement de reponse a une demande d'achat
addBehaviour(new ReponseDemandeAchat(catalogue));
}
```

Définition d'un agent vendeur III

```
// Fermeture de l'agent
protected void takeDown() {
 // S'effacer du service pages jaunes
 try { DFService.deregister(this); }
 catch (FIPAException fe) { fe.printStackTrace(); }
 myGui.dispose();
 System.out.println("Vendeur: "+getAID().getName()+" part.");
}
/** Methode invoquee par l'ihm pour ajout de livre
* ajoute un comportement s'executant une seule fois
* consistant a mettre a jour la table  */
public void updateCatalogue(final String titre, final int prix) {
 addBehaviour(new OneShotBehaviour() {
  public void action() {
   catalogue.put(titre, new Integer(prix));
   System.out.println(titre +" insere au catalogue. Prix="+prix);
  } } );
 }
}
```

Cas d'étude - Comportement d'un acheteur

comportement et messages échangés , point de vue acheteur

- Le comportement d'un acheteur consiste à rechercher la liste des agents inscrits en tant que vendeurs, puis :
  1. à envoyer à tous les vendeurs une demande de proposition de prix,
  2. à réceptionner toutes les propositions,
  3. à choisir la meilleure offre et à envoyer un message d'acceptation de la proposition au vendeur retenu,
  4. à attendre la confirmation de la vente.
- le comportement prend fin après la 4ème étape

Cas d'étude - Comportement d'un acheteur I

```
import jade.core.AID;
import jade.core.behaviours.Behaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class EffectuerDemande extends Behaviour
{
 private AID meilleurVendeur;
 private int meilleurPrix;
 private int nbReponses = 0;
 private MessageTemplate mt;
 private int etape = 0;
 private DFAgentDescription[] agentsVendeurs;
 private String titreLivreCible;

 public EffectuerDemande( String _titreLivreCible)
```

Cas d'étude - Comportement d'un acheteur II

```
 { titreLivreCible = _titreLivreCible; }

// action coeur du comportement, Ã  4 Ã©tapes ici
public void action() {

// Recherche des agents vendeurs
// recherche des agents jouant le service du type vente-livre
DFAgentDescription modele = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("vente-livre");
modele.addServices(sd);
try {
 agentsVendeurs = DFService.search(myAgent, modele);
 System.out.println("agents vendeurs trouvÃ©s :");
 for (int i = 0; i < agentsVendeurs.length; ++i)
  System.out.println(agentsVendeurs[i].getName());
 }
 catch (FIPAException fe) { fe.printStackTrace(); }
```

Cas d'étude - Comportement d'un acheteur III

```
switch (etape) {

 case 0: // Envoyer le CFP Ã  tous les vendeurs
  ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
  for (int i = 0; i < agentsVendeurs.length; ++i)
     cfp.addReceiver(agentsVendeurs[i].getName());
  cfp.setContent(titreLivreCible);
  cfp.setConversationId("vente-livre");
  // placer un identifiant 'unique', ici cfp+date en ms
  cfp.setReplyWith("cfp"+System.currentTimeMillis());
  //envoyer le message
  myAgent.send(cfp);
  //dÃ©finir le filtre pour la rÃ©ception
  mt = MessageTemplate.and( MessageTemplate.MatchConversationId("
       vente-livre"), MessageTemplate.MatchInReplyTo(cfp.
       getReplyWith()));
  etape = 1;
 break;
```

Cas d'étude - Comportement d'un acheteur IV

```
case 1: // recevoir des reponses (propositions / refus) de tous les agents
// ne prendre en compte qu'un message correspondant au filtre dÃ©fini prÃ©cÃ©demment
 ACLMessage reponse = myAgent.receive(mt);
 if (reponse != null) {
  // si le message reÃ§u est une proposition
  if (reponse.getPerformative() == ACLMessage.PROPOSE) {
   int prix = Integer.parseInt(reponse.getContent());
   if (meilleurVendeur == null || prix < meilleurPrix) {
   // si elle est la meilleure, la mÃ©moriser
    meilleurPrix = prix;
    meilleurVendeur = reponse.getSender();
}    }
  nbReponses++;
  // si toutes les rÃ©pondes sont reÃ§ues, passer Ã  la suite
  if (nbReponses>= agentsVendeurs.length) etape = 2;
 }
 block(); //attendre un message
       break;
```

Cas d'étude - Comportement d'un acheteur V

```
case 2: // Envoyer l'acceptation au meilleur vendeur
 ACLMessage commande = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL)
      ;
 commande.addReceiver(meilleurVendeur);
 commande.setContent(titreLivreCible);
 commande.setConversationId("vente-livre");
 commande.setReplyWith("commande"+System.currentTimeMillis());
 myAgent.send(commande);
 // Preparation du filtre de la prochaine Ã©tape
 mt = MessageTemplate.and(MessageTemplate.MatchConversationId("
      vente-livre"), MessageTemplate.MatchInReplyTo(commande.
      getReplyWith()));
 etape = 3;
break;
```

Cas d'étude - Comportement d'un acheteur VI

```
 case 3: // reception de la confirmation d'achat
  reponse = myAgent.receive(mt);
  if (reponse != null) {
   if (reponse.getPerformative() == ACLMessage.INFORM) {
    // Achat reussi, fin de l'agent
    System.out.println(titreLivreCible +" achetÃ© avec succÃ¨s.");
    System.out.println("Prix = "+meilleurPrix);
    // destruction de l'agent en train de rÃ©aliser le comportement
    myAgent.doDelete();
   }
   etape = 4;
  }
   block(); // attendre un message
        break;
}}
```

Cas d'étude - Comportement d'un acheteur VII

```
//test de fin de comportement
public boolean done() {
//fin si pas de proposition ou si achat validÃ©
 if (etape == 2 && meilleurVendeur == null) {
  System.out.println(titreLivreCible + "␣indisponible␣a␣la␣vente
      ..."); }

                return ((etape == 2 && meilleurVendeur == null) ||
                    etape == 4);
        }

} // Fin de classe EffectuerDemande
```

Cas d'étude - Comportements d'un vendeur

comportement et messages échangés , point de vue vendeur

- Un vendeur possède deux comportement cyclique :
  1. attente et traitement d'une demande de proposition de vente
  2. attente et traitement d'une confirmation de d'achat
- Ces deux traitements fonctionnent "simultanément" car il peut exister plusieurs acheteurs et donc plusieurs actes de vente

Cas d'étude - Comportement d'un acheteur 1/2 I

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

import java.util.Hashtable;

//Comportement cyclique de reponse a une demande d'offre

public class ReponseDemandeOffre extends CyclicBehaviour {
 // catalogue present
 Hashtable catalogue;

 public ReponseDemandeOffre(Hashtable _catalogue)
  { catalogue = _catalogue; }

 // coeur du comportement, en 1 etape
 public void action() {
  // prendre en compte uniquement les messages de type CFP
  MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage
      .CFP);
```

Cas d'étude - Comportement d'un acheteur 1/2 II

```
  ACLMessage msg = myAgent.receive(mt);
  if (msg != null) {
   String titre = msg.getContent();
   // creer la reponse a la demande. le message est alors cree avec les champs appropries
        (receiver, sender, in-reply-to, ...)
   ACLMessage reponse = msg.createReply();
   Integer prix = (Integer) catalogue.get(titre);
   if (prix != null) {
   // Le Livre est disponible. Repondre avec le prix
   reponse.setPerformative(ACLMessage.PROPOSE);
   reponse.setContent(String.valueOf(prix.intValue()));
   }
   else {
   // livre non disponible a la vente, repondre un refus
   reponse.setPerformative(ACLMessage.REFUSE);
   reponse.setContent("indisponible");
   }
   myAgent.send(reponse);
  }
   block();     // attente d'un message
 }
}            // Fin de la classe interne ReponseDemandeOffre
```

Cas d'étude - Comportement d'un acheteur $_{1/2}$ III

Cas d'étude - Comportement d'un acheteur 2/2 I

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

import java.util.Hashtable;

public class ReponseDemandeAchat extends CyclicBehaviour {
 // catalogue present
 Hashtable catalogue;

 public ReponseDemandeAchat(Hashtable _catalogue)
 { catalogue =  _catalogue; }

 public void action() {
 // creation du filtre sur un message d'acceptation
  MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage
      .ACCEPT_PROPOSAL);
  ACLMessage msg = myAgent.receive(mt);
  if (msg != null) {
   String titre = msg.getContent();
```

Cas d'étude - Comportement d'un acheteur 2/2 II

```
    ACLMessage reponse = msg.createReply();
    Integer prix = (Integer) catalogue.remove(titre);
    if (prix != null) {
            // Le livre est toujours present, envoyer un message de confirmation
     reponse.setPerformative(ACLMessage.INFORM);
     System.out.println(titre+" vendu a l'agent "+msg.getSender().
          getName());
    }
    else {
     // Le livre demande a ete vendu a un autre acheteur entre temps
     reponse.setPerformative(ACLMessage.FAILURE);
     reponse.setContent("indisponible");
    }
    myAgent.send(reponse);
   }
   block(); // attente d'un message
  }
} // fin de ReponseDemandeAchat
```

Solution avec Protocole FIPA-ContractNet

Protocole FIPA-ContractNet

- Le protocole FIPA-ContractNet est composé de deux comportements pour la gestion des communications dans le cadre d'un appel à propositions :

  ContractNetInitiator : Initiant la demande et traitant les différents types de réponses. L'utilisation de ce comportement nécessite l'implémentation de fonctions déclenchées en fonction du type de message reçu.

  ContractNetResponder : permettant la gestion des demandes d'offres et des réponses associées. L'utilisation de ce comportement nécessite l'implémentation de fonctions déclenchées en fonction du type de message reçu.

Initiateur d'un ContractNet

méthodes principales d'un ContractNetInitiator

void handleAllResponses(Vector réponses, Vector acceptations) méthode appelée à la réception de toutes les réponses au cfp, ou après le délai imparti. acceptations est la liste des messages d'acceptations/rejets à retourner, automatiquement si autorisation est différent de 'null'.

void handleAllResultNotifications(Vector resultNotifications) méthode appelée quand tous les messages de notification de l'acceptation ont été reçus

void handleFailure(ACLMessage failure) pour chaque message de type 'FAILURE' reçu

void handleInform(ACLMessage inform) pour chaque message de type 'INFORM' reçu

void handleNotUnderstood(ACLMessage notUnderstood) pour chaque message de type 'NOT_UNDERSTOOD' reçu

void handleOutOfSequence(ACLMessage msg) pour chaque message tardif reçu

void handlePropose(ACLMessage proposition, Vector acceptations) méthode appelée à chaque proposition reçue. acceptations est la liste des messages d'acceptations/rejets à retourner.

void handleRefuse(ACLMessage refuse) pour chaque message de refus, 'REFUSE' reçu

Participant à un ContractNet

méthodes principales d'un ContractNetResponder

handleCfp ACLMessage handleCfp(ACLMessage cfp) : méthode appelée à la réception du premier message dans le cadre d'un CFP. Retourne le message de réponse à l'initiateur (un message de type autre que INFORM termine le protocole).

handleAcceptProposal ACLMessage handleAcceptProposal(ACLMessage cfp, ACLMessage proposition, ACLMessage acceptation)] : méthode appelée à la réception d'un message d'acceptation. retourne le message de confirmation/infirmation à l'initiateur

Solution avec Protocole FIPA-ContractNet

Cas d'étude

- Les classes définissants l'agent acheteur et l'agent vendeur sont légèrement modifiées :
  - L'agent acheteur reçoit alors périodiquement le comportement ContractNetAchat
  - L'agent vendeur reçoit alors le comportement ContractNetVente et se déclare en tant que membre du service "vente_livre"

Cas d'étude - protocole initiateur de l'acheteur I

```
import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.proto.ContractNetInitiator;

import java.util.Date;
import java.util.Vector;

public class ContractNetAchat extends ContractNetInitiator{
 int nbRepondants = 0;
 String titreLivreCible;
```

Cas d'étude - protocole initiateur de l'acheteur II

```
//Constructeur
public ContractNetAchat(Agent agent, ACLMessage msg, String
    _titreLivreCible)
{
 super(agent, msg);
 titreLivreCible = _titreLivreCible;
 // Recherche / Mise a jour des agents vendeurs
 DFAgentDescription modele = new DFAgentDescription();
 ServiceDescription sd = new ServiceDescription();
 sd.setType("vente_livre");
 modele.addServices(sd);
 DFAgentDescription[] agentsVendeurs = null;

 try {
  agentsVendeurs = DFService.search(myAgent, modele);
  System.out.println("agents␣vendeurs␣:");
  for (int i = 0; i < agentsVendeurs.length; ++i)
   System.out.println(agentsVendeurs[i].getName());
                }
 catch (FIPAException fe) { fe.printStackTrace(); }

 nbRepondants = agentsVendeurs.length;
```

Cas d'étude - protocole initiateur de l'acheteur III

```
  for (int i = 0; i < agentsVendeurs.length; i++)
   msg.addReceiver(agentsVendeurs[i].getName());

  msg.setProtocol(FIPANames.InteractionProtocol.FIPA_CONTRACT_NET);
  // Reponse plus tard dans 10 secs
  msg.setReplyByDate(new Date(System.currentTimeMillis() + 10000));
  msg.setContent(titreLivreCible);
  this.reset(msg);
}
```

Cas d'étude - protocole initiateur de l'acheteur IV

```
//prise en compte d'une proposition
protected void handlePropose(ACLMessage propose, Vector v) {
  System.out.println("Agent "+propose.getSender().getName()+" propose "+propose.getContent());
}

//prise en compte d'un refus
protected void handleRefuse(ACLMessage refuse) {
  System.out.println("Agent "+refuse.getSender().getName()+" refuse");
}

// prise ne compte d'un message d'information
protected void handleInform(ACLMessage inform) {
  System.out.println("Agent "+inform.getSender().getName()+" a effectue l'action avec succes");
}
```

Cas d'étude - protocole initiateur de l'acheteur V

```
//prise en compte d'une retour d'erreur
protected void handleFailure(ACLMessage failure) {
 if (failure.getSender().equals(myAgent.getAMS())) {
  // ERREUR : le destinataire n'existe pas
  System.out.println("Le destinataire n'existe pas...");
 }
 else
  System.out.println("Agent "+failure.getSender().getName()+" a
      echoue");
 // nombre de repondants decremente
 nbRepondants --;
}
```

Cas d'étude - protocole initiateur de l'acheteur VI

```
// prise en compte de l'ensemble des reponses
protected void handleAllResponses(Vector reponses, Vector
    acceptations) {
 if (reponses.size() < nbRepondants ) {
  // Si quelques vendeurs n'ont pas repondu a temps
  System.out.println("Temps d'attente expire : il manque "+(
      nbRepondants - reponses.size())+" rÃ©ponses");
 }
 // Evaluer les propositions
 int meilleurPrix = -1;
 AID meilleurVendeur = null;
 ACLMessage accept = null;
 for(int i=0; i<reponses.size(); i++)
 {
  ACLMessage msg = (ACLMessage) reponses.get(i);
  if (msg.getPerformative() == ACLMessage.PROPOSE) {
        // creer une reponse de refus par defaut pour chaque vendeur
   ACLMessage reponse = msg.createReply();
   reponse.setContent(titreLivreCible);
   reponse.setPerformative(ACLMessage.REJECT_PROPOSAL);
```

Cas d'étude - protocole initiateur de l'acheteur VII

```
        acceptations.addElement(reponse);
        // chercher la meilleure propositions
        int prix = Integer.parseInt(msg.getContent());
        if (meilleurVendeur == null || prix < meilleurPrix) {
          meilleurPrix = prix;
          meilleurVendeur = msg.getSender();
          accept = reponse;
    }}}
    // Accepter la meilleure offre, modifier le type de son message
    if (accept != null) {
      System.out.println("Accepte la proposition a " + meilleurPrix +"
          de l'agent "+meilleurVendeur.getName());
      accept.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
}}}
// remarque : les reponses du vecteur reponses sont prises automatiquement de la file
// les messages du vecteur acceptations sont automatiquement envoyes
```

Cas d'étude - protocole répondeur du vendeur I

```
//comportement de reponse dans le protocole contractNet
public class ContractNetVente extends  ContractNetResponder
{
 //catalogue du vendeur
 Hashtable catalogue;
 public ContractNetVente(Agent agent, MessageTemplate template,
     Hashtable _catalogue)
 { super(agent, template); catalogue = _catalogue; }
```

Cas d'étude - protocole répondeur du vendeur II

```
//prepare une reponse a un appel d'offre
protected ACLMessage prepareResponse(ACLMessage cfp) throws
    NotUnderstoodException, RefuseException {

 System.out.println("Agent␣"+myAgent.getLocalName()+":␣CFP␣recu␣de
     ␣"+cfp.getSender().getName()+".␣Sujet␣"+cfp.getContent());
 String titre = cfp.getContent();
 Integer prix = (Integer) catalogue.get(titre);

 if (prix != null) {
  // Le Livre est disponible. Repondre avec le prix
  System.out.println("Agent␣"+myAgent.getLocalName()+":␣Propose␣"+
      prix);
  ACLMessage propose = cfp.createReply();
  propose.setPerformative(ACLMessage.PROPOSE);
  propose.setContent(String.valueOf(prix.intValue()));
  return propose;
 }
 else {
  // Le livre n'est pas disponible a la vente, repondre avec un refus
```

Cas d'étude - protocole répondeur du vendeur III

```
    System.out.println("Agent␣"+myAgent.getLocalName()+":␣Refus,␣pas
        ␣de␣livre␣'" + titre + "'␣en␣catalogue...");
    throw new RefuseException("evaluation-a-echoue");
  }
}
```

Cas d'étude - protocole répondeur du vendeur IV

```
//gestion d'une acceptation de l'offre
protected ACLMessage prepareResultNotification(ACLMessage cfp,
    ACLMessage propose,ACLMessage accept) throws FailureException
{
 String titre = accept.getContent();
 Integer prix = (Integer) catalogue.remove(titre);
 if (prix != null){
  System.out.println("Agent "+myAgent.getLocalName()+": Action
      effectuee avec succes");
  ACLMessage inform = accept.createReply();
  inform.setPerformative(ACLMessage.INFORM);
  return inform;
 } else {
  System.out.println("Agent "+myAgent.getLocalName()+": Action en
      echec - livre epuise entre temps...");
  throw new FailureException("unexpected-error");
 }
}


// gestion d'un refus de l'offre
```

Cas d'étude - protocole répondeur du vendeur V

```
 protected void handleRejectProposal(ACLMessage cfp, ACLMessage
     propose, ACLMessage reject) {
  System.out.println("Agent␣"+myAgent.getLocalName()+":␣Proposition
      ␣rejetee");
 }
}
```