La Programmation Orientée Agent

FIPA - Jade

Emmanuel ADAM

Université de Valenciennes et du Hainaut-Cambrésis

UVHC/ISTV-LAMIH

1 FIPA - Généralités

2 JADE
- Généralités
- Architecture
- Outils
- Agents
- Comportements
- Communication
- 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
- Définir les entités architecturales et leurs relations
  - Transport de messages, langage de communication,
  - services de renseignements, ...

Transports de messages

- But : gérer la livraison des messages par des protocoles de communications intra et inter-plateformes
  - un MTS (Message Transport Service) sur chaque plateforme
  - Message = enveloppe + corps

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

Groupes de travail FIPA

- Agents and Web Services Interoperability Working Group (AWSI WG)
- Human-Agent Communications Working Group (HAC WG)
- Mobile Agents Working Group (MA WG)
- P2P Nomadic Agents Working Group (P2PNA WG)

Généralités sur FIPA (Foundation of Intelligent Physical Agents)

Plateforme FIPA

- 16 plateformes "FIPA-compliant", dont 9 du domaine public, dont :
  - ZEUS (British Telecoms)
  - JADE (CSELT)
  - FIPA-OS (Emorphia)
  - April Agent Platform (Fujitsu Labs of America)
  - LEAP (Lightweight Extensible Agent Platform)
  - 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

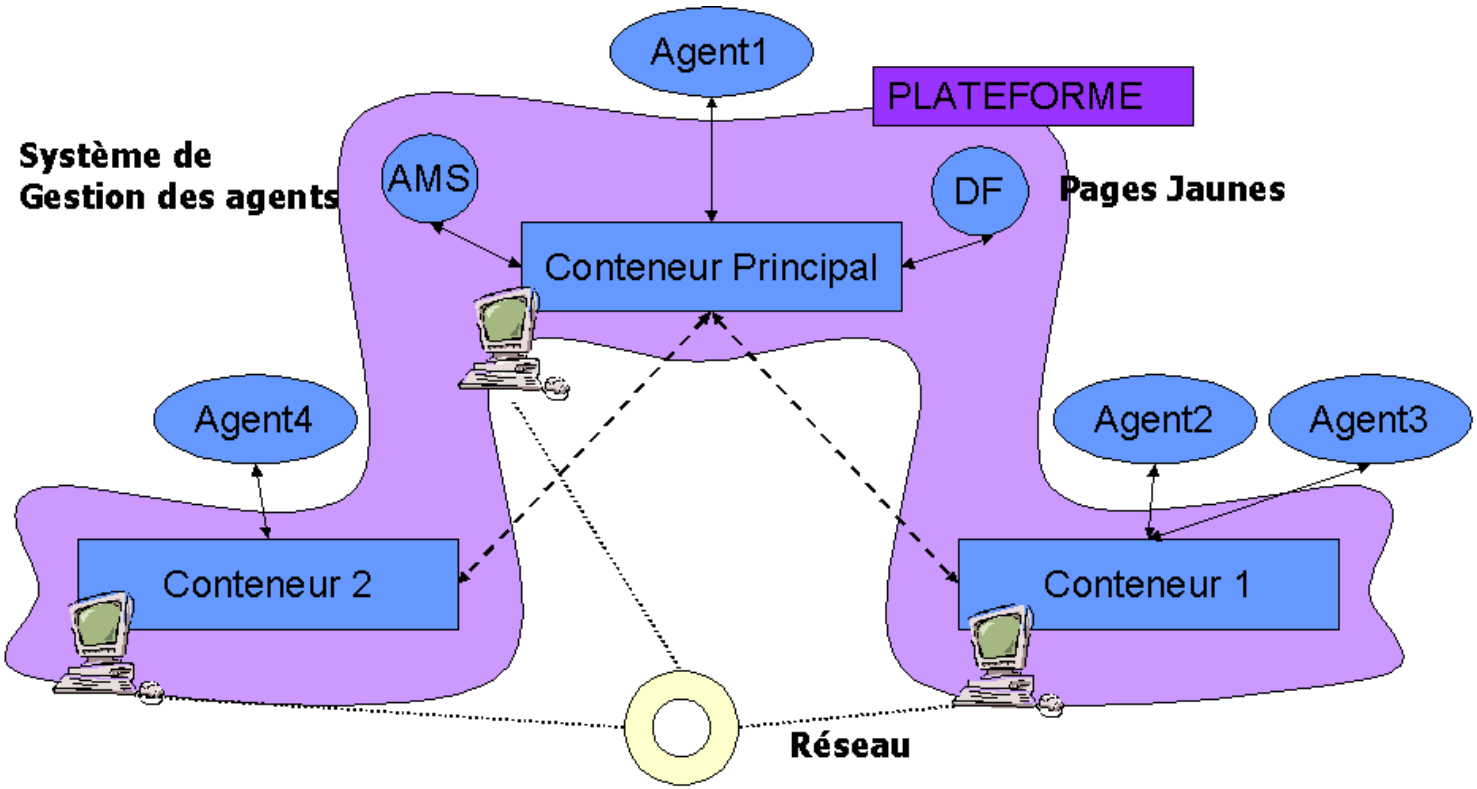
Chaque conteneur d'agents :

- 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

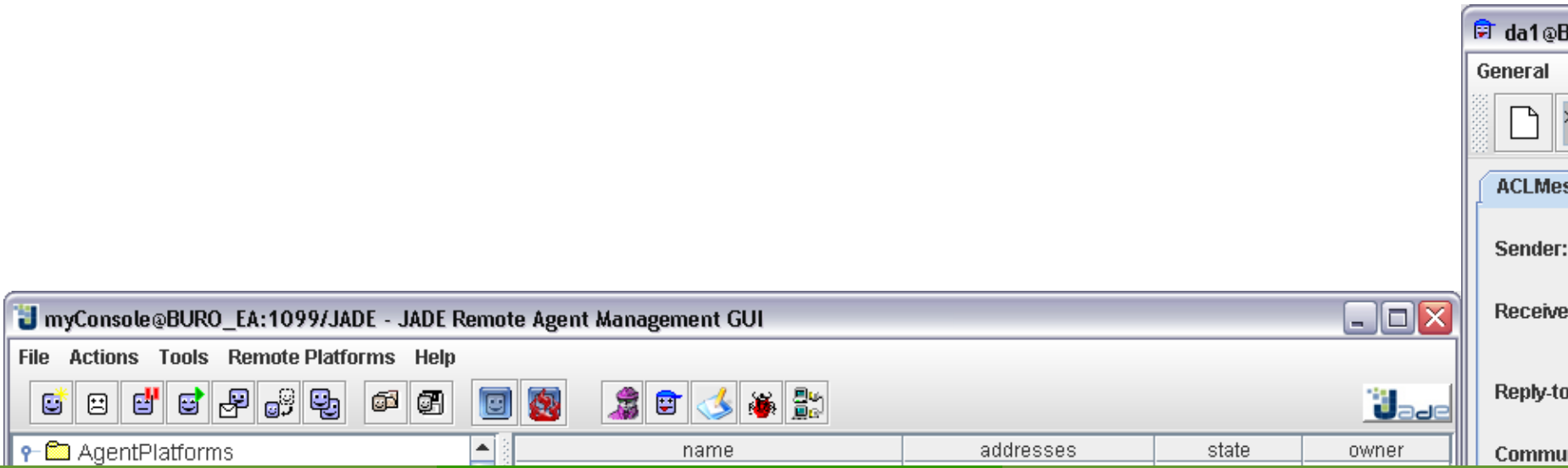


Outils de JADE

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

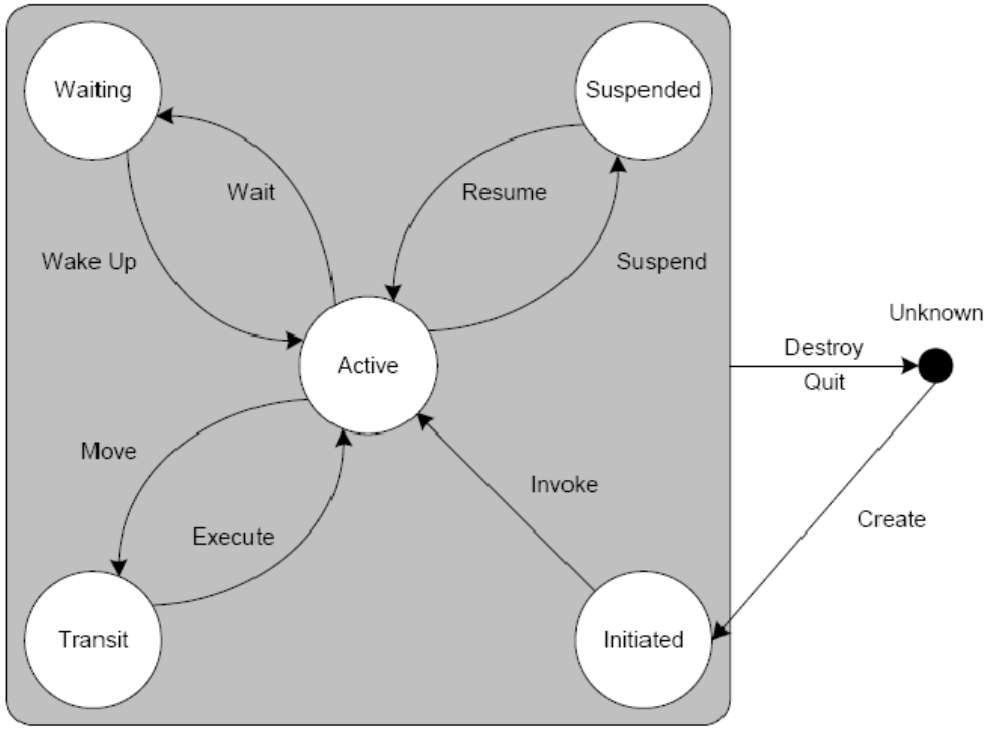


Figure 3 - Agent life-cycle as defined by FIPA.

Agents en JADE

Agent JADE - Elément de programmation

- hérite de la classe Agent
- Thread
- Possibilité de s'inscrire/rechercher un *service*
  - Action enregistrée et dispensée par la plateforme
  - Comportement d'un ou de plusieurs agents répondant à une demande
  - 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)

Comportements d'agents en JADE

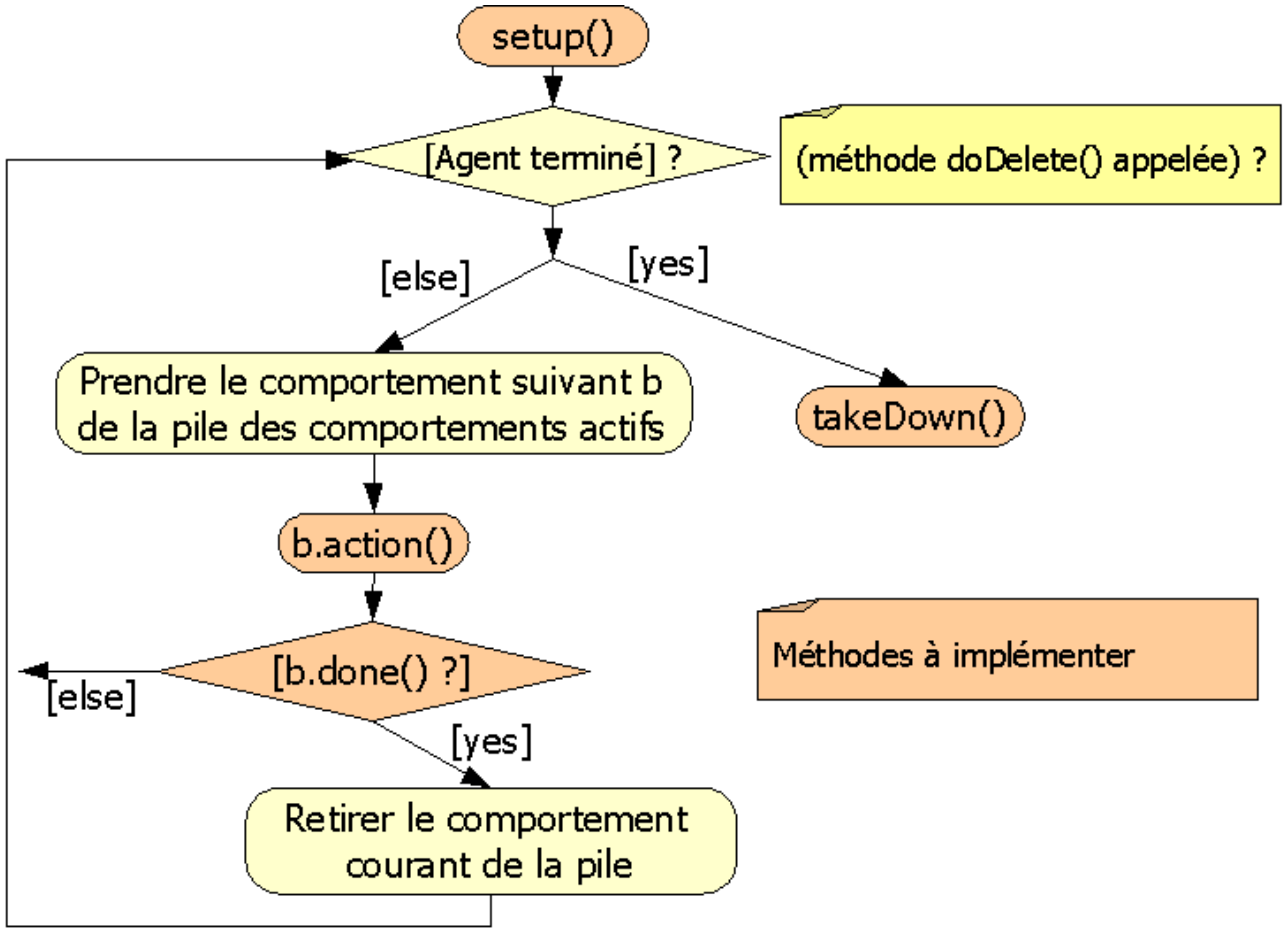
Comportement JADE - Élément de programmation

- 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,
      N.B. dans un agent, tous les comportements bloqués sont reprogrammés dans la file dès qu'un message est reçu
    - la date limite de blocage préfixée auparavant,
    - 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

Les types de comportements

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); }
}
```

Les types de comportements

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 tâches */ }
}
```

Les types de comportements

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 tâches */ }
}
```

Les comportements complexes

Autres comportements

- comportements incluant de sous-comportements :

  SequentialBehavior : enchaînement de comportements

  ParallelBehavior : exécution de comportements en concurrence

  FSMBehavior : exécution de comportement selon une Machine d'Etats Finis (Finished State Machine) définie par le programmeur

- comportements prédéfinis :

  SenderBehavior : comportement one-shoot qui effectue l'envoie d'un message

  ReceiverBehavior : effectue la réception d'un message.

  WakerBehavior : effectue une tâche après un délai

  TickerBehavior : effectue une tâche cycliquement en effectuant des pauses.

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 entre agents en JADE

Communication JADE - Elément de programmation

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

```
// Message transmettant une demande d'offre
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
cfp.addReceiver(new AID("vendeur", AID.ISLOCALNAME));
cfp.setContent("La proie");
myAgent.send(cfp);
```

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

Méthode non bloquante, retourne le premier message de la boîte, null si elle est vide

```
// prendre un message de la file
ACLMessage msg = receive();
if (msg != null) {/* traitement du message */}
```

Protocoles de communication entre agents 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 peut répondre par `not-understood`, `refuse` ou `agree`. Suite à l'accord (`agree`), le receveur retourne un message de type `inform` (réponse) ou `failure`.

FIPA - Contract NET : Un initiateur sollicite d'autres agents par un CFP, les receveurs peuvent 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)

Protocoles de communication entre agents en JADE

Protocoles de communication

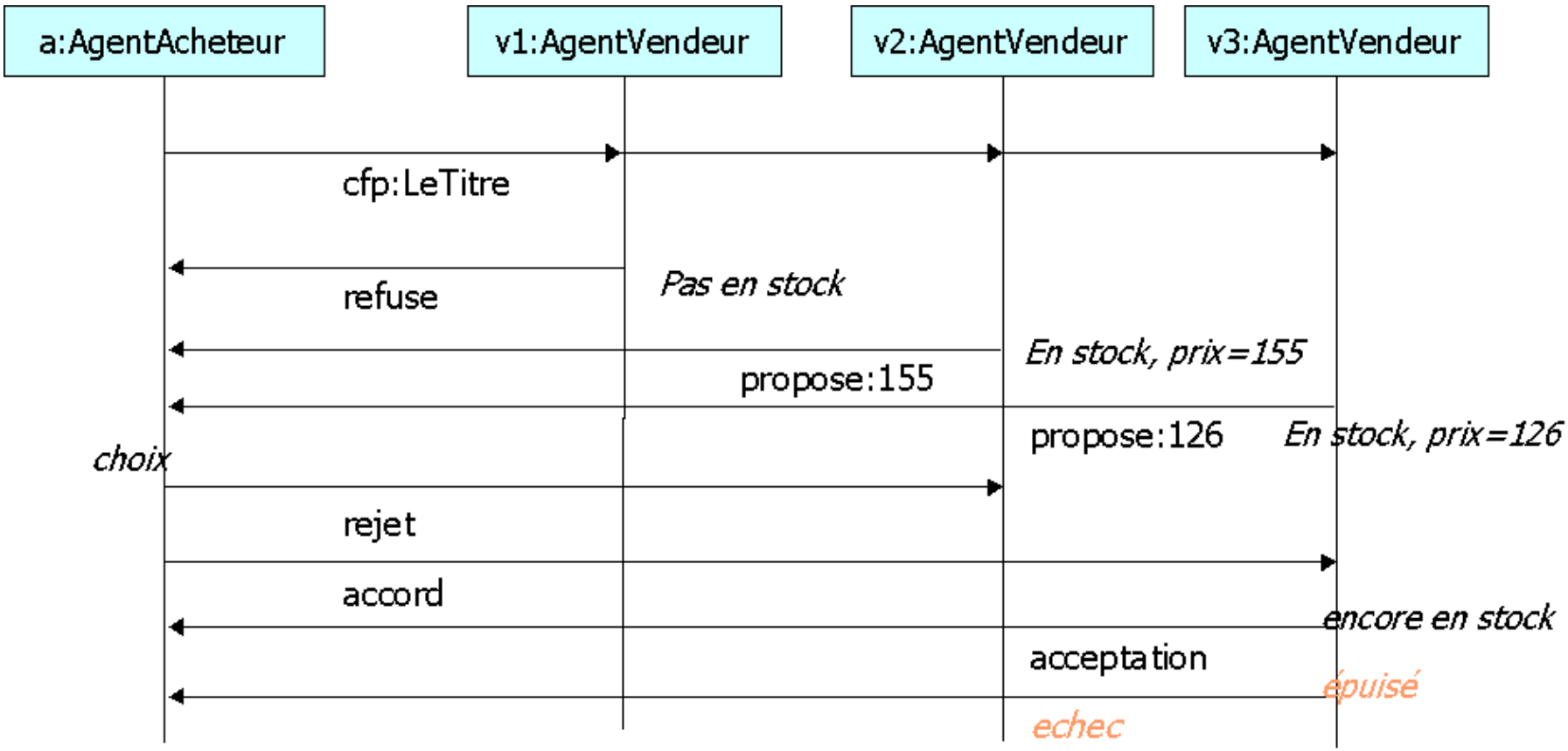
FIPA - Propose : un Initiateur envoie un message à un Participant lui indiquant qu'il effectuera une action si le Participant est d'accord (`agree`). Le Participant répond par un refus ou un accord. Lorsque l'accord est reçu, l'Initiateur doit effectuer l'action et retourner un résultat.

FIPA - Subscribe : un Initiateur envoie un message à un Participant lui demande s'il souhaite souscrire (`subscribe`). Le participant répond par un accord (`agree`) ou un refus (`refuse`). En cas d'accord, le Participant envoie les informations répondant à la souscription jusqu'à ce que l'Initiateur annule la souscription ou que le Participant émette un message de type `failure`.

Exemple de protocole CFP en JADE

Achat de livres par CFP

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



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.

Définition d'un agent acheteur I

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

public class AgentAcheteur extends Agent {
 /** titre du livre à 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];

    // Toutes les 1.5 sec, le comportement de demande d'
        achat (EffectuerDemande) est ajouté à la pile de
        comportement
```

Définition d'un agent acheteur II

```
        addBehaviour(new TickerBehaviour(this, 15000) {
         protected void onTick() {
          myAgent.addBehaviour(new EffectuerDemande(
                titreLivreCible));
         }
        } );
     } else {
                System.out.println("Aucun livre cible trouve...");
                // détruire 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 à 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
//décrire une description sur soi
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
//décrire 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 à une demande d'offre
addBehaviour(new ReponseDemandeOffre(catalogue));
// Ajout du comportement de reponse à 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.")
     );
}

/** Méthode invoquée par l'ihm pour ajout de livre
* ajoute un comportement s'exécutant une seule fois
* consistant à mettre à 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);
  } } );
```

Définition d'un agent vendeur IV

```
    }
}
```

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)
  { titreLivreCible = _titreLivreCible; }
```

Cas d'étude - Comportement d'un acheteur II

```
// action coeur du comportement, à 4 étapes 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 trouvés :");
 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 à 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);
  //définir le filtre pour la réception
  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 défini précédemment
 ACLMessage reponse = myAgent.receive(mt);
 if (reponse != null) {
  // si le message reçu 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 mémoriser
    meilleurPrix = prix;
    meilleurVendeur = reponse.getSender();
 }   }
  nbReponses++;
  // si toutes les répondes sont reçues, passer à la suite
  if (nbReponses>= agentsVendeurs.length) etape = 2;
 }
 else block(); //sinon, 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 étape
 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 +" acheté avec
        succès.");
    System.out.println("Prix = "+meilleurPrix);
    // destruction de l'agent en train de réaliser le
        comportement
    myAgent.doDelete();
   }
   etape = 4;
  }
  else block(); //sinon, 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 validé
   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 réponse à une demande d'offre

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

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

 // coeur du comportement, en 1 étape
 public void action() {
  // prendre en compte uniquement les messages de type CFP
```

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

```
MessageTemplate mt = MessageTemplate.MatchPerformative(
    ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
 String titre = msg.getContent();
 // créer la réponse à la demande. le message est alors
     créé avec les champs appropriés (receiver, sender, in
     -reply-to, ...)
 ACLMessage reponse = msg.createReply();
 Integer prix = (Integer) catalogue.get(titre);
 if (prix != null) {
 // Le Livre est disponible. Répondre avec le prix
 reponse.setPerformative(ACLMessage.PROPOSE);
 reponse.setContent(String.valueOf(prix.intValue()));
 }
 else {
 // livre non disponible à la vente, répondre un refus
 reponse.setPerformative(ACLMessage.REFUSE);
 reponse.setContent("indisponible");
 }
```

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

```
      myAgent.send(reponse);
    }
    else block();    // attente d'un message
  }
}          // Fin de la classe interne ReponseDemandeOffre
```

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 présent
 Hashtable catalogue;

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

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

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

```
String titre = msg.getContent();
ACLMessage reponse = msg.createReply();

Integer prix = (Integer) catalogue.remove(titre);
if (prix != null) {
    // Le livre est toujours présent, 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);
}
else
```

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

```
        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 envoyés automatiquement par Jade en sortie de fonction.

void handleFailure(ACLMessage failure)
: appelée à chaque réception de message de type 'FAILURE'

void handleFailure(ACLMessage inform)
: appelée à chaque réception de message de type 'INFORM'

void handleOutOfSequence(ACLMessage msg)
: appelée à chaque réception de message hors délai

void handleRefuse(ACLMessage refuse)
: appelée à chaque réception de message de type 'REFUSE'

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

- 'cfp' est le message d'appels à proposition initial
- 'proposition' est le message contenant la proposition faite par le répondant (donc par l'agent lui-même)
- 'acceptation' est le message d'acceptation envoyé par l'initiateur de l'appel à propositions

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(); }
```

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

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

    msg.setProtocol(FIPANames.InteractionProtocol.
        FIPA_CONTRACT_NET);
    // Réponse 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

}

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

```
//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 répondants décrémenté
        nbRepondants --;
}
```

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

```
// prise en compte de l'ensemble des réponses
protected void handleAllResponses(Vector reponses, Vector
    acceptations) {
 if (réponses.size() < nbRepondants ) {
  // Si quelques vendeurs n'ont pas répondu à temps
  System.out.println("Temps d'attente expiré : il manque "
      +(nbRepondants - réponses.size())+" réponses");
 }
 // 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) {
       // créer une reponse de refus par défaut pour chaque
           vendeur
   ACLMessage reponse = msg.createReply();
```

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

```
        reponse.setContent(titreLivreCible);
        reponse.setPerformative(ACLMessage.REJECT_PROPOSAL);
        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 réponses du vecteur reponses sont prises
    automatiquement de la file
```

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

```
// les messages du vecteur acceptations sont automatiquement
    envoyés
```

Cas d'étude - protocole répondeur du vendeur I

```
import jade.core.Agent;
import jade.domain.FIPAAgentManagement.FailureException;
import jade.domain.FIPAAgentManagement.
    NotUnderstoodException;
import jade.domain.FIPAAgentManagement.RefuseException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.proto.ContractNetResponder;
import java.util.Hashtable;

//comportement de réponse dans le protocole contractNet
public class ContractNetVente extends  ContractNetResponder
{
 //catalogue du vendeur
 Hashtable catalogue;

 public ContractNetVente(Agent agent, MessageTemplate
     template, Hashtable _catalogue)
 {
```

Cas d'étude - protocole répondeur du vendeur II

```
    super(agent, template);
    catalogue = _catalogue;
  }
```

Cas d'étude - protocole répondeur du vendeur III

```
//prépare une réponse à un appel d'offre
protected ACLMessage handleCfp(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;
 }
```

Cas d'étude - protocole répondeur du vendeur IV

```
    else {
     // Le livre n'est pas disponible à la vente, répondre
          avec un refus
     System.out.println("Agent "+myAgent.getLocalName()+": "
          Refuse, pas de livre '" + titre + "' en catalogue..."
          );
     throw new RefuseException("evaluation-a-echoue");
    }
  }
```

Cas d'étude - protocole répondeur du vendeur V

```
//gestion d'une acceptation de l'offre
protected ACLMessage handleAcceptProposal(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");
 }
}
```

Cas d'étude - protocole répondeur du vendeur VI

```
    // gestion d'un refus de l'offre
    protected void handleRejectProposal(ACLMessage cfp,
        ACLMessage propose, ACLMessage reject) {
     System.out.println("Agent "+myAgent.getLocalName()+": 
         Proposition rejetee");
    }
}
```