Les Behaviours dans JADE

par Youghourta BENALI

Date de publication : 7 juin 2009

Après avoir vu comment créer un agent JADE sous eclipse **ici**. Nous allons voir dans cet article comment utiliser les comportements (behaviours), l'un des aspects les plus importants de la programmation orientée agent.

I - Introduction....3
II - Vue générale....3
III - Les Behaviours du JADE....3
III-A - Les Behaviours simples....3
III-A-1 - One-shot Behaviour....3
III-A-2 - Cyclic Behaviour....3
III-A-3 - Generic Behaviour....3
III-A-4 - Exemple....4
III-B - Les Behaviours planifiés....4
III-B-1 - WakerBehaviour....4
III-B-2 - TickerBehaviour....5
III-B-3 - Exemple....5
III-C - Les Behaviours composés....5
III-C-1 - SequentielBehaviour....5
III-C-2 - FSMBehaviour....6
III-C-3 - ParallelBehaviour....8
III-D - Combiner plusieurs Behaviours composés....9
IV - Conclusion....10
IV-A - Bibliographie....10
IV-B - Remerciements....10

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

I - Introduction

Dans la programmation orientée agent, les différentes tâches d'un agent doivent obéir à certaines règles et doivent être écrite sous une forme compréhensible par la plateforme utilisée. Nous allons voir dans ce qui suit comment définir ces tâches et les utiliser dans JADE

II - Vue générale

Pour qu'un agent JADE exécute une tâche, nous avons tout d'abord besoin de définir ces tâches. Les tâches dans JADE (appelées des **behaviours** ou des **comportements**) sont des instances de la classe **jade.core.behaviours** . pour qu'un agent exécute une tâche on doit lui l'attribuer par la méthode **addBehaviour(Behaviour b)** de la classe **jade.core.Agent**.
Chaque Behaviour doit implémenter au moins les deux méthodes :

- **action() :** qui désigne les opérations à exécuter par le Behaviour;
- **done() :** qui exprime si le Behaviour a terminé son exécution ou pas.

Il existe deux autres méthodes dont l'implémentation n'est pas obligatoire mais qui peuvent être très utiles :

- **onStart() :** appelée juste avant l'exécution de ma méthode action();
- **onEnd() :** appelée juste après la retournement de **true** par la méthode done().

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

III - Les Behaviours du JADE

III-A - Les Behaviours simples

JADE offre trois types de Behaviours simple. Ces Behaviours sont :

III-A-1 - One-shot Behaviour

Un one-shot Behaviour est une instance de la classe **jade.core.behaviours.OneShotBehaviour**. Il a la particularité d'exécuter sa tâche une et une seule fois puis il se termine. La classe OneShotBehaviour implémente la méthode **done()** et elle retourne toujours **true**.

III-A-2 - Cyclic Behaviour

Un cyclic Behaviour est une instance de la classe **jade.core.behaviours.CyclicBehaviour**. Comme son nom l'indique un cyclic Behaviour exécute sa tâche d'une manière répétitive. La classe CyclicBehaviour implémente la méthode **done()** qui retourne toujours **false**.

III-A-3 - Generic Behaviour

Un Generic Behaviour est une instance de la classe **jade.core.behaviours.Behaviour**.Le Generic Behaviour vient entre le One-shot Behaviour et le Cyclic Behaviour de faite qu'il n'implémente pas la méthode done() et laisse son implémentation au programmeur, donc il peut planifier la terminaison de son Behaviour selon ces besoin.

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

III-A-4 - Exemple

Voici un exemple sur l'utilisation des Behaviours simples

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;

public class SimpleAgent extends Agent {
  protected void setup() {
 // l'ajout d'un one-shot behaviour pour afficher un Hello world :D
   addBehaviour(new OneShotBehaviour(this){
   public void action(){
       System.out.println("Bonjour tous le monde je suis l'agent "+getLocalName());
    }
   });

    // l'ajout d'un  CyclicBehaviour pour afficher un message à chaque fois qu'il s'execute
  addBehaviour(new CyclicBehaviour(this) {
   public void action() {
    System.out.println("cyclique... ");
      }
    });

    // l'ajout d'un generic behaviour
    // le Behaviour s'arrête quand aléatoire reçoit la valeur 7
    addBehaviour(new RandomBehaviour());
  }

  /**
   * Inner class RandomBehaviour
   */
  private class RandomBehaviour extends Behaviour {
    private int aleatoire ;

    public void action() {
     aleatoire = (int) (Math.random()*10);
     System.out.println("aleatoire ="+ aleatoire);
    }

    public boolean done() {
      return aleatoire == 7;
    }

    public int onEnd() {
      myAgent.doDelete();
      return super.onEnd();
    }
  }
}
```

III-B - Les Behaviours planifiés

Pour planifier une tâche d'un agent JADE offre deux types de Behaviours :

III-B-1 - WakerBehaviour

Le WakerBehaviour est implémenté de façon à exécuter la méthode **onWake()** après une période passée comme argument au constructeur. Cette période est exprimée en millisecondes. Le Behaviour prend fin juste après avoir exécuté la méthode **onWake()**.

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

III-B-2 - TickerBehaviour

Le TickerBehaviour est implémenté pour qu'il exécute sa tâche périodiquement par la méthode **onTick()**. La durée de la période est passée comme argument au constructeur.

III-B-3 - Exemple

Voici un exemple d'un d'un compte à rebours.
Le premier Behaviour affiche le temps restant chaque seconde.
Le deuxième arrête le compte à rebours et termine l'agent.

```
import jade.core.Agent;
import jade.core.behaviours.WakerBehaviour;
import jade.core.behaviours.TickerBehaviour;
public class CompteaRebours extends Agent {

  protected void setup() {



final int nombreDeSecondes = (int)(Math.random()*15);
System.out.println("compteur a rebours de "+nombreDeSecondes);

//ce Behaviour montre le temps restant
    addBehaviour(new TickerBehaviour(this, 1000) {
      protected void onTick() {
        System.out.println("Il reste "+(nombreDeSecondes-getTickCount())+" seconds");
      }
    });

    //ce Behaviour va arreter le compte et terminer l'agent
    addBehaviour(new WakerBehaviour(this, 1000*nombreDeSecondes) {
      protected void handleElapsedTimeout() {
        System.out.println("Terminé");
        myAgent.doDelete();
      }
    });
  }
}
```

III-C - Les Behaviours composés

Vous allez rapidement vous rendre compte que les Behaviours présentés jusqu'ici ne peuvent pas répondre d'une manière efficace à tous les besoin d'un développeur d'un système multi-agents. Pour cela JADE offre aussi un ensemble de Behaviours composés qui servent à présenter des tâches complexes. La classe mère de toutes les autres complexes est la classe **jade.core.behaviours.CompositeBehaviour** .
Une instance de cette classe, un Behaviour bien entendu, contient des sous-Behaviours, la méthode **action()** est déjà implémentée et invoque à chaque fois la méthode **action()** de l'un de ses sous-Behaviour. L'ordre d'exécution des Behaviours est à la charge des deux méthodes **scheduleFirst()** et **scheduleNext()** que les classes héritant de la classe CompositeBehaviour doivent implémenter. Le programmeur n'est pas dans l'obligation d'utiliser directement la classe CompositeBehaviour car il on dispose de trois classes filles.

III-C-1 - SequentielBehaviour

La logique suivie dans les SequentielBehaviour est simple et intuitive. Le Behaviour commence par exécuter le premier sous-Behaviour et lorsque celui-là termine son exécution (sa méthode done() retourne true), il passe au prochain Behaviour, et ainsi de suite.
Les sous-Behaviours sont ajoutés au sequentielBehaviour par la méthode addSubBehaviour(). L'ordre de l'ajout détermine l'ordre d'exécution.

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Voici un exemple :

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.SequentialBehaviour;
import jade.core.behaviours.WakerBehaviour;


public class Seq extends Agent {
 protected void setup(){
  SequentialBehaviour comportementSequentiel = new SequentialBehaviour();

  comportementSequentiel.addSubBehaviour(new OneShotBehaviour(){
   @Override
   public void action() {
    System.out.println("le premier sous-comportement");
   }
  });

  comportementSequentiel.addSubBehaviour(new OneShotBehaviour(){
   @Override
   public void action() {
    System.out.println("le second sous-comportement");
   }
  });

  comportementSequentiel.addSubBehaviour(new OneShotBehaviour(){
   @Override
   public void action() {
    System.out.println("le derniers sous-comportement");
    myAgent.doDelete();
   }
  });
  addBehaviour(comportementSequentiel);
 }
}
```

III-C-2 - FSMBehaviour

FSMBehaviour est une sorte de Behaviour qui implémente un automate à états finis dont chaque état correspond à l'exécution d'un sous-Behaviour.
L'introduction de l'automate se fait de la manière suivante :

- L'ajout d'un nouveau état se fait par la méthode registerState (Behaviour state, String name) ;
- L'ajout de l'état initial (il n'existe qu'un seule état initial) se fait par la méthode registerFirstState (Behaviour state, String name);
- L'ajout d'un état final (il est possible d'en avoir plusieurs) se fait par le méthode registerLastState (Behaviour state, String name).

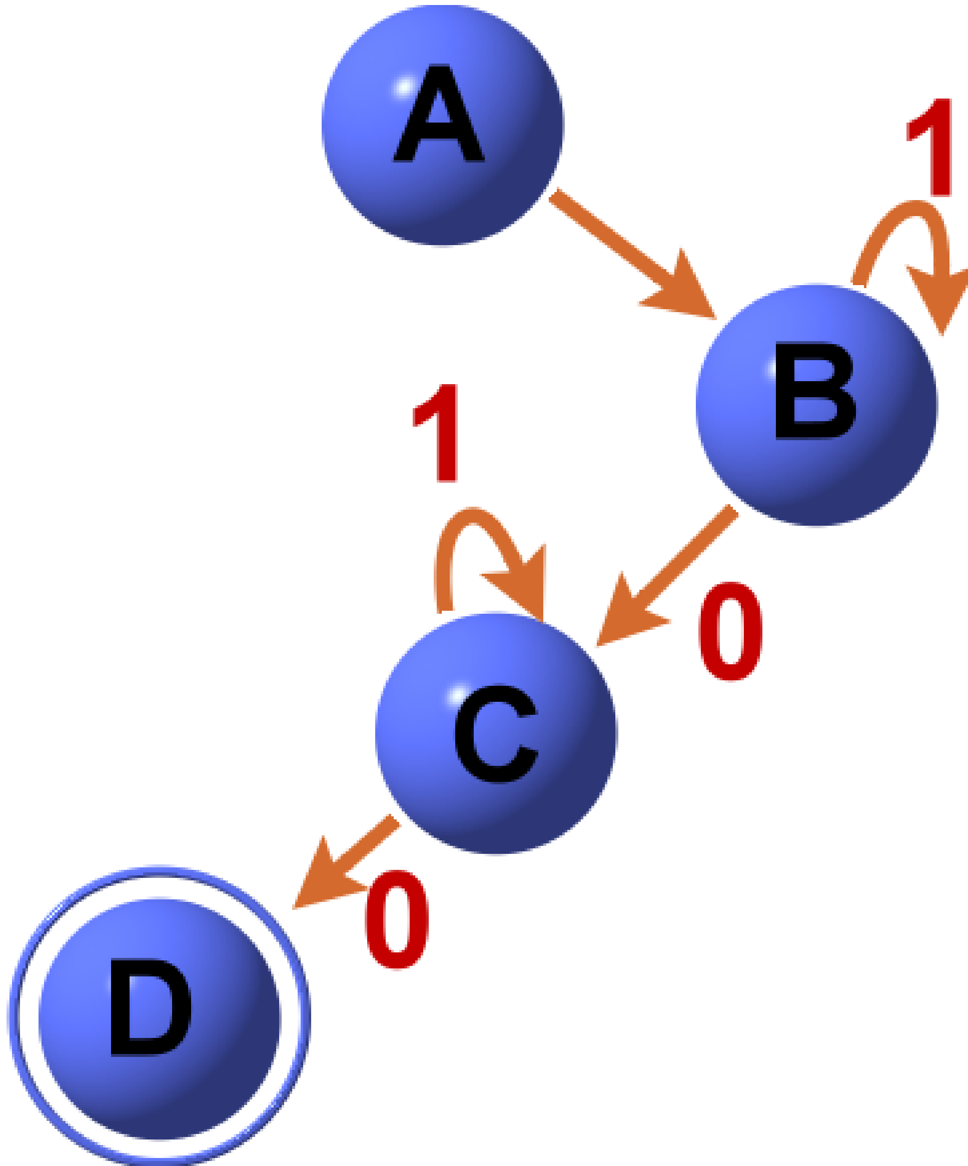
a State : le Behaviour qui représente l'état;
b name :le nom de l'état.

- L'ajout d'une nouvelle transition se fait par la méthode registerTransition(String s1, String s2, int event);
- L'ajout d'une transition par défaut (la seule transition entre deux états ou bien la transition à prendre si aucune autre n'est prise) se fait par la méthode registerDefaultTransition(String s1, String s2, String[] toBeReset)

a s1 : l'état source;
b s2 : l'état destination;
c event: l'étiquette de la transition;
d String[] toBeReset : l'ensemble d'état pour lesquels on doit faire un reset() avant de les ré-exécuter parcequ'on peut pas re-éxecuter un Behaviour une autre fois avant de lui faire un reset().

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

Exemple : considérons l'automate suivant :



voici le FSM Behaviour qui le représente :

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.FSMBehaviour;


public class Agent_FSM extends Agent{

 protected void setup(){
  FSMBehaviour fsm = new FSMBehaviour(this) {
   public int onEnd() {
    System.out.println("FSM behaviour terminé");
    myAgent.doDelete();
    return super.onEnd();
   }

  };
  //definiton des etats
  fsm.registerFirstState (new UnComportement(),"A");
  fsm.registerState(new UnComportement(),"B");
  fsm.registerState(new UnComportement(),"C");
```


Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
  fsm.registerLastState(new UnAutreComportement(),"D");

  //definition des transaction
  fsm.registerDefaultTransition("A","B");
  fsm.registerTransition("B","B",1);
  fsm.registerTransition("B","C",0);
  fsm.registerTransition("C","C",1);
  fsm.registerTransition("C","D",0);

  addBehaviour(fsm);
 }

 //Inner Class
 private class unComportement extends OneShotBehaviour{
  int aleatoire;

  @Override
  public void action() {
   System.out.println("execution de l'etat  "+ getBehaviourName());
   aleatoire = (int)Math.random() * 1;
  }

  public int onEnd(){
   return aleatoire;
  }
 }

 private class unAutreComprtement extends OneShotBehaviour{

  @Override
  public void action() {
   System.out.println("arrivée à l'etat finale");
  }
 }
}
```

III-C-3 - ParallelBehaviour

Le parallelBehaviour permet d'exécuter plusieurs Behaviours en parallèle. Par parallèle on comprend qu'après avoir invoqué la méthode **action()** d'un sous-Behaviour, on pointe sur le suivant sans attendre que le premier termine son exécution. L'ajout de sous-Behaviour se fait par la méthode **addSubBehaviour()** . Si on veut que le parallelBehaviour termine dès qu'un de ses sous-Behaviours termine alors on doit passer à son constructeur l'argument **WHEN_ANY**. (Pour attendre la fin de tous les sous-Behaviours on doit lui passer l'argument **WHEN_ALL**).

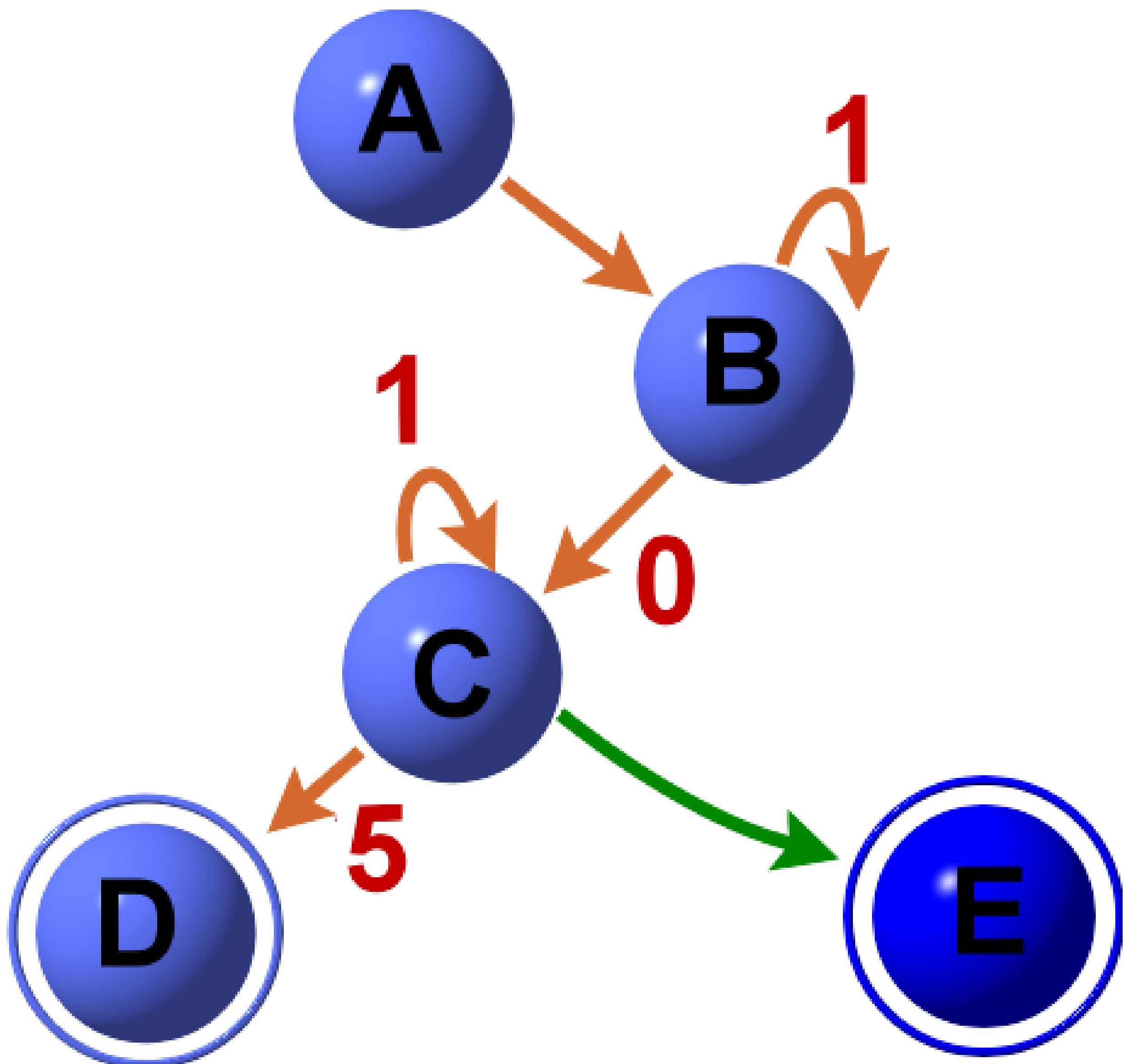
```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.ParallelBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.behaviours.WakerBehaviour;

  public class agent_parallel extends Agent{
   protected void setup(){
    ParallelBehaviour comportementparallele = new ParallelBehaviour(ParallelBehaviour.WHEN_ANY);
    comportementparallele.addSubBehaviour(new WakerBehaviour(this,1000){
     @Override
     protected void handleElapsedTimeout() {
      System.out.println("le temps est écoulé..");
      myAgent.doDelete();
     }
    });
    comportementparallele.addSubBehaviour(new TickerBehaviour(this,50){

    int aleatoire;
    @Override
    protected void onTick() {
     aleatoire=(int) (Math.random()*100);
```


Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

```
        System.out.println("aleatoire =  "+aleatoire);
        if(aleatoire==5)
        {
        System.out.println("Bingo!");
        myAgent.doDelete();
        }
      }
    });
     addBehaviour(comportementparallele);
    }
  }
```

III-D - Combiner plusieurs Behaviours composés

Il est possible de combiner les différents comportements composés pour créer des comportements plus complexes. Par exemple on peut combiner le FSMBehaviours et le parallelBehaviour précédents de la manière suivante : Attribuer à l'état C le parallelBehaviour. Si le Behaviour réussit à générer le 5 dans une second alors on passe à l'état D .Sinon on passe à l'état E



Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.

IV - Conclusion

IV-A - Bibliographie

- **Developing multi-agent systems with JADE edition wiley**.
- JADE PROGRAMMER'S GUIDE de la documentation officielle

IV-B - Remerciements

Je remercie **dourouc05** pour la relecture et la correction orthographique.

Copyright ® 2009 djug. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts. Cette page est déposée à la SACD.