Une Ontologie pour la Réutilisation des Interactions dans un Système Multi-Agents

Wafa Ketata — Wided Lejouad-Chaari

Unité de recherche SOIE, Ecole Nationale des Sciences de l'Informatique – ENSI
Campus Universitaire La Manouba 2010 Manouba – Tunis - TUNISIE

Wafa.ketata@gmail.com - Wided.chaari@ensi.rnu.tn

RÉSUMÉ. *Une des principales propriétés des Systèmes Multi-Agents (SMA) est la communication entre agents. Cette dernière représente un mécanisme capital dans une communauté d'agents. Dans ce contexte, des travaux proposent de dissocier les interactions du code des agents et de les considérer comme des ressources indépendantes de l'agent. Cette approche a été expérimentée dans une Simulation de Trafic Routier (STR). Parmi ses apports figure celui de la dynamicité des interactions entre les agents, ainsi que leur réutilisation. Dans ce papier, nous nous intéressons à structurer et à organiser ces interactions en vue de leur réutilisation. Dans ce contexte, nous proposons un modèle d'Interactions Logicielles pour les SMA, ainsi qu'une ontologie et une bibliothèque d'interactions dédiées à une STR. Enfin, et afin d'évaluer l'impact de la séparation et de la réutilisation des interactions, nous procédons à une étude des performances.*

ABSTRACT. *Agent communication is one of the most important features in Multi-Agent Systems (MAS). It is a vital mechanism in the agent community. In this regard, some works propose the disassembling of these interactions from the agent code and consider them as a set of independent resources. This approach has been experimented in a Road Traffic Simulation (RTS). Among the advantages of this approach, we note the dynamic agent interactions and their reuse. In this work, we consider the restructuring of these interactions for reuse purposes. We propose a MAS Software Interaction Model as well as an Interaction Ontology and Library for a RTS. A performance study is carried out in order to assess the impact of the disassembling and the reuse of the agent interactions.*

MOTS-CLÉS: *Système Multi-Agent, Modèle d'Interaction Logicielle, Ontologie et Bibliothèque d'Interaction Logicielle, Simulation du Trafic Routier, Evaluation de Performances.*

KEYWORDS: *Road Traffic Simulation, Software Interaction Model, Software Interaction Ontology and Library, Multi-Agent System, Performance Evaluation.*

1. Introduction

Dans un Système Multi-Agents, les agents interagissent et coopèrent pour, conjointement, réaliser une tâche ou atteindre un but commun. Dans (Ferber, 1995), *" Une interaction est la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques"*. Les interactions ne sont pas seulement la conséquence des actions effectuées par plusieurs agents en même temps, mais aussi l'élément nécessaire à la constitution de l'organisation sociale. Que ce soit dans le modèle coopératif ou compétitif, les interactions sont obligatoires. Par ce fait, dans ces deux cas, une coordination s'avère indispensable afin d'améliorer le fonctionnement global du système. Pour que les agents se comprennent, la communication doit être conforme aux notions suivantes :

- La **syntaxe** : les agents doivent parler le même langage;
- L'**ontologie** : les mêmes objets et les concepts doivent avoir la même signification pour tous les agents. les agents doivent partager une ontologie commune;
- Le **langage déclaratif** : les agents doivent être capables de dialoguer entre eux : pouvoir s'informer, se poser des questions. Pour ce faire, les agents doivent spécifier l'état désiré dans un langage déclaratif.

Généralement, dans les communications entre agents, les interactions sont prises en compte lors de la modélisation comme un comportement inclus dans le code de l'agent. Néanmoins, des travaux récents (Khalfaoui *al*, 2004), (Hanachi *et al*, 2004), (Honiden *et al*, 2005) et (Mathieu *et al*, 2006) proposent de dissocier les interactions du code des agents. Les interactions sont considérées donc comme des ressources indépendantes de l'agent. Cette approche a pour avantage notable, de favoriser la réutilisation des interactions.

Un des objectifs de cet article est ainsi, de proposer une structuration et une organisation de ces interactions en vue de leur réutilisation. Pour la structuration, nous nous basons sur un modèle d'interactions logicielles. Ce dernier constitue une référence pour la communication et l'échange d'informations entre les agents logiciels intelligents (Dieng, 2006). Pour l'organisation, nous nous servons d'une ontologie et une bibliothèque d'interactions logicielles. Le système du trafic urbain nous sert comme expérimentation afin de valider l'ontologie et la bibliothèque proposées.

Cet article est organisé comme suit : dans la section 2, nous décrivons la nature des communications entre agents. La section 3 est consacrée à l'état de l'art de l'approche centrée interactions ainsi que ses limites tout en proposant quelques solutions. Dans la section 4, nous détaillons notre nouvelle organisation des interactions. Dans la section 5, nous nous orientons plus vers l'exploitation de cette nouvelle organisation par la description d'une l'ontologie et d'une bibliothèque d'interactions logicielles. Nous montrons le principe de la réutilisation du code des interactions dans la section 6 pour la mettre en œuvre sur un simulateur de trafic routier dans la section 7. Dans cette dernière, nous procédons aussi à une étude des performances, afin d'évaluer la séparation et la

réutilisation du code des interactions. Finalement, nous présentons les principales perspectives du présent travail.

2. Communication entre agents

La communication accroît les perspectives des agents en leur agréant le bénéfice des informations et du savoir-faire des autres agents. Subséquemment, la communication des agents constitue l'un des moyens fondamentaux pour assurer la répartition des tâches et la coordination des actions entre agents. FIPA (Fipa, 1999) acteur dans le domaine des Systèmes Multi-Agents a pour principale mission de mettre au point un standard pour la communication entre agents. Un de ses aboutissements est la norme FIPA-ACL (Chaib *et al*, 2002). Il en existe une autre celle de KQML (Finin *et al*, 1994).

Un **Langage de Communication Agent** (ACL) doit être conçu pour l'échange entre agents d'informations, de connaissances ou de services. Le format utilisé pour l'échange des connaissances est fourni par un **langage de contenu**, indépendant du langage ACL (par exemple: KIF, FIPA-SL, Prolog, Clips). Le vocabulaire commun concerne les définitions précisées dans une **ontologie** (voir Figure 1).

Figure 1. *Modèle des Langages de Communication entre Agents*

FIPA-ACL propose un système standard d'échange de messages entre agents, la sémantique de ces messages est similaire à celle du langage KQML. Nous exposons, au sein de cette section, la structure d'un message au format ACL :

([**performatif**, exemple REQUEST]

:**sender** [expéditeur, exemple agent-identifier :name Prince]

:**receiver** [destinataire, exemple agent-identifier :name Aviateur]

:**content** [contenu, exemple "((action(agent-identifier :name StExupery) (draw :object cercle)))"]

:**ontology** [ontologie du domaine, exemple PetitPrince]

:**language** [langage dans lequel est exprimé le contenu, exemple FIPA-SL]

:**reply-with** [code d'identification du message, exemple dessin])

Dans l'exemple ci-dessus, le Prince demande à l'Aviateur de dessiner un cercle. Ce message a été émis avec la performative *Request*, la structure standard du message apparaît clairement. Elle se compose des champs expéditeur (*:sender*), destinataire (*:receiver*), contexte (*:ontology*), message (*:content*) et d'un certain nombre de champs métalinguistiques, en particulier le champ (*:language*) qui spécifie le langage du contenu du message. Le champ (*:ontology*) sert quant à lui, à préciser l'ontologie décrivant la structure de la conversation entre agents.

Dans le paragraphe suivant, nous présenterons brièvement des travaux portant sur une conception différente de la communication.

3. Approche centrée Interactions

Généralement, les interactions entre agents, prises en compte lors de la modélisation, finissent par être codées comme un comportement d'agent. Cependant, des travaux comme ceux de (Khalfaoui *et al*, 2004), (Hanachi *et al*, 2004), (Honiden *et al*, 2005) et (Mathieu *et al*, 2006) cherchent au contraire à leur donner un poids opérationnel égal aux entités et aux activités auxquelles elles prennent part.

(Khalfaoui *et al*, 2004) et (Khalfaoui *et al*, 2005) proposent ainsi de séparer les interactions du code métier des agents et de déléguer leur gestion à un serveur d'interactions. En fait, cette approche introduit une nouvelle architecture interne d'agent pour la modélisation des Systèmes Multi-Agents (Jouvin *et al*, 2003) puisque les interactions, comme indique la Figure 2, sont dissociées de cette architecture.

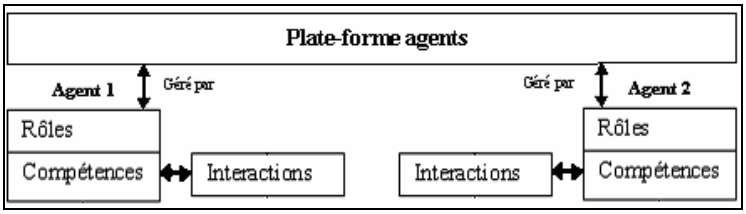

Figure 2. *Nouvelle architecture agent (Khalfaoui et al, 2004)*

Le principal apport de cette approche est la possibilité d'ajouter, de modifier et de supprimer une interaction en cours d'exécution et surtout de la réutiliser.

Dans la suite, nous exposons une mise en œuvre de cette approche.

3.1. *Mise en œuvre de l'approche centrée interactions*

Notre travail se place dans la continuité de celui de (Khalfaoui et al, 2005) qui est basé sur l'utilisation des interactions entre composants logiciels pour les Systèmes Multi-Agents. Dans l'approche centrée interactions, les agents sont gérés par la plate-forme agent. Les interactions sont dissociées du code métier des agents. Elles sont exprimées dans un langage de spécification d'interaction et sont gérées par un serveur de gestion d'interaction. La démarche de développement d'un Système Multi-Agents devient la suivante :

- La première étape est la construction des classes d'agents ainsi que leurs compétences sans se préoccuper des interactions entre elles ;
- La seconde étape consiste à développer des schémas d'interactions permettant de mettre en interaction des agents ;
- La dernière étape consiste à poser ces schémas d'interactions dans un serveur d'interactions.

Néanmoins, cette approche présente bien une limite due au principe de fonctionnement de ce modèle d'interactions conçu à l'origine pour des interactions entre composants logiciels. En effet, l'interaction entre agents est gérée par des règles. Ces dernières sont composées de deux parties : une partie gauche et une partie droite nommée ***réaction***, cette dernière se présente sous la forme d'une méthode qui représente le comportement à exécuter dans une interaction. Il est à noter que l'exécution de la première partie provoque le déclenchement immédiat de la seconde, ce qui rend l'interaction synchrone. Or, les interactions entre agents sont qualifiées essentiellement d'**asynchrones**.

Pour remédier à cette lacune, nous proposons deux solutions qui seront exposées dans les deux sections suivantes.

3.2. *Utilisation des Services*

Comme première solution, nous avons entrepris l'intégration des services, les services étant en effet des fonctions susceptibles de retourner une valeur à l'issue de leurs exécutions. L'appel de ces services est asynchrone, dans la mesure où la réception d'une requête n'est pas nécessairement suivie immédiatement par l'envoi de la réponse.

Pour la mise en œuvre, nous avons introduit deux classes. La première classe est «S*ervice*», elle caractérise un service. La deuxième classe est «*FileService*», elle a le profil d'une file de services de type FIFO (First In First Out). Au niveau des interactions, la méthode *réaction* doit enfiler un service dans la «*FileService*». Les services seront traités par la suite en mode asynchrone.

3.3. *Utilisation des Threads*

Comme seconde solution, nous proposons l'intégration des *Threads* dans le traitement des interactions. En fait, l'invocation d'une méthode n'est pas souvent très adaptée à la communication asynchrone, en particulier, lors d'un appel d'une méthode qui retourne un résultat. Toutefois, l'appel d'un *Thread* force l'attente de la disponibilité de ce résultat. Pour la mise en œuvre, la méthode *réaction* se présente de la manière suivante :

```
Public …. Methode_reaction(){
new Thread() {
    public void run() {
     /*comportement relatif à la réaction (le comportement de la partie droite de
       l'interaction)*/
    }}.start(); }
```

4. Modèle d'Interactions Logicielles pour les Systèmes Multi-Agents

La séparation des interactions des agents doit conserver la richesse de leur sémantique. Pour ce faire, nous proposons un méta-modèle pour la définition des interactions logicielles. Une interaction se produit entre **participants** de type **agent**. Une **interaction logicielle** est constituée de **schémas d'interactions**. Ces dernières se présentent sous la forme de **règles**. Ce modèle est inspiré du modèle d'interactions entre composants logiciels (Blay *et al*, 2002).

Une règle décrit comment le comportement d'un agent doit être modifié quand il interagit avec les autres. Une règle est constituée d'une partie gauche et une partie droite :

- La première nommée **message notifiant** décrit la réception d'un message ;
- La seconde nommée **réaction** décrit le nouveau comportement réactif à exécuter.

La sémantique de l'exécution d'un comportement réactif peut être spécifié à l'aide d'un ensemble d'**opérateurs réactifs** (Par exemple : opérateur séquentiel, opérateur *ifThen* ou *ifThenElse*). Dans certaines situations, le comportement réactif ne s'exécute que si des **conditions** sont vérifiées.

Grâce aux schémas d'interactions, il est possible d'exprimer à un haut niveau d'abstraction la sémantique de la communication. En particulier, la notion de règle d'interaction permet de modéliser les modifications de comportement et d'échanger des **concepts** entre les agents interagissants (participants). Les échanges de concepts se réalisent entre les **méthodes** : une méthode **retourne** le concept à transférer et une autre le prend comme un de ses **paramètres**. La Figure 3 représente le modèle d'Interactions Logicielles pour les Systèmes Multi-Agents (MILSMA).

Comme exemple applicatif, nous nous sommes basés sur une simulation de trafic routier. Selon *(*Nicolas *et al, 2005),* les Systèmes Multi-Agents semblent être particulièrement adaptés à la mobilité. Ainsi, une simulation de trafic qui utilise les SMA a l'avantage de donner une vision très fine de déplacements des individus. De plus, la philosophie des SMA ressemble bien à celle du phénomène de trafic routier. Parmi les simulateurs de trafic utilisant les SMA, nous citons «TraffSim » *(*Alexandru *et al, 2005)* et « MIRO » *(*Nicolas *et al, 2002).*

Afin de valider ce méta-modèle, nous l'avons projeté sur le Domaine du Trafic Urbain. Le résultat de cette projection est une Ontologie d'Interaction Logicielle pour le Système de Trafic Routier. Cette dernière sera détaillée dans la prochaine section.

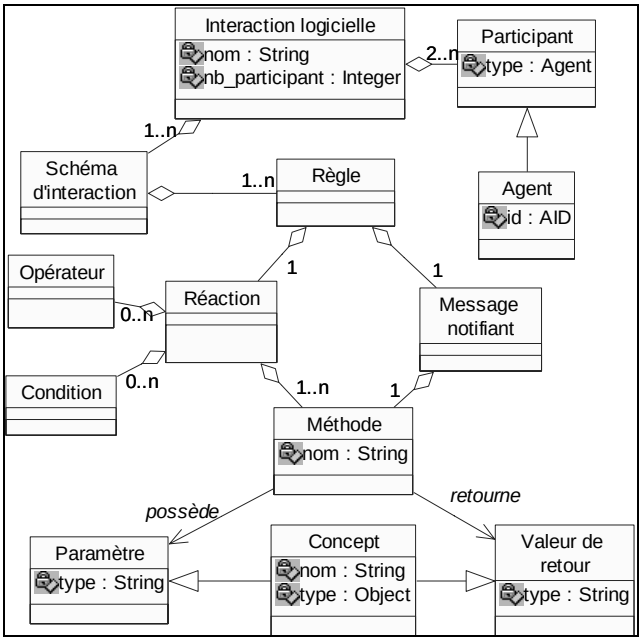

Figure 3. *Représentation conceptuelle des interactions logicielles pour les Systèmes Multi-Agents.*

5. Ontologie et Bibliothèque d'Interaction dédiées à une Simulation de Trafic Routier

L'ontologie d'Interactions Logicielles pour le Système de Trafic Routier est un ensemble de concepts et d'interactions suffisamment généraux pour pouvoir être réutilisées dans d'autres simulateurs de trafic routier :

– Les concepts représentent les données et les informations qui peuvent être transférées entre les agents. Par exemple, le concept « Carte » représente la description d'un réseau routier, ce concept est nécessaire pour un conducteur afin de déterminer ses déplacements. Comme deuxième exemple, le concept « Poursuivant » renferme les coordonnées et les propriétés d'un conducteur poursuivi, ce concept est transféré entre les conducteurs pour éviter les collisions ;
– Les interactions constituent un acte de communication entre les agents du système de trafic routier. Les conducteurs se basent sur ces interactions pour se déplacer en sécurité. Nous avons défini la majorité des schémas d'interaction nécessaires pour notre simulation. Ces schémas sont conformes à MILSMA.

Pour favoriser l'utilisation de ces interactions, nous les avons classifiées selon deux types (voir Figure 4).

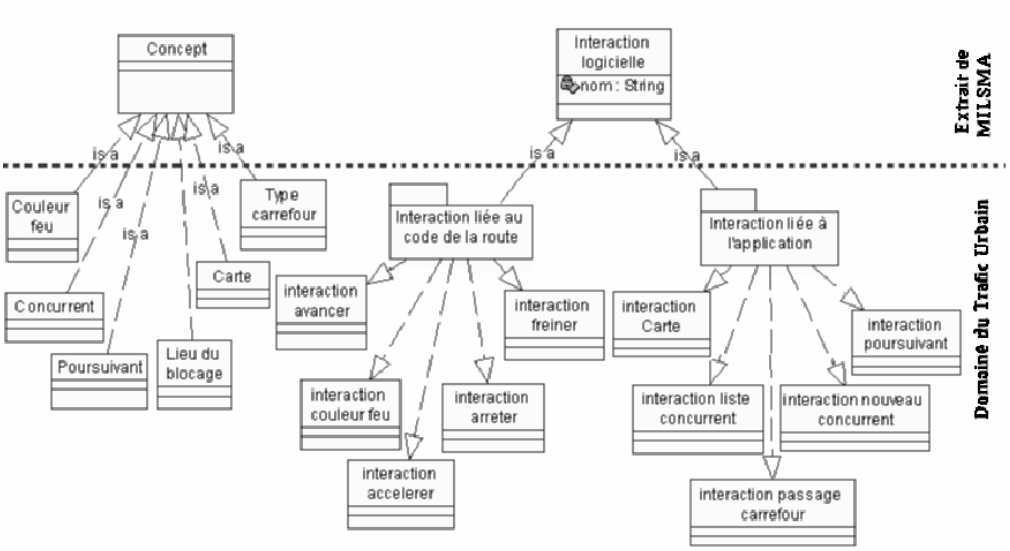

Figure 4. *Extrait de l'Ontologie d'Interactions Logicielles pour le Système de Trafic Routier*

Le premier type concerne les interactions liées au code de la route. Le code de la route étant en effet un ensemble de lois, il définit généralement le comportement du conducteur pour chaque situation spécifiée. Par exemple, un conducteur se sert de l'interaction «*CouleurFeu*» pour se renseigner sur les signaux du feu, le schéma d'interaction correspondant est comme suit :

```
Interaction CouleurFeu (conducteur c, feu f) {
  c.couleurFeu() → c._call ; c.setCouleurFeu(f.getCouleurFeu())   }
```

Puis, comme deuxième type, nous avons souligné des interactions liées à l'application qui sont utilisées pour le fonctionnement du simulateur. Par exemple, l'environnement se sert de l'interaction «*Carte*» pour renseigner un conducteur sur le réseau routier, le schéma d'interaction correspondant est comme suit :

```
Interaction Carte (environnement e, conducteur c) {
  e.donnerCarte() → e._call ; c.setCarte(e.getCarte())   }
```

L'Ontologie d'Interaction Logicielle pour le Système de Trafic Routier a donné naissance à une Bibliothèque d'Interaction Logicielle dédiée à une Simulation de Trafic Routier. Cette bibliothèque comprend les interactions les plus fréquemment utilisées dans le Domaine du Trafic Urbain, à savoir, «*Avancer*», «*Arreter*», «*Freiner*», «*PassageCarrefour*» et «*Poursuivant*». Nous envisageons de l'enrichir dans des versions futures.

Le principal intérêt d'avoir une bibliothèque d'interactions est de rendre possible leur réutilisation dans tout simulateur de trafic routier. Un second apport de point de vue génie logiciel est que les interactions sont structurées, explicites et manipulables. Pour une meilleure utilisation de cette bibliothèque d'interactions, nous avons conçu une API (Application Programming Interface) de gestion d'interactions. Cette API permet aux agents de la simulation d'instancier des interactions.

Nous avons l'intention, dans l'avenir, de rendre la bibliothèque d'interaction et le simulateur développé accessible au public.

Dans le paragraphe suivant, nous détaillons le principe de la réutilisation du code des interactions.

6. Principe de la réutilisation du code des interactions

Dans le but d'orienter l'agent pour retrouver le nom de l'interaction dont il a besoin et de l'utiliser, nous nous sommes servis d'une ontologie nommée « guide d'utilisation des interactions logicielles pour une simulation de trafic routier ». Cette ontologie est orientée « thesaurus », elle permet de définir des correspondances terminologiques afin de guider la recherche d'une interaction. En effet, l'objectif de cette ontologie est d'assurer la correspondance entre le besoin d'un agent et le nom de l'interaction logicielle en question. Pour clarifier encore, prenons la série d'actions suivante : ralentir, freiner, diminuer, modérer, réduire. Ces actions sont synonymes et sont équivalentes à l'interaction logicielle « *freiner* » (voir Figure 5).

Grâce à cette ontologie, l'agent est capable de retrouver le nom de l'interaction dont il a besoin. Par conséquent, la réutilisation des interactions se réalise en quatre phases (voir Figure 6) :

- La première phase est la sélection de l'interaction requise à travers l'ontologie « guide d'utilisation des interactions pour une simulation de trafic routier » ;

- La deuxième phase est l'instanciation du code de l'interaction en question à partir de l'API de gestion des interactions et de la bibliothèque des interactions ;
- La troisième phase est la sauvegarde de cette instance dans un serveur d'interactions tout en spécifiant les agents interagissants ;
- La dernière est la gestion de ces interactions par le serveur d'interactions en liant et déliant les agents interagissants.

Afin de clarifier encore le principe de la réutilisation du code des interactions, prenons l'exemple d'un agent policier qui interagit avec un conducteur pour l'ordonner à avancer. D'abord et à travers le « guide d'utilisation des interactions », l'agent policier retrouve l'identification de l'interaction requise, dans cet exemple, l'interaction en question est «*Avancer*». Ensuite, l'agent policier fait intervenir l'API de gestion d'interactions, afin de fournir une instance de l'interaction en question. Cette instance est sauvegardée dans le serveur responsable de lier et délier les agents policier et conducteur.

Dans la suite, nous évaluons la séparation et la réutilisation des interactions en procédant à une étude des performances.

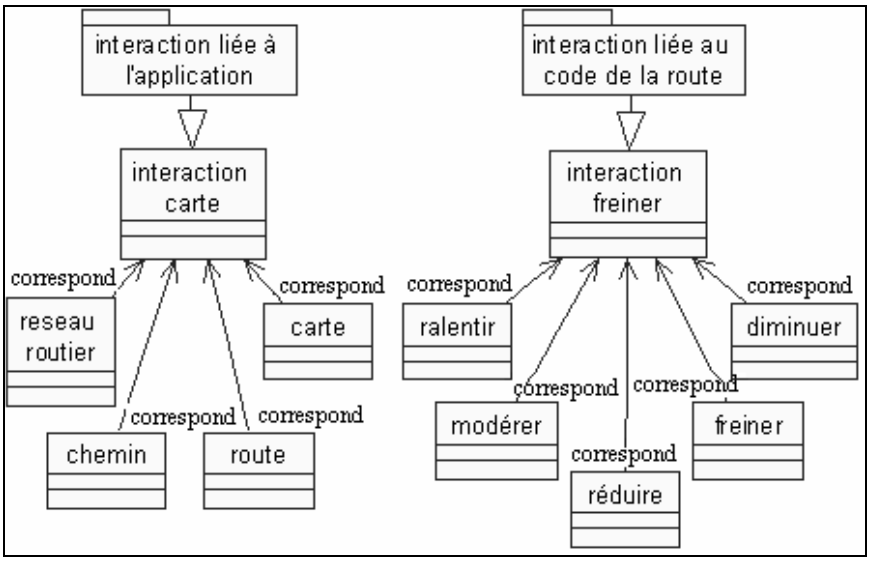

Figure 5. *Extrait de l'ontologie « guide d'utilisation des interactions logicielles pour une STR»*

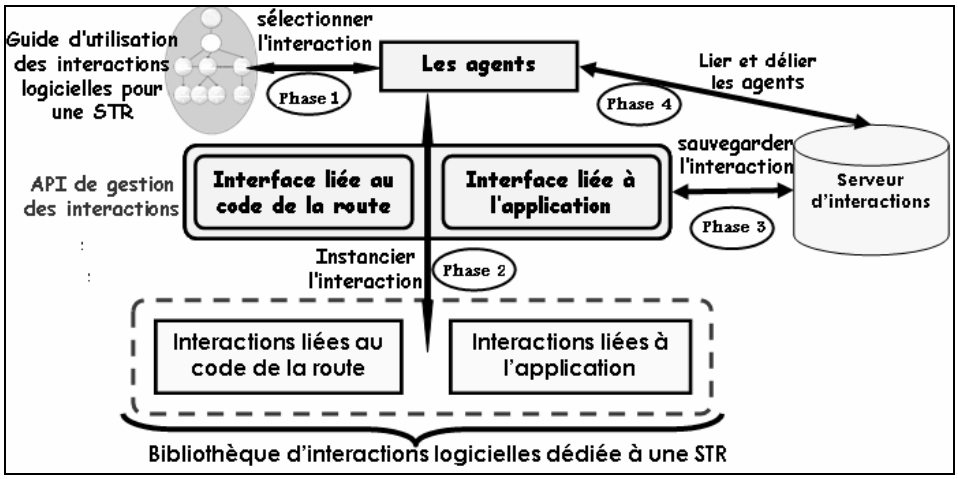

Figure 6. *Principe de réutilisation du code des interactions*

7. Etude comparative et validation

Dans la présente section, nous commençons par décrire les simulations développées. Ensuite, nous exposons une étude comparative de leurs performances. Nous terminons par une discussion. Nous désignons par AA une application agent classique et par AS et AT des applications centrées interactions (AS simulant la solution avec les Services, AT celle avec les *Threads*) :

- Une simulation de trafic routier selon une approche classique a été implémentée sous la plate-forme agents JADE (URL2). Les interactions entre agents se manifestent par l'échange de messages ACL. JADE fournit trois façons pour la mise en place de la communication entre agents. Nous avons opté pour celle introduisant la définition de l'ontologie. Pour implémenter cette ontologie, nous avons eu recours à l'éditeur d'ontologie protégé2000 (URL3). Un des avantages de cet éditeur est de pouvoir générer automatiquement des fichiers Java à travers le plug-in BeanGenerator (URL4), ces fichiers étant reconnus par la plate-forme JADE (Chris *et al*, 2002).
- Une simulation de trafic routier centrée interactions asynchrones où les agents sont gérés par la plate-forme JADE. Les interactions sont dissociées du code métier des agents. Elles sont conformes au modèle d'interaction MILSMA proposé et sont exprimées dans le langage d'interactions ISL et gérées par le serveur d'interactions NOAH (URL1). Ces simulateurs se basent sur l'ontologie et la bibliothèque décrites précédemment.

Nos applications font intervenir des agents de types : conducteur, piéton, environnement, carrefour, voie, policier et feu. Afin de comparer ces applications, nous avons choisi d'évaluer leurs performances. Pour ce faire, nous avons identifié six situations (a, b, c, d, e, f)1, pour lesquelles nous avons extrait l'usage de la CPU à partir de l'outil d'administration des performances fourni par Microsoft. Ces situations sont décrites dans le Tableau 1. A titre d'exemple, la situation « e » est constituée d'un environnement, de trois carrefours, de vingt voies, d'un feu et de dix conducteurs.

Le modèle comparatif se présente comme suit : d'abord, nous exposons quelques courbes de performances pour ensuite commenter et interpréter les résultats obtenus. Nous dressons après un tableau récapitulatif des temps qu'a pris chaque exécution, il faudra souligner que les valeurs exposées sont des moyennes résultantes de plusieurs exécutions, l'unité de temps étant la seconde. Ce tableau présente aussi un ratio de gain calculé à l'aide de la formule [1].

Gain_temps_d'exécution = ((temps_1–temps_2) / (temps_1)) * 100 **[1]**

Situations	**a**	**b**	**c**	**d**	**e**	**f**
Environnement	1	1	1	1	1	1
Carrefour	1	1	2	2	3	4
Voie	8	8	14	14	20	26
Conducteur	4	4	7	8	10	10
Policier	0	1	0	1	0	0
Feu	1	0	1	1	1	1
Total	**15**	**15**	**25**	**27**	**35**	**42**

Tableau 1. *Description des situations d'évaluation*

7.1. *Evaluation des performances des applications AT – AS*

7.1.1. Les courbes de performances

Dans la Figure 7, et d'après l'allure de la courbe AT, l'usage de CPU ne dépassent pas les 98,5%, ce qui n'est pas le cas pour ceux de l'AS. Nous avons constaté la même chose dans les autres situations.

[1] Une carte routière créée sur laquelle des conducteurs interagissent entre eux et avec les composants qui les entourent afin d'atteindre leurs objectifs.

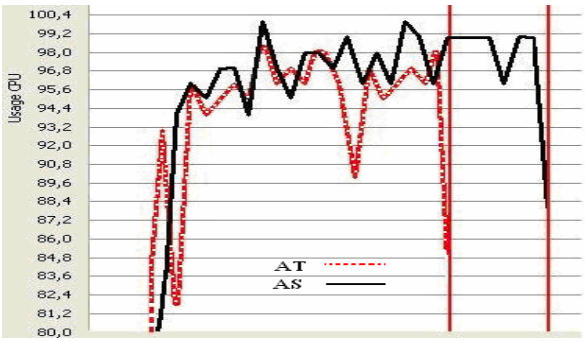

Figure 7. *L'usage CPU d'AT et d'AS (Situation a)*

7.1.2. Interprétations

D'après le Tableau 2, nous obtenons un gain en temps d'exécution d'AT par rapport à AS. Ce gain n'est pas pour autant important et il dépend de la situation. Par exemple, il est minime dans certains cas (4,2 % pour «c») et considérable dans d'autres (~22 % pour «e»). L'explication de ceci étant que chaque agent d'AS dispose d'un comportement en plus (celui qui traite les services en mode asynchrone) par rapport à AT. Plus le nombre d'agents est grand, plus le traitement de ce comportement est coûteux (situations « e » « f »). D'après cette étude, nous jugeons que l'application utilisant les Threads est légèrement plus performante que celle utilisant les services.

Simulation	**a**	**b**	**c**	**d**	**e**	**f**
AT	21	20,3	45,3	49,3	54,6	57,6
AS	26,6	25	47,3	55,6	70,3	65,6
Gain AT%AS	**21%**	**18,8%**	**4,2%**	**11,5%**	**21,9%**	**12,2%**

Tableau 2. *Tableau récapitulatif des gains (AT%AS)*

7.2. *Evaluation des performances des applications AT – AA*

7.2.1. Les courbes de performances

D'après la Figure 8, les pics de surcharge de CPU d'AA sont plus nombreux et plus intensifs que ceux d'AT. Le même constat a été établi pour les autres situations.

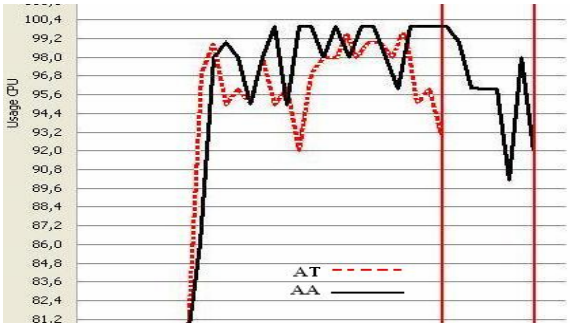

Figure 8. *L'usage CPU d'AT et d'AA (Situation a)*

7.2.2. Interprétations

D'après le Tableau 3, nous déduisons un gain de temps d'AT par rapport à AA. En outre, nous constatons que ce gain dépend de la situation, il est moyen pour certains cas (25,61 % pour « a ») et assez considérable pour d'autres (~52 % pour « d »). Le gain est plus important lorsque les situations sont plus complexes. En fait, la plate forme JADE ne peut pas supporter un grand nombre d'agents. D'ailleurs, à partir de 35 agents, AA devient très lente, plusieurs collisions inattendues ont lieu et les déplacements des conducteurs deviennent anormaux. Pour cette raison, les courbes de performances renferment des pics de surcharge de CPU. De plus, avec JADE, chaque transfert d'information se manifeste par l'envoi et la réception d'un message ACL. Or avec l'approche centrée interactions, il est possible d'intégrer des transferts d'informations dans une même interaction. Donc, nous réduisons ainsi le nombre d'interactions dans AT. Pour cela, AT met toujours un temps inférieur par rapport à celui mis par AA.

Simulation	**a**	**b**	**c**	**d**	**e**	**f**
AA	28,3	27,6	56,3	102,3	86,6	80
AT	21	20,3	45,3	49,3	54,6	57,6
Gain AT%AA	**25,6%**	**26,4%**	**19,6%**	**51,5%**	**36,9%**	**28%**

Tableau 3. *Tableau récapitulatif des gains (AT%AA)*

D'après cette étude, nous jugeons que l'application centrée interactions est plus performante que l'application classique.

7.3. *Discussion*

L'approche centrée interactions offre la dynamicité des interactions, ainsi que la facilité de leur réutilisation. D'après la section précédente, nous avons montré qu'il y a une amélioration des performances dans une application centrée interactions. Toutefois, nous avons rencontré des difficultés qui méritent d'être présentées.

Dans un scénario de trafic routier, le nombre d'agents interagissants est variable. Alors que dans notre application, ce nombre est fixe dans le paramétrage des schémas d'interactions. Pour contourner ce problème, nous proposons deux solutions. La première consiste à raisonner avec des interactions binaires. Cette solution amplifie le nombre d'interactions, ce qui n'est pas souhaitable. La deuxième solution consiste à générer instantanément les interactions selon le besoin. Le traitement de cette solution est coûteux en temps d'exécution. Une solution plus adéquate sera étudiée dans l'avenir.

8. Conclusion

Dans cet article, nous avons présenté la communication dans un Système Multi-Agents, ainsi que les deux techniques de communication entre agents : celle de la norme FIPA-ACL, et celle proposée dans les travaux de (*Khalfaoui et al*, 2005). Cette dernière consiste à dissocier les interactions de l'architecture interne de l'agent. Nous avons ainsi proposé une structuration et une organisation de ces interactions en vue de leur réutilisation. Ceci est désormais possible grâce à la définition d'un méta-modèle d'interaction logicielle, et la construction d'une ontologie et d'une bibliothèque d'interactions dédiées à une simulation de trafic routier. Dans ce papier, nous avons présenté le principe de la réutilisation du code des interactions. Et afin d'évaluer leur séparation et leur réutilisation, nous avons effectué une étude des performances d'une application centrée interactions et nous avons dégagé les gains obtenus par rapport à l'approche classique.

Comme perspectives, nous souhaitons appliquer l'approche centrée interactions à des protocoles d'interactions plus élaborés et plus complexes. Nous espérons tester les simulateurs développés sur des données réelles, par exemple sur une ville donnée. De même, nous pensons qu'une meilleure intégration de l'approche centrée interaction au niveau des plates-formes agents pourrait faire l'objet de travaux futurs.

9. Bibliographie

Alexandru Cicortas, Nosbert Somsi, « Multi-Agent System Model for Urban Traffic Simulation », 2005.

Blay-Fornarino M., Emsellem D., Occello A., Pinna-Dery A-M., Riveill M., Fierstone J., Nano O., Chabert G., « Un service d'interactions: principes et implémentation », *Journées Composants*, 18-19 octobre, Grenoble -France, 2002.

Chaib-Draa B. and F.Dignum, « Trends in Agent Communication Language», Journal: Computational Intelligence, 2002.

Chris Var. Aart, G. Caire, Ruurd Pels, Federico Bergenti, « Creating and Using Ontologies in Agent Communication », *Workshop Ontologies in Agents System 1 international CAASMA*, 2002.

Dieng-Kuntz R. « Ontologies for Knowledge Management Interoperability Research School », *International Research School: on "Ontologies a smart way towards interoperability?*" Paris, 2006.

Ferber Jacques, « *Les systèmes multi-agents »,* InterEditions, pages 67, 1995.

Finin, T., Fritzson, R., McKay, D., McEntire, R. « KQML as an Agent Communication Language ». *Proceedings of the 3rd International CIKM*, pages 456–463, USA. ACM Press, 1994.

Fipa: Foundation for Intelligent Physical Agents, « Agent Communication Language », *FIPA 99 Specification Draft*, 1999.

Hanachi C., Sibertin-Blanc C., « Protocol moderators as active middle-agent in Multi-agent System », *In Autonomous Agents and Multi-Agent System*, Vol8, page 131-164, 2004.

Honiden S., Doi T., Tahara Y., « An interaction description language for multi-agent systems »*, In Actes of 4rd A International joint Conference on Autonomous Agents and Multi-Agent System*, Utrecht, the Netherlands, 2005.

Jouvin D., S. Hassas, « Architectures dynamiques de systèmes multi-agents conversationnels», *Journées Francophones des Systèmes Multi-Agents : JFSMA*, 2003.

Mathieu Ph., S. Picault, « Vers une représentation des comportements centrée interactions», *Acte des 15éme Rencontres Francophones Reconnaissance des Formes et Intelligence Artificielle (RFIA 06*), 2006.

Nicolas Marilleau, Lang Christophe, Arnaud Banos, Sonia Chardonnel, Thomas Thévenin, « Une approche multi-agents de la ville en mouvement », 2002.

Nicolas Marilleau, Christophe Lang, Pascal Chatonnay, Laurent Philippe, « Un méta-modèle à base d'agents pour modéliser la mobilité urbaine »*. 3 éme conférence international : SETIT, Tunisie, 2005*.

Kalfaoui Sami, W. Lejouad-Chaari, A.-M Pinna-Dery. « Interactions entre composants pour environnements multi-agents », *Journée Multi-Agents et composants: JMAC*, 2004, Paris.

Kalfaoui Sami, W. Lejouad-Chaari, « Interactions entre composants pour des SMA dynamiques et évolutifs : application à une simulation de trafic routier », *MHOSI*, Tunisie, 2005.

(URL1) http://rainbow.essi.fr, ISL et NOAH, Novembre 2005.

(URL2) www.sharon.cselt.it/projetcs/jade, JADE, Octobre 2005.

(URL3) www.protege.stanford.edu, protege2000, Mars 2006.

(URL4) www.protege.cim3.net/cgibin/OntologyBeanGenerator, BeanGenerator, Mars 2006.