Collaboration dynamique dans un SMA

Bruno Mermet
Bruno.Mermet@univ-lehavre.fr

LIH
Université du Havre,
76058 Le Havre Cedex, France

Résumé :
Dans cet article, nous proposons un modèle d'interaction entre agents permettant non seulement à un agent de résoudre des buts qui n'ont pas forcément été prévus lorsqu'il a été développé, mais autorisant aussi plusieurs agents développés indépendamment, donc ignorant totalement leurs comportements respectifs, à apprendre à se connaître afin d'éventuellement collaborer. Pour ce faire, dans une première partie, nous décrivons ce que nous entendons par *but* et *résoudre un but* pour une agent. Dans une deuxième partie, nous expliquons les principes de notre architecture et nous expliquons intuitivement pourquoi une optimisation est nécessaire à sa mise en oeuvre. Afin de prouver la validité de cette optimisation, nous nous appuyons sur une méthode formelle, la méthode B, décrite dans la quatrième partie. Enfin, dans la cinquième et dernière partie, nous appliquons la méthode B à notre problème et prouvons ainsi la validité de notre système.

Mots-clés : système multiagent, collaboration, méthode formelle, proactivité, adaptativité

Abstract:
In this article, a model of interaction between agents is presented. This model gives to agents two essential dynamic characteristics. Firstly, it allows to agents to solve goals they discover at runtime and for which they were not written. Secondly, il allows to agents developped independently to collaborate to solve common goals. Moreover the correcteness of an optimized architecture for such a model is proved.

Keywords: multiagent system, collaboration, formal method, proactivity, adaptativity

1 Introduction

Dans un système multi-agent constitué d'agents *proactifs*, ceux-ci ont un comportement essentiellement guidé par un *but*. Un tel système n'est à proprement parlé proactif que dans la mesure où un agent est capable de changer de but au besoin. Dans le cas général, un agent *adaptatif* devra être en mesure d'essayer de résoudre tout nouveau but qu'il se fixe, même dans le cas d'un but inconnu du concepteur du système. Cela nécessite qu'un agent soit capable d'évaluer le rôle des différentes actions qu'il peut effectuer vis à vis de la progression vers son but courant. Une façon de résoudre ce problème est exposée dans la partie 2 de l'article. Dans les parties suivantes nous étendons le problème à l'ensemble d'un système multi-agents dans lequel les agents peuvent collaborer pour résoudre certains buts. Il faut alors que ceux-ci soient capables d'échanger leurs besoins et leurs moyens d'action. Le processus présenté en partie 3 étant relativement compliqué, nous présentons dans la partie 4 une méthode formelle qui nous permet de vérifier (en partie 5) l'optimisation envisagée.

2 Résolution d'un but

2.1 But et variant

Nous supposons que nous avons des agents dont le comportement est guidé par la résolution d'un but. Que ce but soit unique, choisi dans une liste, ou bien qu'il provienne d'un besoin calculé au cours de l'exécution, n'a pas d'importance ici. Par contre, un but, pour nous, doit être caractérisé par deux choses :

– une définition (formelle ou informelle) avec un nom ;
– un *variant*.

Comme la notion de variant est fondamentale dans tout le reste de l'article, revenons sur sa définition.

définition : un *variant* est une suite strictement décroissante définie sur un ensemble bien fondé.

En résumé, lorsqu'un agent essaie de résoudre un but, il faut que chacune des actions qu'il entreprend fasse décroître le variant[11] associé au but en question. Ce variant doit être calculable par l'agent ce qui implique qu'il ne peut être fonction que :

– des caractéristiques propres de l'agent ;
– de certaines caractéristiques *visibles* des agents faisant partie de son réseau d'accointances ;
– de l'environnement qu'il perçoit dans le cas d'un agent situé dans un environnement.

2.2 La résolution d'un but à proprement parler

A priori, un agent sait effectuer un certain nombre d'actions. Pour résoudre un but, il doit choisir parmi ces actions une action qui fait décroître le variant associé au but. Deux possibilités s'offrent alors au concepteur :

- soit la liste des buts que l'agent peut avoir à résoudre est connue dès le départ, auquel cas il suffit de stocker directement dans le code de l'agent la liste des actions possibles pour chaque but ;
- soit le but peut être inconnu au départ. Dans un tel cas, il est alors nécessaire que l'agent découvre quelles sont ses actions qui permettent de faire décroître le variant associé au but courant.

Le deuxième point présenté ci-dessus implique deux choses sur la structure d'un système multi-agent permettant à un agent de résoudre des buts dont il prend connaissance dynamiquement :

- lorsqu'un agent découvre un nouveau but à résoudre (un but qui, par exemple, lui a été assigné par un autre agent), il faut en même temps qu'on lui fournisse une fonction d'évaluation du variant associé à ce but ;
- pour choisir quelle action effectuer, le système doit permettre à l'agent :
  - de calculer le variant associé au but courant dans la situation courante ;
  - de calculer le variant associé au but courant sur la situation obtenue après simulation de l'exécution d'une de ses actions.

3 Collaboration entre agents

3.1 Principe général

Lorsqu'un agent A doit résoudre un but, il peut demander à d'autres agents (ceux faisant partie de son réseau d'accointances) de l'aider. Ceci peut notamment avoir lieu lorsque l'agent A ne sait pas comment s'y prendre (c'est-à-dire qu'aucune de ses actions ne permet de faire décroître le variant associé au but courant).

Pour ce faire, l'agent A demande aux agents qu'il connait les actions qu'ils sont susceptibles de faire pour les aider. Les agents interrogés répondent par un sous-ensemble de leurs actions où ne figurent que les actions qu'ils sont éventuellement prêts à exécuter compte tenu de contraintes propres.

L'agent A évalue alors la simulation de l'exécution des différentes actions qui lui ont été proposées et demande alors à un certain nombre d'agents dont une action donnée peut l'aider qu'ils l'exécutent.

3.2 Mise en oeuvre

La mise en oeuvre d'un tel mécanisme n'est pas évidente car les agents n'ont qu'une vue partielle de l'environnement. Si tel n'était pas le cas, la solution théorique est relativement simple :

Soit Env l'environnement (dont les agents). Soit $E(Env)$ l'état de l'environnement. Soit B_A le but courant de l'agent A et $V = V_{B_A}(E(Env))$ la valeur du variant associé au but courant de l'agent A dans l'état courant de l'environnement Env.

Soit a_C une action proposée par un agent C acceptant de collaborer avec l'agent A. Pour savoir si l'action est intéressante, l'agent A doit alors calculer :

$$V2 = V_{B_A}(E(a_C(Env)))$$

Si V2 est inférieur à V, alors l'action proposée est intéressante et l'agent A doit vraisemblablement demander à l'agent B d'effectuer l'action a_C.

Pratiquement, cette solution n'est cependant pas réalisable car elle nécessite de copier tout l'environnement pour tester l'application de l'action a_C sur celui-ci, ce qui n'est pas forcément réalisable.

Le fait que les agents n'aient qu'une vue partielle de leur environnement, d'un côté, complique les choses, mais d'un autre, les rend plus réalisables.

Soit $\mathcal{V}_A(Env)$ et $\mathcal{V}_C(Env)$ les vues respectives des agents A et C de l'environnement dans lequel ils sont plongés.

Le variant de chaque agent n'étant défini que sur la vue qu'il a de son environnement, on a :

$$V_{B_A}(E(\mathcal{V}_A(Env))) = V_{B_A}(E(Env))$$

Et plus généralement :

$$V_{B_A}(E(\mathcal{V}_A(Env))) = V_{B_A}(E(\mathcal{V}_A(Env) \cup P(Env)))$$

où $P(Env)$ est une partie quelconque de l'environnement.

Un agent ne pouvant agir que sur les éléments appartenant à la vue qu'il a de son environnement, on a :

$$E(a_C(Env)) = E((Env - \mathcal{V}_C(Env)) \cup a_C(\mathcal{V}_C(Env))$$

Par conséquent, si on définit Env' ainsi :

$$Env' = \mathcal{V}_A(Env) \cup \mathcal{V}_C(Env)$$

Alors la valeur $V2$ peut se calculer ainsi :

$$V2 = V_{B_A}(E(a_C(Env')))$$

Afin de mieux formaliser le problème, nous utilisons une méthode de spécification formelle, la méthode B, que nous présentons dans la partie suivante.

4 La méthode B

4.1 Historique

Le développement de logiciels de plus en plus gros intervenant dans des domaines où la sécurité joue un rôle critique a sensibilisé la communauté informatique à la nécessité de faire des preuves. Les preuves directement sur les programmes semblant difficiles et intervenant tard dans la phase de conception d'un logiciel, on a rapidement vu l'émergence de méthodes formelles. Parmi celles-ci, on trouve notamment les méthodes dites *orientées modèles*. Elles se caractérisent par la définition d'opérations sous une forme *pré-post*, qui définissent des actions possibles sur des données définies par leurs types, et devant éventuellement vérifier certaines propriétés.

La première de ces méthodes fut VDM (Vienna Development Method), mise au point par Bjorner et Jones ([9]). Puis vint Z, initialement développée par Jean-Raymond Abrial ([12]), qui utilisait plus largement la théorie des ensembles. Par la suite, J-R Abrial a créé la méthode B, plus structurée et plus facile d'accès.

4.2 Fondements

La méthode B [1] partage certaines propriétés avec Z. Ainsi, elle permet l'expression de propriétés sous la forme de prédicats du premier ordre, et les opérateurs auxquels on peut faire appel sont ceux de l'arithmétique et de la théorie des ensembles.

Les données, en B, sont encapsulées au sein de *machines abstraites*, et ne peuvent être modifiées que par des opérations définies dans la même machine. Les preuves que l'on cherche à faire en B sont des preuves d'invariance. Aussi chaque machine est caractérisée par un *invariant*, qui définit l'ensemble des valeurs que ses variables peuvent prendre. Aussi toute machine engendre une *obligation de preuve* établissant la conservation de l'invariant par les opérations de la machine. Une différence notable par rapport à Z et VDM est l'utilisation de la substitution généralisée, et non plus de la logique, pour exprimer la post-condition d'une opération.

Une caractéristique particulière de la méthode B est de permettre, par le moyen de raffinements successifs *prouvés*, de passer progressivement d'une spécification abstraite (indéterministe) à une spécification déterministe automatiquement traduisible en un langage de programmation.

Un autre aspect de la méthode B est qu'elle a été conçue pour être aisément automatisée. Ainsi, la génération des obligations de preuve (de correction totale et de raffinement) obéit à des règles simples qui peuvent être facilement implantées par un logiciel.

4.3 Les outils

L'aspect automatisable de la méthode B a rapidement donné naissance à deux outils : l'Atelier B [5] et le B-Toolkit [3]. Ceux-ci implantent plus ou moins des fonctionnalités telles que la gestion de projet, la génération des obligations de preuve et leur preuve, la traduction automatique en un langage de programmation, l'animation de spécification, et la génération de documents.

L'existence de tels outils a permis une rapide utilisation de la méthode dans le milieu industriel [4], comme l'a notamment montré la première conférence B [2].

4.4 Quelques notions de B

Les variables sont définies dans la théorie des ensembles de Zermelo. Par exemple, Ainsi, pour caractériser l'ensembles des agents d'un système, on peut définir la variable $agents$ comme ceci :

$$agents \subseteq AGENTS$$

Où AGENTS représente le *type* des agents.

Pour caractériser le fait qu'à chaque instant, un agent est dans un état donné, on peut définir la variable $etat$ suivante :

$$etat \in agents \rightarrow ETATS$$

Le symbole $\rightarrow$ correspond à une fonction totale[1]. Cela signifie que tout agent du système a un et un seul état à un instant donné, mais plusieurs agents peuvent être dans le même état.

Dans la suite, nous utilisons quelques autres symboles ensemblistes que nous rappelons ici :

- $A \backslash B$ représente la différence ensembliste : $A \backslash B = \{x / x \in A \wedge x \notin B\}$.
- $f \in A \leftrightarrow B$ signifie que f est une relation entre des éléments de A et des éléments de B (un élément de A peut avoir de 0 à n images dans B par f et un élement de B peut avoir de 0 à n antécédents dans A par f.
- $A \lhd f$ représente la restriction de la fonction f aux éléments qui ne sont pas dans A : $A \lhd f = \{(x, y) / (x, y) \in f \wedge x \notin A\}$.

5 Formalisation B du système

Soit Env l'ensemble (des identifiants) des composants de l'environnement. Soit $\mathcal{O}$ l'ensemble des objets de l'environnement et $\mathcal{A}$ l'ensemble des agents de l'environnement avec :

$$Env = \mathcal{O} \cup \mathcal{A}$$
$$\mathcal{O} \cap \mathcal{A} = \emptyset$$

Soit `Etat` la fonction définissant l'état des différents composants de l'environnement :

$$Etat \in Env \rightarrow \texttt{ETATS}$$

[1] une fonction partielle est représentée par $\nrightarrow$

Soit $\mathcal{V}$ la fonction définissant la vue qu'un agent a de l'environnement ($\mathcal{P}(Env)$ désigne une partie de Env) :

$$\mathcal{V} \in \mathcal{A} \rightarrow \mathcal{P}(Env)$$

Soit a un agent du système et c un agent connu de a :

$$a \in \mathcal{A}$$
$$c \in \mathcal{A}$$
$$c \in \mathcal{V}(a)$$

Une action est une application modifiant l'état de l'environnement, c'est-à-dire les états internes d'un certain nombre de composants de l'environnement (visibles par un agent effectuant l'action). Donc Si on définit l'action d'un agent comme une opération :

Etat $\leftarrow$ action(agent, EtatAvant) =
PRE
  agent $\in$ A
$\wedge$ EtatAvant $\in$ Env $\rightarrow$ ETATS
  THEN
    ANY Etat'
    WHERE Etat' $\in$ Env $\rightarrow ETATS$
    $\wedge\ V(agent) \lhd Etat' = V(agent) \lhd EtatAvant$[2]
    THEN
      Etat := Etat'
    END
END

Soit $actions$ la fonction définissant les actions d'un agent ($ACTIONS$) désigne toutes les actions possibles ; une action est rattachée à un et un seul agent si elle est instanciée, mais un agent peut disposer de plusieurs actions) :

$$actions^{-1} \in ACTIONS \nrightarrow \mathcal{A}$$

Soit ac une action de l'agent c :

$$ac \in actions[\{c\}]$$

Soit $buts$ la fonction définissant les buts d'un agent :

$$buts \in \mathcal{A} \leftrightarrow BUTS$$

[2] l'action ne modifie pas l'état des éléments de l'environnement qu'il ne voit pas

Et soit but la variable définissant le but courant des agents :

$$\begin{cases} but \in \mathcal{A} \rightarrow BUTS \\ but \subseteq buts \end{cases}$$

Soit $variant$ la fonction définissant le variant associé à chaque but :

$$variant \in BUTS \rightarrow (Etat \nrightarrow NATURAL)$$

Pour tout but, son variant est défini sur la vue de l'environnement qu'à l'agent auquel le but est associé. Ainsi, si b_a est un but de l'agent a dont le variant est v_b, on doit avoir :

$$\begin{cases} b_a \in buts[a] \\ v_b = variant(b_a) \\ dom(v_b) \in \mathcal{V}(a) \end{cases}$$

Par rapport aux deux agents a et c définis plus haut, on peut partitionner l'environnement du système en 4 :

$$\begin{cases} Env = E_1 \cup E_2 \cup E_3 \cup E_4 \\ E_1 \cap E_2 = \emptyset \\ E_1 \cap E_3 = \emptyset \\ E_1 \cap E_4 = \emptyset \\ E_2 \cap E_3 = \emptyset \\ E_2 \cap E_4 = \emptyset \\ E_3 \cap E_4 = \emptyset \\ E_1 = \mathcal{V}(c) \backslash \mathcal{V}(a) \\ E_2 = \mathcal{V}(a) \backslash \mathcal{V}(c) \\ E_3 = \mathcal{V}(a) \cap \mathcal{V}(c) \\ E_4 = Env \backslash \mathcal{V}(c) \backslash \mathcal{V}(a) \end{cases}$$

On a alors :

$$\begin{array}{l} Env' = action(c, Env) \\ Env' = action(c, E_1 \cup E_3 \cup E_3 \cup E_4) \\ Env' = action(c, E_1) \cup action(c, E_2) \cup \\ \qquad action(c, E_3) \cup action(c, E_4) \end{array}$$

Avec :

$$\begin{cases} action(c, E_1) = E_1' \\ action(c, E_2) = E_2 \\ action(c, E_3) = E_3' \\ action(c, E_4) = E_4 \end{cases}$$

Ainsi, si b est le but courant de l'agent a avec le variant v, on a :

$$\begin{array}{l} v(Etat(action(c, Env))) = \\ \qquad v(Etat(E_1') \cup Etat(E_1) \cup \\ \qquad Etat(E_3') \cup Etat(E_4)) \end{array}$$

Or le variant étant défini sur $\mathcal{V}(a)$, on a :

$$\begin{array}{l} v(Etat(action(c, Env))) = \\ \qquad v(Etat(E_2) \cup Etat(E_3')) \end{array}$$

Ainsi, on retrouve le fait que pour évaluer l'intérêt de l'action ac de l'agent c pour l'agent a, il suffit de simuler cette action sur une copie d'une partie de l'environnement constituée de la réunion des vues qu'ont les agents a et c de l'environnement.

6 Conclusion

L'article est en fait composé de deux parties. Dans la première, nous montrons comment l'association systématique d'un variant (et de sa fonction de calcul) à un but permet à un agent de gérer dynamiquement ses buts, et ce, de façon individuelle ou collective. Cette vision étant très théorique, nous montrons, dans un deuxième temps, comment il est possible de l'implanter au moyen d'une optimisation que nous prouvons formellement. Pour ce faire, nous utilisons une notation formelle, la notation B, pour modéliser un SMA. Ce n'est pas la première fois qu'un langage formel est utilisé. De nombreuses autres utilisations des méthodes formelles pour les Systèmes Multi-Agents ont déjà été réalisées. On peut citer principalement trois types d'approches : celles fondées sur Z [6], celles fondées sur la logique temporelle [8] et celles utilisant des réseaux de Pétri [7]. L'inconvénient de ce dernier type est la taille démesurément grande du graphe à construire dans le cas d'un système relativement compliqué. Concernant les deux premiers types de méthodes, ils sont relativement proches l'un de l'autre, mais la logique temporelle rajoute la possibilité de faire des preuves de propriétés de vivacité. Toutefois, les utilisations actuelles de ces deux approches, même lorsqu'elles sont outillées, ne font pas appel à des prouveurs et ne définissent pas non plus de règles de preuve, rendant l'intérêt de l'aspect formel plus limité. La méthode B que nous utilisons ici présente l'avantage d'être

outillée pour la preuve et certaines extensions permettent de réaliser des preuves de propriétés de vivacités [10]. Par rapport à la plupart des travaux présentés plus haut, nous ne modélisons ni le comportement d'un agent, ni celui d'un SMA, mais nous nous intéressons juste ici à la validation d'un modèle d'implantation. La vérification formelle ayant été faite, la réalisation opérationnelle est en cours.

Références

[1] J.-R. Abrial. *The B-Book*. Cambridge Univ. Press, 1996.

[2] Henri Abrias, editor. *1st Conf. on the B Method*, 1996.

[3] B-core. B-Toolkit User's Manual, release 3.2. Technical report, B-core, 1996.

[4] M. Carnot, C. Dasilva, B. Dehbonei, and F. Mejia. Formal Specification in the development of industrial applications : the subway speed control mechanism. In R. Groz, editor, *Proceedings of the Fifth International Conference on Formal Description Techniques, FORTE'92*, 1992.

[5] Digilog. Atelier B, Guide de l'utilisateur v3.0. Technical report, Digilog, 1997.

[6] Mark d'Inverno, Michael Fisher, Alessio Lomuscio, Michael Luck, Maarten de Rijke, Mark Ryan, and Michael Wooldridge. Formalisms for Multi-Agent Systems. Technical report, FORMAS'96, 1996.

[7] A. El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. A formal study of interactions in multiagent systems. In *Proceedings of ISCA International Conference in Computer and their Applications (CATA'99), April 1999.*, 1999.

[8] M. Fisher. A survey of concurrent METATEM – the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic - Proceedings of the First Intemational Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag : Heidelberg, Germany, 1994.

[9] C.B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, 1990.

[10] Bruno Mermet. Formal model of a multiagent system. In Robert Trappl, editor, *Cybernetics and Systems*, pages 653–658. Austrian Society for Cybernetics Studies, 2002.

[11] Gaële Simon, Marianne Flouret, and Bruno Mermet. A methodology to solve optimisation problems with MAS, application to the graph coloring problem. In Donia R. Scott, editor, *Artificial Intelligence : Methodology, Systems, Applications*, volume 2443. LNAI, 2002.

[12] J. M. Spivey. *Understanding Z : a specification language and its formal semantics*. Cambridge University Press, 1987.