Implanter des comportements d'agents prouvés via des automates

Marianne Flouret, Gaële Simon, Bruno Mermet

21 octobre 2005

Résumé

Afin de fournir un outil efficace permettant d'implanter des comportements d'agents validés, nous décrivons dans la première partie de cet article les arbres de décomposition des buts (*Goal Decomposition Tree*, ou GDT), un modèle permettant de spécificier aussi bien l'aspect déclaratif que l'aspect procédural du comportement d'un agent, et dont la correction peut être prouvée. Ce modèle est ensuite utilisé pour générer un automate de comportement en utilisant des schémas (ou *patterns*) de composition d'automates associés aux opérateurs utilisés dans le GDT. Ce processus permet d'obtenir une expression finie représentant tous les comportements valides des agents d'un système multiagent.

Mots Clef

Agent, Comportement, Automate, Expressions rationnelles

Abstract

In order to provide an effective tool allowing to implement validated agents behaviours, this paper first presents a Goal Decomposition Tree, a model to specify behaviours both in procedural and declarative ways which can be proved. This model is then used to generate a behaviour automaton using automata composition patterns associated to operators used in the tree. This process allows to obtain a finite expression representing all valid behaviours of agents of a MAS.

Keywords

Agent, Behaviour, Automaton, Rationnal expressions

1 Introduction

Le travail que nous présentons dans cet article s'insère dans une démarche plus globale dont l'objectif est de définir un processus complet permettant d'obtenir une implantation d'un SMA validé à partir d'une spécification formelle initiale du problème. Plus précisément, notre approche se décompose en quatre étapes :

– une méthode d'agentification [12] qui aide à déterminer l'ensemble des agents qui doivent être utilisés pour résoudre un problème donné ;
– un modèle conceptuel d'agent, reposant sur un arbre, pour aider à concevoir un comportement prouvable pour chacun des agents du système ;
– un système de preuve [7, 10] pour prouver le modèle d'agent ;
– un modèle d'implantation (SPACE [11]), reposant sur des automates, qui peut être généré automatiquement à partir du modèle conceptuel prouvé.

Pour pouvoir implanter et valider un agent, son comportement doit être clairement spécifié. Dans le cadre de nos recherches, cette spécification repose sur la notion d'arbre de décomposition des buts (*Goal Decomposition Tree* ou GDT). Cet arbre sert à décrire la façon dont un agent gère ses buts. Le principal apport de ce modèle est qu'il permet de prouver la correction du comportement de l'agent.

Avec les arbres de décomposition des buts, il est également possible de générer sous une forme implantable le comportement de l'agent auquel est associé le GDT. En effet, comme l'indiquent Riemsdijk *et al.* dans [14], le rôle essentiel des buts dans les agents n'est pas tant la possibilité de raisonner sur eux mais la capacité qu'ils ont à pouvoir engendrer un comportement («the biggest role of goals in agents is thus not the ability to reason about them but their motivational power of generating behaviour»). De plus, un plan ou un comportement est généré *par rapport* à un but (« A certain plan or behaviour is generated *because* of a goal»). C'est exactement ce qu'un GDT permet d'exprimer.

Une autre façon, complémentaire, de représenter les comportements intrinsèques des agents consiste à utiliser des automates. Les automates d'états finis ne sont pas seulement utilisés dans la théorie des langages mais aussi dans des domaines variés, tels que les systèmes et les réseaux, ou encore la compression d'images [4]. Les automates ou des structures similaires ont déjà été employés pour représenter le comportement d'agents. Dans [6], Ferber présente de tels exemples. Toutefois, dans notre approche, nous utilisons les propriétés du langage associée aux automates, ce que ne permettent pas par exemple les ATN, et nous utilisons également des opérations sur les structures d'automates.

En effet, un des avantages de ce formalisme est qu'il fournit une expression finie du comportement de l'agent qui correspond à toutes les successions possibles d'opérations. Les arbres de décomposition des buts et les automates sont deux formalisations du comportement d'un agent, et les automates peuvent être vus comme des interpréteurs du GDT. Lötzsch *et al.* utilisent des structures similaires [9] mais leurs machines à états ne correspondent pas à nos automates étant donné qu'ils ne modélisent les comportements au même niveau et qu'ils ne donnent pas de comportements

généraux. Dans cet article, nous présentons une traduction automatique d'un GDT vers l'automate correspondant. Le principal avantage de ce processus est d'obtenir une implantation validée du comportement d'un agent. De plus, comme l'automate est généré automatiquement, le concepteur peut se concentrer sur la conception du comportement. En conséquence, l'introduction d'erreurs durant la phase de codage est évitée (moyennant la correction du logiciel réalisant la traduction automatique, bien sûr !). Des solutions pour traduire des formules de la logique temporelle en automates ont déjà été proposées [15], mais elles font en général deux hypothèses qui ne sont pas vérifiées dans notre cas : l'ensemble des états du monde est fini et les formules spécifient des comportements de systèmes fermés, sans interaction avec leur environnement. Au cours de l'exécution du SMA, les automates permettent de connaître à tout moment l'état des agents, et également de garder une trace explicite de leur comportement passé.

Dans la partie suivante, nous décrivons le modèle d'arbre de décomposition des buts permettant de définir le comportement de l'agent. Dans la troisième partie, nous montrons le processus de traduction automatique d'un GDT en automate. Notre travail est illustré sur une partie d'une étude de cas proie-prédateur (développé de façon plus complète dans [13]).

2 Décomposition de buts en sous-buts

2.1 Introduction

Le principal avantage de l'utilisation d'un GDT est de fournir une description déclarative des buts. Plusieurs travaux ont déjà mis en exergue l'avantage de disposer d'une telle description déclarative [17, 14]. De nombreux modèles et systèmes multiagents mettent surtout l'accent sur les aspects procéduraux des buts, ce qui est essentiel si l'on veut obtenir un agent exécutable. Mais l'aspect déclaratif des buts est aussi très important. En effet, comme cela est précisé dans [17], *en omettant l'aspect déclaratif des buts, la possibilité de raisonner sur les buts est perdue. Si l'on ne sait pas ce qu'un but est sensé réaliser, on ne peut ni savoir si le but a été résolu ni détecter qu'il est impossible à résoudre* («by omitting the declarative aspect of goals, the ability to reason about goals is lost. Without knowing what a goal is trying to achieve, one can not check wether the goal has been achieved, check wether the goal is impossible»). Riemsdijk *et al.* [14] affirment que *les buts déclaratifs permettent de dissocier l'exécution d'un plan de la résolution d'un but* (« provide for the possibility to decouple plan execution and goal achievement»). Un GDT fournit une réponse partielle à ces besoins : comme nous le montrons dans les sections suivantes, un arbre de décomposition des buts permet de décrire les aspects procéduraux et déclaratifs de la gestion des buts par un agent. De plus, un GDT permet d'établir un lien entre un plan (un sous-arbre) et le but (un noeud) qu'il permet éventuellement de résoudre ; les deux notions sont donc également distinctes.

Les noeuds d'un GDT correspondent aux buts que l'agent doit résoudre. Comme dans [17], les buts sont considérés comme des états du monde que l'agent doit atteindre. Nous avons déterminé plusieurs types de buts, en utilisant trois critères différents. Cette typologie des buts est utile aussi bien pour vérifier la cohérence de l'arbre que pour établir l'automate de comportement.

Dans l'arbre, un but est décomposé en sous-buts en utilisant différents opérateurs de décomposition. De notre point de vue, les sous-buts correspondent à des conditions suffisantes pour la satisfaction d'un but parent. La méthode TAEMS [16] utilise une notion fortement similaire à la nôtre. Par contre, ce que nous appelons *but* est appelé *tâche*.

Un opérateur de décomposition encapsule un ensemble de mécanismes correspondant à une stratégie particulière de gestion des buts [14, 17]. Chaque opérateur est spécifié par quatre sémantiques différentes :

- une sémantique de décomposition des buts qui décrit comment un but peut être décomposé en sous-buts avec cet opérateur (nombre de sous-buts, nombre à résoudre, ordre de résolution, par exemple) ;
- une sémantique qui décrit comment déduire le *type* du but parent en fonction des types des buts fils ;
- une sémantique qui associe un *schéma de composition d'automates* à chaque opérateur. Ces schémas sont utilisés incrémentalement pour obtenir l'automate complet qui décrit le comportement de l'agent. Cette sémantique est décrite dans la partie 3 ;
- une sémantique qui permet d'associer un schéma de preuve local et un schéma de propagation de contexte à chaque opérateur. Ces deux types de schémas sont utilisés pour prouver le comportement de l'agent (c'est-à-dire vérifier que le comportement d'un agent lui permet de satisfaire son but principal). Cette sémantique est décrite dans [7].

La section 2.2 définit la notion de but telle que nous l'utilisons dans nos travaux et décrit la typologie des buts qui a été définie. La section 2.3 décrit l'ensemble des opérateurs qui permettent de décomposer un but en sous-buts dans un GDT. Pour chaque opérateur, les deux premières sémantiques décrites ci-dessus sont présentées. Enfin, la section 2.4 montre comment un GDT peut être construit en utilisant les outils décrits dans les deux parties précédentes.

2.2 Buts et typologie des buts

Dans le cadre des arbres de décompostion des buts, un but est essentiellement défini par un nom et une condition de satisfaction. Les conditions de satisfaction permettent de spécifier formellement les buts : elles fournissent la vision déclarative d'un but dont nous avons parlé dans la partie précédente. Un but est considéré comme satisfait (ou résolu) si sa condition de satisfaction est logiquement évaluée à *vrai*. Les conditions de satisfaction sont exprimées en utilisant un formalisme de la logique temporelle qui est

un sous-ensemble de TLA [8]. Elles sont définis sur des variables connues de l'agent (soit des attributs de l'agent, soit des données de l'environnement dont il a connaissance). Ainsi, spécifier les buts d'un agent aide à déterminer ses atributs et la vue que l'agent a de l'environnement.

Une typologie des buts a été définie afin de distinguer plus précisément différentes façons de gérer la décomposition des buts.

Dans le cadre d'un GDT, chaque but peut être caractérisé par trois critères qui peuvent être combinés indépendamment les uns des autres. Ces critères sont les suivants :

- **position du but dans l'arbre** : un but associé à un noeud feuille est appelé *but élémentaire*. Les autres buts sont appelés *buts intermédiaires*. Les buts élémentaires sont décrits non seulement par un nom et une condition de satisfaction, mais également par un ensemble d'actions de base (affectation de valeurs à des variables exprimées en logique temporelle). L'exécution de ces actions est supposée satisfaire le but, c'est-à-dire rendre sa condition de satisfaction vraie. Elles correspondent à l'aspect procédural des buts dont nous avons déjà parlé. Les buts intermédaires sont quant à eux décrits par un nom, une condition de satisfaction et un *arbre de décomposition local* (*Local Decompostion Tree* ou LDT). Un LDT contient un noeud racine correspondant au but intermédiaire et un opérateur de décomposition qui crée autant de branches (et de sous-buts) que spécifiés par la sémantique de l'opérateur. Il décrit comment le but intermédiaire peut être décomposé en sous-buts. Le nombre et l'ordre des sous-buts devant être résolus pour satisfaire le but parent dépend de l'opérateur choisi (voir la partie suivante pour plus de détails).
- **mode de satisfaction du but** : les buts *Nécessairement Satisfaisables* (NS) sont des buts dont la condition de satisfaction est forcément vraie une fois que tout ce qui devait être fait pour résoudre le but a été exécuté. Les buts *Non Nécessairement Satisfaisables* (NNS) sont complémentaires des buts précédents et correspondent au cas le plus fréquent. Pour ce type de but, on ne peut jamais être sûr que le but sera satisfait après sa résolution, c'est-à-dire après l'exécution des actions ou du sous-arbre qui y sont associés. Ce type de but permet de prendre en compte que certaines actions ou décompositions ne sont valables que dans certains contextes d'exécution (liés à l'état de l'environnement et des autres agents).
- **paresse du but** : Nous distinguons les buts paresseux (*lazy*) des buts non paresseux (*not lazy*). Dans le cas d'un but *paresseux* (L), l'action ou le LDT associés ne sont considérés que si la condition de satisfaction est évaluée à *faux* au préalable. Pour les buts *non paresseux* (NL), les actions ou le LDT associés sont toujours exécutés, quelle que soit la valeur de vérité de la condition de satisfaction *avant* la résolution du but.

Chacun des trois critères précédents a deux valeurs, ce qui implique l'existence de huit types de buts qui peuvent être utilisés dans les GDT. La figure 1 présente les notations graphiques que nous avons introduites pour ces critères. Le type effectif d'un but modifie le processus de résolution du but comme nous le montrons dans la section suivante. Nous noterons toutefois que, dans le processus de résolution, seuls les premier et troisième critères interviennent. De plus, seul le mode de satisfaction est pris en compte dans la sémantique de composition des types de buts des opérateurs.

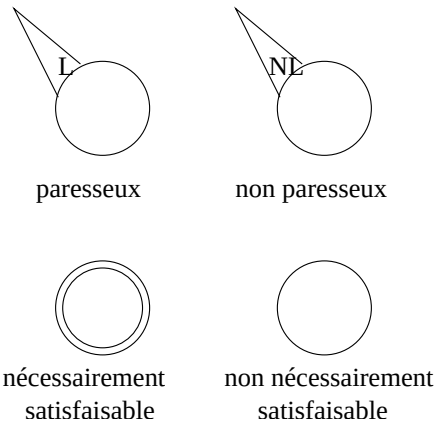


FIG. 1 – but paresseux, non paresseux, NS et NNS

2.3 Opérateurs de décomposition

Dans cette partie, nous décrivons les opérateurs de décomposition disponibles. Pour chaque opérateur, nous donnons les deux premières sémantiques que nous avons mentionnées, c'est-à-dire la sémantique de décomposition et la sémantique de composition des types. La sémantique de composition des types repose uniquement sur le critère de satisfaisabilité des buts. La sémantique de décomposition de chaque opérateur décrit le nombre de sous-buts à résoudre et dans quel ordre afin de satisfaire le but parent. Il est important de remarquer que la satisfaction de la décomposition *implique* la satisfaction du but parent, mais n'est pas forcément équivalente.

Nous avons défini les huit opérateurs suivants :

- **AND** : correspond au *et* logique,
- **OR** : correspond au *ou* logique,
- **SEQAND** : version séquentielle du *AND*,
- **SEQOR** : version séquentielle du *OR*,
- **SYNCSEQAND** : version synchronisée du *SEQAND* : le processus de résolution des deux sous-buts ne peut pas être interrompu par les autres agents,
- **SYNQSEQOR** : version synchronisée du *SEQOR*,
- **CASE** : permet la décomposition d'un but en sous-buts en fonction de conditions définies par des expressions logiques,
- **ITER** : permet d'exprimer les notions d'itération et de progrès dans le processus de résolution d'un but.

Pour des raisons de place, seuls les opérateurs SEQAND et ITER sont décrits plus précisément ci-dessous.

Opérateur SEQAND. Cet opérateur correspond à un opérateur «et séquentiel». Plus précisément, la seule différence avec l'opérateur *AND* est que les deux sous-buts doivent être résolus *dans l'ordre* (sous-but du fils gauche, puis sous-but du fils droit). Mais comme pour le *AND*, il est nécessaire de satisfaire les 2 sous-buts pour garantir la satisfaction du but parent. Les buts fils peuvent être soit NS, soit NNS. Si l'un des buts fils au moins est de type NNS, alors le but père sera de type NNS également. Sinon, il sera de type NS.

Opérateur ITER. Il s'agit d'un opérateur unaire. Il est essentiel pour représenter toutes les situations où la résolution d'un but n'est pas atomique. C'est pourtant un opérateur qui n'a pas d'équivalent dans la plupart des travaux proposant des modèles de représentation des buts d'un agent [5, 17, 14]. La sémantique de décomposition de cet opérateur établit que le but parent ne sera satisfait qu'après plusieurs étapes de satisfaction du sous-but. Autrement dit, la condition de satisfaction du sous-but doit être vraie plusieurs fois pour la condition de satisfaction du but parent devienne vraie. Cet opérateur est essentiel puisqu'il permet de prendre en compte la notion de progrès dans le processus de résolution d'un but.

Cet opérateur ne peut être utilisé que lorsque la satisfaction du sous-but implique un progrès dans la satisfaction du but parent. En d'autres termes, à chaque fois que le sous-but est satisfait, l'agent doit s'être rapproché de l'état de satisfaction du but parent. Toutefois, il peut arriver que le sous-but ne puisse être satisfait (suivant l'état de l'environnement par exemple). Dans ce cas, le degré de satisfaction du but parent reste le même et le sous-but doit à nouveau être résolu. Un aspect essentiel de cet opérateur est que le degré de satisfaction du but parent ne peut pas régresser lors d'une résolution du sous-but, même en cas d'échec. Si cela devait advenir, alors l'opérateur ITER n'aurait pas dû être choisi (le schéma de preuve associé à cet opérateur [7] nous permet de détecter formellement ce type de problème). La sémantique de composition des types de buts spécifie que le sous-but peut être nécessairement satisfaisable ou non. Par contre, le but parent est toujours nécessairement satisfaisable. En effet, étant donné que la sémantique de cet opérateur impose que l'on boucle jusqu'à ce que le but parent soit satisfait, si celui-ci pouvait ne pas être satisfaisable, alors on aurait une boucle infinie.

2.4 Processus de conception du GDT

Un arbre de décomposition des buts (GDT) spécifie comment chacun des buts peut être résolu par un agent. Plus précisément, le noeud racine est associé au but principal de l'agent, c'est-à-dire à celui qui lui a été assigné durant la phase d'agentification [11, 18]. Si ce but est résolu par l'agent, l'agent est considéré comme *satisfait* du point de vue du système multiagent. L'arbre décrit comment ce but peut être décomposé pour être résolu en utilisant la solution la plus adaptée au contexte de l'agent. Chaque feuille de l'arbre correspond à un but élémentaire. Les autres noeuds sont associés à des buts intermédiaires qui sont eux-mêmes décomposés grâce aux opérateurs de décomposition que nous avons présentés dans la partie précédente. On notera que l'arbre dans sa globalité peut être vu comme une collection de plans locaux permettant de résoudre chacun des buts intermédiaires. Un plan local correspond à un arbre de décomposition local (LDT) associé à un sous-but. La principal différence avec les plans utilisés dans [1] est que, dans le cadre d'un GDT, les plans sont hiérarchisés.

Le processus de conception du GDT se décompose en 4 étapes. Dans un premier temps, un arbre est construit par le concepteur, en partant du but racine et en utilisant une démarche descendante. Cette première étape permet de définir les sous-buts ainsi que leur condition de satisfaction, les buts élémentaires avec les actions associées ainsi que les opérateurs de décomposition. Le concepteur doit également déterminer, pour chaque but, si celui-ci est paresseux ou non. Dans la deuxième phase de conception du GDT, le concepteur doit décider pour chaque but élémentaire s'il est nécessairement satisfaisable ou pas, ce qui se détermine relativement aisément en fonction des actions associées au but. La troisième étape consiste à déterminer automatiquement , par un processus ascendant, pour chaque but intermédiaire, s'il est nécesairement satisfaisable ou non, en utilisant la sémantique de chacun des opérateurs de décomposition. Lors de cette phase, des incohérences ou des simplifications peuvent être détectées. Dans ce cas, un retour à la première phase s'impose.

Une fois que ces trois premières étapes ont été effectuées avec succès, la quatrième phase, à savoir la preuve de l'arbre, peut être abordée. Le processus utilisé pour la preuve est décrit dans [10, 7]. Là encore, des incohérences dans l'arbre peuvent être détectées suite à des échecs de preuve.

Après ces 4 étapes, l'arbre de décomposition des buts de l'agent est validé. On peut donc en déduire l'automate de comportement de l'agent, selon le processus décrit dans la section 3.

Comme nous l'avons expliqué précédemment, la construction de l'arbre amène à définir, outre les buts, les propriétés, les variables et les actions de l'agent. En conséquence, le GDT et le processus de conception associé peuvent être considérés comme des outils de conception pour un modèle d'agent validé dans le cadre de la conception de SMA.

Ce modèle a été appliqué avec succès pour décrire et prouver le comportement d'un agent prédateur dans un système proie/prédateurs [13]. La figure 2 montre une partie du GDT de cette application qui décrit le comportement d'un prédateur essayant de se rapprocher d'une cellule cible (en vue de se rapprocher de la proie). Cette cellule cible est supposée être accessible en un déplacement du prédateur, mais elle n'est pas forcément libre. C'est pourquoi le prédateur doit éventuellement attendre qu'elle devienne libre. Aussi, le but F est satisfait si le prédateur a effectivement atteint la cellule cible. Le but I est satisfait si le prédateur s'est déplacé vers une position plus proche de la cellule

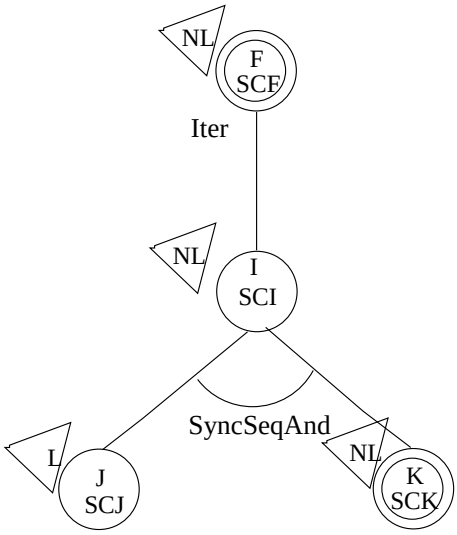


FIG. 2 – sous-arbre d'un prédateur

cible que sa position précédente. Le but J est quant à lui satisfait si la cellule cible est libre (aucun agent ne l'occupe). L'action associée à ce but consiste à attaquer l'agent situé sur la cellule cible. Ce but étant paresseux, cette action n'est effectuée que si un agent est présent sur la cellule cible. Le but K est satisfait si le prédateur est sur la cellule cible. L'action associée à ce but est un mouvement du prédateur vers la cellule cible libre, ce qui est toujours satisfaisable dans notre modèle. Ce but ne peut être satisfait que si le but J a été satisfait avant. Comme un opérateur synchronisé est utilisé entre les buts J et K, on est sûr que la cellule cible est encore libre lorsque le but K est résolu. On remarque que dans ce cas, l'opérateur ITER est utilisé de façon un peu particulière : si I est satisfait une fois, alors F est aussi satisfait. Cependant, on doit utiliser cet opérateur car I peut échouer (à cause de J) et dans ce cas, la résolution de I doit être entreprise de nouveau.

3 Implanter des comportements d'agents par des automates

Dans notre approche, l'implantation des comportements d'agents repose sur des automates chaîne-chaîne (en fait des transducteurs) [2]. Dans la section suivante, nous montrons que les buts et les opérateurs de décomposition peuvent être représentés par des schémas de composition d'automates.

3.1 Automate de comportement

Comme nous l'avons présenté en partie 2.1, une sémantique en terme de schéma de composition d'automates est associée à chaque opérateur de décomposition. Des automates de base sont également associés aux buts élémentaires, avec des propriétés différentes selon le type de ces buts : nécessairement satisfaisable ou pas, paresseux ou pas. Pour les buts intermédiaires, étant donné qu'ils sont décomposés en sous-buts au moyen d'opérateurs de décomposition, les automates correspondant sont obtenus par composition, suivant l'opérateur choisi, des automates obtenus précédemment pour chacun des sous-buts. Cette composition est spécifiée par un schéma de composition d'automates. Ainsi, la construction de l'automate chaîne associé à un comportement d'agent est obtenue incrémentalement par une successsion de telles opérations. Les automates ainsi obtenus donnent un *schéma de comportement* des agents mais n'ont pas pour but de définir tous les éléments du comportement des agents, comme les interactions entre agent ou les attentes de messages (mais elles peuvent apparaître dans les actions).

Avant de présenter les automates élémentaires et les règles de composition, rappelons brièvement la notion d'*automate chaîne-chaîne*. Un automate chaîne-chaîne est définir par un sextuplet $(\Sigma, Z, Q, I, F, \delta)$ tel que : Σ est un alphabet d'entrée fini, Z est un alphabet de sortie fini, Q est l'ensemble fini des états, $I \subseteq Q$ est l'ensemble des états initiaux, $F \subseteq Q$ est l'ensemble des états finaux, $\delta : Q \times \Sigma \rightarrow Q \times Z$ est l'application associée aux transition. Les *expressions rationnelles* sont obtenues à partir des éléments de Σ ou Z par un nombre fini de compositions de lois rationnelles telles que ($\cup$, $\cdot$, $*$). L'expression finie du comportement d'un agent [3] est en fait une telle expression rationnelle, avec l'équivalence suivante, étant donné que nous travaillons dans un contexte logique : $\cup$ correspond au Ou logique, $\cdot$ correspond à notre opérateur SeqAnd, et $*$ correspond à l'opérateur Iter. Les autres opérateurs peuvent être exprimés en fonction de ceux-ci. Pour SyncSeqAnd et SyncSeqOr, la seule différence avec les opérateurs SeqAnd et SeqOr est la synchronisation avec des comportements d'autres agents, ce qui n'intervient pas dans les compositions d'automates. Pour l'opérateur Case, considérons deux conditions de satisfactions A et B. Si nous notons C_A la condition devant être évaluée avant le test de A, alors $A\ Case\ B = (C_A\ SeqAnd\ A) Or (\neg C_A\ SeqAnd\ B)$. Pour les autres opérateurs, nous pouvons remarquer que l'on a : $A\ And\ B = (A\ SeqAnd\ B)\ Or\ (B\ SeqAnd\ A)$, $A\ SeqOr\ B = A\ Or\ (\neg A\ SeqAnd\ B)$.

L'adaptation des automates chaîne-chaîne pour permettre de décrire le comportement d'un agent a découle d'un choix spécifique sur la signification des états et des étiquettes associées aux transitions. Ils sont définis ainsi :

- Σ est l'ensemble fini des conditions de satisfaction logiques sc et leur négation de l'agent a ;
- Z est l'ensemble fini des actions possibles (décrites dans la section 2.2) ;
- Q représente l'ensemble des buts et sous-buts de l'agent, auquel les deux états $'+'$ et $'-'$ (succès et échec) sont rajoutés ;
- $I = \{i\}$ est l'état de départ de l'agent a, c'est-à-dire correspondant à sa situation initiale ;
- $F = \{+, -\}$.

Un *chemin* d'un automate est une séquence d'états

$p_0, \cdots, p_n$ telle que $\forall 1 \leq j \leq n$ il existe $a_j \in \Sigma$, $z \in Z$, $(p_j, z) \in \delta(p_{j-1}, a_j)$.

Un *chemin réussi* d'un automate est un chemin partant de l'état initial $p_0 \in I$ et aboutissant à un état final $p_n \in F$ avec $\{p_j\}_{0 \leq j \leq n} \in Q^{n+1}, n \geq 1$ et $\{a_j\}_{1 \leq j \leq n} \in \Sigma^n$ tel que $\delta(p_j, a_{j+1})$ pour $0 \leq j \leq n$ existe. Le langage de l'automate est l'ensemble des étiquettes des chemins réussis (il correspond à tous les comportements possibles de l'agent). Ici, le langage que nous considérons est l'ensemble des compositions logiques des formules de Σ. De plus, chaque chemin réussi aboutissant à l'état $+$ correspond à un parcours possible de l'arbre de décomposition des buts de l'agent a. Un *comportement valide* correspond donc à un chemin réussi dans l'automate se terminant par l'état $+$. Un tel chemin est appelé *chemin valide*.

Les actions de Z sont considérées comme étant associées aux états de l'automate, mais il est équivalent de les représenter sur les transitions, associées aux conditions logiques de Σ. Notons également qu'implicitement, dans chaque état, une action de test est exécutée pour savoir quelle transition choisir. Deux résultats finis sont obtenus à partir d'une telle structure : d'une part, l'ensemble de toutes les combinaisons possibles de conditions logiques permettant de satisfaire le but racine de l'agent, et d'autre part, en consultant le contenu des états, la séquence d'actions exécutées par l'agent a, c'est-à-dire ses comportements valides. Autrement dit, la vue sous forme d'automate donne une formalisation rationnelle du comportement décrit dans l'arbre de décomposition des buts.

3.2 Automates de base et schémas de composition d'automates

L'automate de comportement d'un agent est construit par composition d'automates élémentaires en utilisant des schémas de composition d'automates associés aux opérateurs de décomposition.

Les automates élémentaires correspondant aux buts élémentaires (les feuilles de l'arbre) sont définis ci-dessous. Soit a un agent et N un noeud feuille de son GDT associé à un sous-but élémentaire G. Selon la typologie des buts définis à la section 2.2, il existe quatre types de buts élémentaires, selon qu'ils sont paresseux ou non, nécessairement satisfaisables ou non. La figure 3 montre les automates élémentaires correspondant à chacun de ces cas. Nous notons A_c les actions associées au but.

On peut remarquer que l'automate d'un but nécessairement satisfaisable a le seul état $+$ comme état final.

La construction d'un automate de comportement de l'agent commence donc par la construction d'un automate de base pour chaque but élémentaire. Ces automates sont ensuite combinés incrémentalement grâce aux schémas de composition d'automates associés aux opérateurs utilisés dans le GDT. De cette étape résulte un nouvel ensemble d'automates que l'on va pouvoir à nouveau combiner. Le processus continue incrémentalement jusqu'à obtenir l'automate complet de comportement de l'agent.

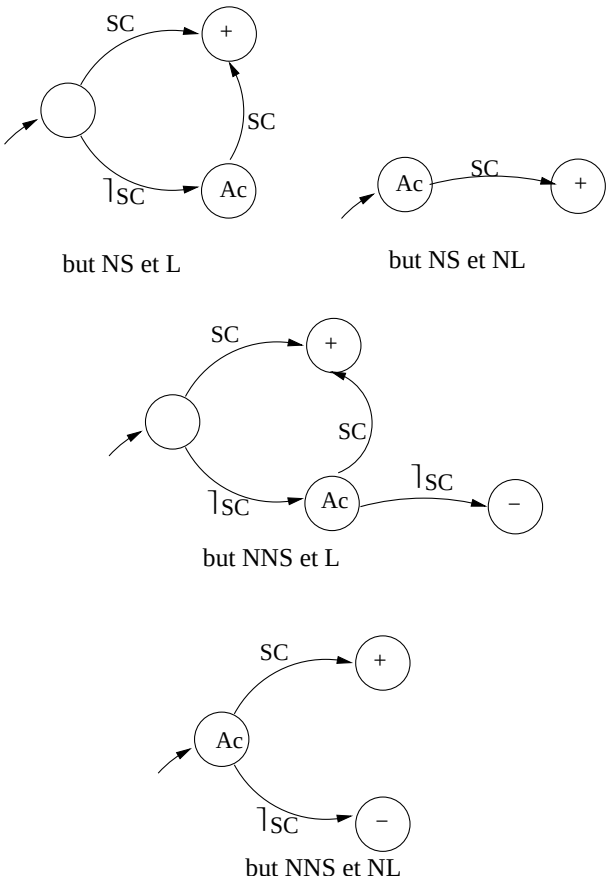


FIG. 3 – Automates des différents types de buts élémentaires

Nous allons à présent décrire les schémas de composition associés à chaque opérateur. Soit B l'automate du but parent et $B1$ et $B2$ les automates associés aux buts fils. Un schéma de décomposition spécifie comment combiner $B1$ et $B2$ pour construire B en fonction de l'opérateur utilisé. Sur les figures 4,5 et 6, nous représentons les schémas de composition associés à chacun des opérateurs.

L'état initial de l'automate résultant est, classiquement, représenté par un cercle simple avec une flèche entrante ne venant d'aucun autre état.

Les automates B_1 et B_2 (élémentaires ou non) sont abstraits et uniquement représentés avec leur état initial et les deux états $+$ et $-$. Quand, à l'issue d'une composition, un état qui était initial ne l'est plus, il est représenté avec un cercle en pointillés. Les flèches en pointillés partant d'un état initial représentent de nouvelles transitions atteignant des états qui étaient précédemment atteints par des transitions provenant d'états en pointillés (ou états initiaux) et avec les mêmes étiquettes. Les flèches en gras allant d'un état X ou atteignant un état Y représentent l'ensemble des transitions qui partaient de X ou qui atteignaient Y dans les automates $B1$ et $B2$. Les flèches barrées d'une croix symbolisent des transitions supprimées dans l'automate résultant et leur remplacement par d'autres avec les mêmes

étiquettes (cela revient à déplacer les transitions). Lorsque les états $+$ et $-$ sont dessinés dans un carré, cela signifie qu'ils sont supprimés dans l'automate résultant.
Pour tous les schémas, il est important de préciser que chaque automate à composer doit avoir exactement un état $+$ et un état $-$, ce dernier étant ajouté (sans flèche) s'il n'existe pas (car il a été supprimé lors d'une composition précédente).
Tous les états inacessibles (après la suppression de toutes les transitions les atteignant) sont supprimés de l'automate résultant afin de garder exactement un et un seul état $+$ et au plus un état $-$.
Dans les figures suivantes décrivant les schémas de composition, le but parent est un but non paresseux nécessairement satisfaisable. Une politique de choix (hasard, etc.) doit être définie pour le premier test dans le cas des opérateurs And et Or.

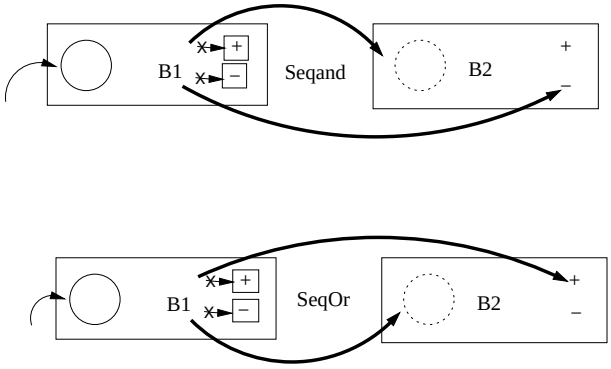


FIG. 4 – Schémas de composition des opérateurs SeqAnd et SeqOr

Notons que les schémas des opérateurs SyncSeqAnd et SyncSeqOr sont les mêmes que ceux des opérateurs SeqAnd et SeqOr.
En ce qui concerne le schéma de composition de l'opérateur Case, on suppose que dans le GDT $cond1$ et $cond2$ sont les deux conditions qui vont permettre de choisir le premier ou le deuxième sous-but. Pour construire l'automate résultant de la composition, nous introduisons deux nouveaux états permettant d'atteindre les états initiaux des automates associés aux buts fils grâce à des transitions étiquetées par les conditions $cond1$ et $cond2$.
Pour l'opérateur Iter, nous supposons, sur la figure 6, que A est le but parent de B_1 et SC_A sa condition de satisfaction. La flèche en pointillés représente une unique nouvelle transition.
Lorsque le but parent est un but paresseux, l'automate résultant doit être modifié par l'addition d'un état et de deux flèches, comme cela est représenté sur la figure 7, quel que soit l'opérateur x utilisé sur la figure.
Quand un but parent n'est pas nécessairement satisfaisable, l'automate obtenu pour ce but doit être modifié par l'application du schéma représenté sur la figure 7.

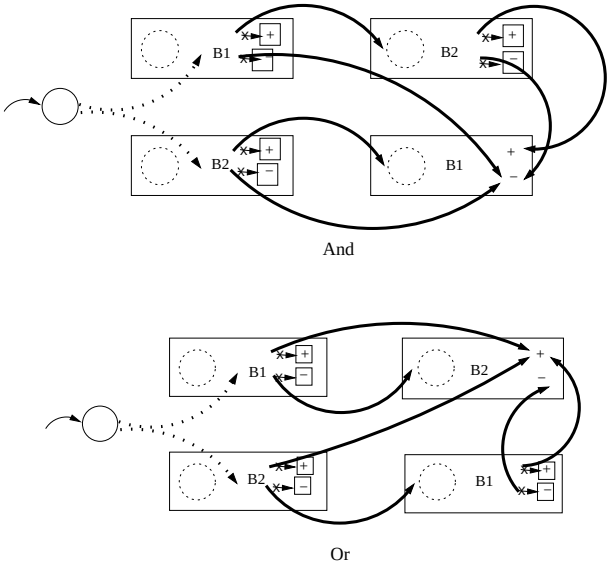


FIG. 5 – Schémas de composition des opérateurs And et Or

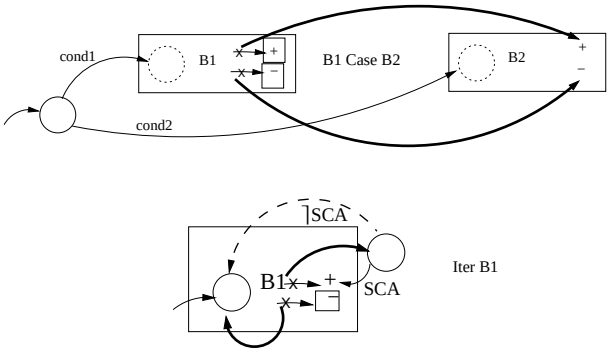
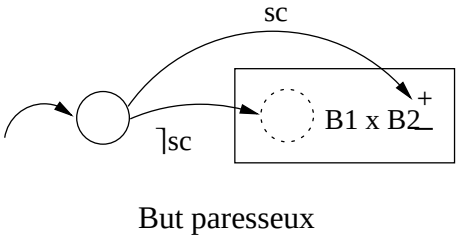


FIG. 6 – Schémas de composition des opérateurs Case et Iter

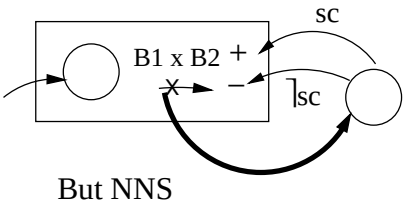
Il est évident que l'ordre d'application des deux schémas précédents (pour les buts paresseux et pour les buts non nécessairement satisfaisables) n'intervient pas.

3.3 Etude de cas

Nous donnons ici un exemple montrant comment ces schémas sont utilisés pour obtenir l'automate de comportement d'un sous-arbre d'un GDT que nous avons construit [13] pour le comportement d'un prédateur dans un système proie-prédateurs. Ce sous-arbre est représenté sur la figure 2. Cet exemple montre que les étiquettes des chemins valides de l'automate complet et l'expression logique donnée par l'arbre de décomposition sont équivalents.
Les figures 8 et 9 représentent les automates des buts J et K de la figure 2.
Après composition de ces deux automates avec le schéma de composition de l'opérateur SyncSeqAnd (figure 4), nous obtenons l'automate de la figure 10. A partir de ce nouvel automate, et en appliquant le schéma de composition de



But paresseux



But NNS

FIG. 7 – buts parents paresseux et non nécessairement satisfaisables

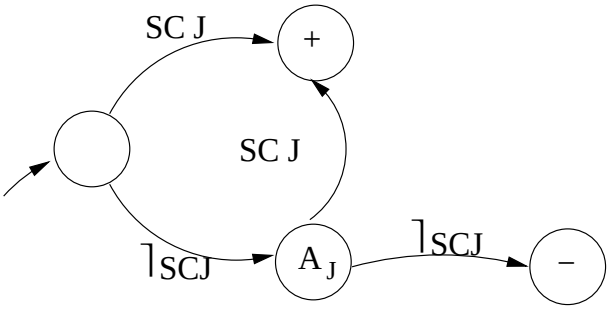


FIG. 8 – Automaton de J

l'opérateur Iter, nous arrivons à l'automate représenté figure 11.

Nous pouvons maintenant donner l'ensembe de tous les comportements possibles de l'agent dont le GDT est représenté sur la figure 2.

Soit $Y = (\rceil SCJ \cdot SCJ \cdot SCK) \cup (SCJ \cdot SCK) \cup (\rceil SCJ \cdot \rceil SCJ \cdot SCI)$, et soit $X = (\rceil SCJ \cdot \rceil SCJ \cdot \rceil SCI)^* \cdot Y$.

Alors l'expression $X \cdot (\rceil SCF \cdot X)^* \cdot SCF$ représente toutes les successions possibles de tests de conditions logiques effectués par l'agent associé au but F. Cela se termine de façon évidente sur la valeur $vraie$ pour la condition logique CF (en effet, le but but F est un but nécessairement satisfaisable).

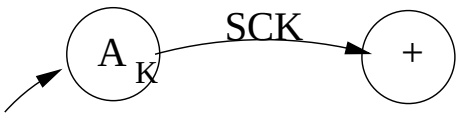


FIG. 9 – Automate de K

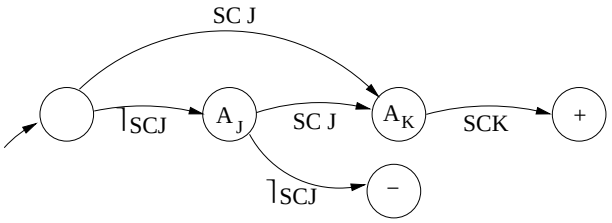


FIG. 10 – Automate intermédiaire

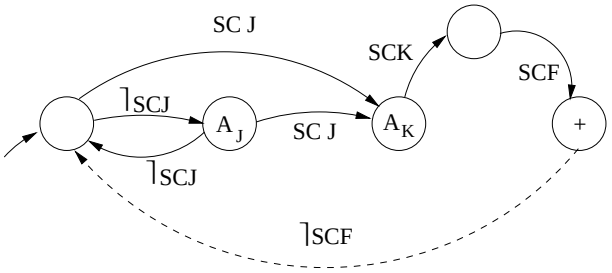


FIG. 11 – Automate final

4 Conclusion

Cet article présente une partie d'un processus global de conception de SMA. Cette partie met l'accent sur des outils permettant de spécifier et d'implanter des comportements d'agents validés. Le comportement est spécifié au moyen d'un GDT prouvable qui peut être traduit automatiquement en un automate, permettant ainsi d'obtenir une implantation validée de l'agent. Un des avantages de la représentation sous forme d'automate est de rendre possible l'utilisation des opérations rationnelles classiques. Les méthodes de minimisation [2] peuvent aussi être appliquées. Ceci permet de réduire l'automate, et donc, par l'application d'opérations inverses, d'obtenir un GDT réduit (si des buts intermédiaires sont susceptibles d'être éliminés). De plus, l'expression du comportement obtenue sous une telle forme peut être facilement et directement implantée, quelle que soit la plateforme de développement de SMA que l'on utilise.

Références

[1] R.H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifiable multi-agent programs. In M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *ProMAS*, 2003.

[2] P. Caron and M. Flouret. Glushkov construction for series : The non commutative case. *Int. J. Comput. Math.*, 4(80) :457–472, 2003.

[3] P. Caron, M. Flouret, and From Glushkov. WFAs to rational expressions. In *Proceeding of DLT'2003*, volume 2710 of *Lecture Notes in Computer Science*, pages 373–385. Springer, 2003.

[4] K. Culik and J. Kari. Image compression using weighted finite automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, pages 599–616. Springer, 1997.

[5] F.S. de Boer, K.V. Hindrinks, W. van der Hoek, and J.-J.Ch. Meyer. Agent programming with declarative goals. In *Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Language*, pages 228–243, 2000.

[6] J. Ferber. *Les systèmes multi-agents*. InterEditions, 1995.

[7] D. Fournier, B. Mermet, and G. Simon. A compositional proof system for agent behaviour. In *SASEMAS'05*, 2005. à paraître.

[8] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 1994.

[9] M. Lötzsch, J. Bach, H-D. Burkhard, and M. Jüngel. Designing agent behavior with the extensible agent behavior specification language xabsl. In Springer, editor, *Proceedings RoboCup 2003*, 2004.

[10] B. Mermet and D. Fournier. Variant extensions to prove mas behaviours. In *Artificial Intelligence : Methods, Systems and Applications (AIMSA'04)*, 2004.

[11] B. Mermet, G. Simon, D. Fournier, and M. Flouret. SPACE : A method to increase tracability in MAS Development. In *ProMAS*, volume 3067. LNAI, 2004.

[12] G. Simon, M. Flouret, and B. 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.

[13] G. Simon, B. Mermet, D. Fournier, and M. Flouret. The provable goal decomposition tree : a behaviour model of an agent. Technical Report XX, Université du Havre, 2005.

[14] M.B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J.Ch. Meyer. Dynamics of declarative goals in agent programming. In *Proceedings of Declarative Agent Languages and Technologies (DALT'04)*, 2004.

[15] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logics In Computer Science (LICS'86)*, pages 332–344, 1986.

[16] R. Vincent, B. Horling, and V. Lesser. An agent infrastructure to build and evaluate multi-agent systems : the java agent framework and multi-agent system simulator. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, 2001.

[17] M. Winikoff, L. Padham, J. Harland, and J. Thamgarajah. Declarative & procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, 2003.

[18] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3) :285–312, 2000.