Vers une modélisation formelle basée sur le raffinement des systèmes multi-agents auto-organisateurs

Zeineb Graja [1,2], **Frédéric Migeon** [2], **Christine Maurel** [2], **Marie-Pierre Gleizes** [2], **Ahmed Hadj Kacem** [1]

1. *Unité de Recherche en Développement et Contrôle d'Applications Distribuées, Faculté des Sciences Economiques et de Gestion - Université de Sfax, Tunisie*
   zeineb.graja@redcad.org,ahmed.hadjkacem@fsegs.rnu.tn
2. *Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier 118, route de Narbonne, 31062 Toulouse - France*
   {migeon,maurel,gleises}@irit.fr

RÉSUMÉ. *Le développement de SMA auto-organisateurs manque encore de méthodes rigoureuses de vérification garantissant la robustesse et la résilience du système conçu. De telles assurances peuvent être obtenues grâce à l'application de méthodes formelles. Mais l'intégration de ces techniques de vérification reste encore modeste due à la complexité liée à la dynamique des SMA auto-organisateurs qui fait émerger leur fonction globale. Dans cet article, nous explorons le potentiel des langages formels, en particulier B-événementiel et la logique TLA, pour prouver des propriétés liées à la convergence et la résilience. Nous supposons que ces propriétés pourront d'abord être observées au niveau global par simulation. Les techniques formelles nous permettront ensuite d'en faire la preuve. Notre travail est illustré par l'étude de cas des fourmis fourrageuses.*

ABSTRACT. *The development of self-organizing MAS still lacks rigorous verification methods to ensure the convergence and resilience of the designed system. Such insurances can be obtained through the application of formal methods. However, the integration of these techniques is still modest due to the complexity of the dynamics of self-organizing MAS which makes the overall function emerging. In this article, we explore the potential of formal languages, in particular Event-B and the TLA logic to prove the convergence and the resilience of the system. We assume that these properties can first, be observed at the global level by simulation. Then, Formal techniques allow us to prove them. Our work is illustrated by the foraging ant's case study.*

MOTS-CLÉS : *SMA auto-organisateurs, fourmis fourrageuses, vérification formelle, B-événementiel, TLA.*

KEYWORDS: *self-organizing MAS, foraging ants, formal verification, Event-B, TLA.*

DOI:10.3166/RIA.30.159-183

1. Introduction

Les systèmes multi-agents (SMA) auto-organisateurs sont constitués de plusieurs entités autonomes, appelées agents, situées dans un environnement et interagissant en vue d'accomplir une tâche bien déterminée. Chaque agent a une connaissance partielle de son environnement et possède ses propres objectifs. La fonction globale du système émerge des interactions entre les agents et entre les agents et l'environnement (Di Marzo Serugendo *et al.*, 2005). La notion d'émergence est définie généralement par la phrase "*Le tout est plus que la somme des parties*". Ainsi, nous distinguons deux niveaux d'observation des SMA auto-organisateurs : le niveau micro (les parties) qui correspond aux comportements des agents locaux et le niveau macro (le tout) dans lequel est observé le comportement global du système.

Selon (Georgé, 2004), les phénomènes émergents sont le résultat d'un processus d'auto-organisation : un ensemble de mécanismes basés sur des règles locales non contrôlés par une entité externe au système et nécessitant une interdépendance entre les parties du système. Dans les SMA auto-organisateurs, ce sont ces mécanismes d'auto-organisation qui permettent aux agents d'adapter leurs comportements et de faire face aux changements de l'environnement.

La conception de SMA auto-organisateurs se fait généralement selon une approche descendante. Ainsi, l'effort du concepteur est focalisé sur le développement du comportement local des agents. Ce comportement doit permettre de faire émerger la fonction globale du système. Il est conçu en se basant sur une heuristique indépendante de la fonction du système comme la coopération par exemple (la théorie des AMAS (Gleizes, 2012)).

L'un des principaux défis relatifs à l'ingénierie des SMA auto-organisateurs est de donner des assurances et des garanties liées à la correction du système, sa convergence et sa résilience. La correction se réfère à la satisfaction des différentes contraintes liées aux activités des agents. La convergence garantit que le système est capable d'atteindre son but (Serugendo, 2009). La résilience informe sur la capacité du système à s'adapter face à la dynamique de l'environnement (Serugendo, 2009).

Cet article représente une première contribution dans un travail ambitieux ayant pour objectif de vérifier les SMA auto-organisateurs formellement (par le biais de la preuve de théorèmes). Pour cet article, nous nous sommes situés dans un cas particulier où la fonction émergente du système est connue et observée par simulation. Notre objectif est de prouver formellement ces propriétés observées relatives à la convergence et la résilience du système.

Nous proposons une modélisation formelle pour le comportement local des agents basée sur des étapes de raffinement à l'aide du formalisme *B-événementiel* (Abrial, 2010). Notre stratégie de raffinement garantit la correction du système. Afin de prouver les propriétés globales souhaitées liées à la robustesse et à la résilience, nous avons eu recours à la logique temporelle des actions (*TLA*) et ses règles de preuve d'équité. L'utilisation de TLA a été proposée dans (Méry, Poppleton, 2013) dans le cadre des

protocoles de populations pour raisonner sur des propriétés de vivacité. Cette logique offre des moyens plus explicite pour prouver des propriétés de vivacité sur des modèles formalisés à l'aide de *B-événementiel*. Nous jugeons qu'elle convient bien aux SMA auto-organisateurs. Notre travail est illustré par le cas d'étude des fourmis fourrageuses.

Cet article est organisé comme suit. La section 2 présente un panorama sur les travaux antérieurs. La section 3 donne une description du langage *B-événementiel* et de la logique *TLA*. Dans la section 3, notre modélisation formelle des SMA auto-organisateurs ainsi que la stratégie de raffinement sont présentées. Une illustration de nos propositions avec les fourmis fourrageuses est donnée dans la section 4. Finalement, la section 5 conclut l'article et dresse nos perspectives.

2. Travaux antérieurs

La vérification des systèmes auto-organisateurs a fait l'objet de plusieurs travaux de recherche. La majorité des approches s'appuient sur la simulation et les techniques du model-checking stochastique pour mesurer l'impact des différents paramètres sur le comportement du système. Dans (Gardelli *et al.*, 2006), Gardelli utilise le langage $Pi - Calculus$ stochastique pour la modélisation d'un SMA auto-organisateur pour la détection d'intrusions. L'outil $SPiM$ (Stochastic Pi-Machine) a servi pour effectuer des simulations permettant d'évaluer l'impact de certains paramètres tels que le nombre d'agents et la fréquence des inspections sur le comportement du système.

Dans (Casadei, Viroli, 2009), une approche hybride pour la modélisation et la vérification de systèmes auto-organisateurs a été proposée. Cette approche utilise la simulation stochastique pour la modélisation du système à l'aide des chaînes de Markov et la technique de model-checking probabiliste pour la vérification. Pour éviter le problème d'explosion d'état, rencontré avec les model-checkers, les auteurs proposent d'avoir recours au model-checking approximatif qui repose sur les simulations.

Les travaux de (Konur *et al.*, 2012) utilisent l'outil $PRISM$ et le model-checking probabiliste pour la vérification des comportements des essaims de robots et en particulier les robots fourrageurs. Les auteurs ont réussi à vérifier, sur plusieurs scénarios, certaines propriétés en utilisant la logique PCTL (*Probabilistic Computation Tree Logic*). Ces propriétés renseignent en particulier sur la probabilité que l'essaim acquiert une certaine quantité d'énergie dans un délai donné et pour un nombre d'agents bien déterminé. Des simulations ont été également utilisées pour montrer la corrélation entre la densité des robots fourrageurs dans l'arène et la quantité d'énergie acquise.

Dans cet article, nous utilisons des techniques de vérification différentes basées sur la preuve de théorèmes. Notre objectif est double : 1) permettre au concepteur de spécifier formellement, par des raffinements, le comportement local des agents et 2) prouver des propriétés aux niveaux local et global. Parmi les travaux visant ces objectifs, nous pouvons citer (Hilaire *et al.*, 2008), (Pereverzeva *et al.*, 2012), (Simonin *et al.*, 2011). Dans (Hilaire *et al.*, 2008), les auteurs proposent la formalisation d'un

métamodèle organisationnel en combinant le langage *ObjectZ* avec les diagrammes *tats/transitions*. Ce métamodèle organisationnel permet de spécifier formellement les concepts de *Rôle*, *Interaction* et *Organisation* mais ne permet pas de raisonner sur les aspects des SMA auto-organisateurs relatifs à la convergence et la résilience. L'objectif de cette spécification a été de valider le modèle obtenu par simulation ou animation.

Pereverzeva et *al.* utilisent *B-événementiel* pour la modélisation des SMA tolérants aux fautes selon une stratégie de raffinement descendante. Cette stratégie part d'une modélisation de l'objectif global du système et permet par raffinement d'identifier le comportement local de chaque agent.

Dans notre travail, nous proposons une stratégie de raffinement ascendante que nous jugeons plus adéquate et plus naturelle pour modéliser les SMA auto-organisateurs. En effet, cette stratégie nous permet de modéliser le comportement local des agents puis de raisonner sur le comportement global du système. Les travaux présentés dans (Simonin *et al.*, 2011) proposent des patrons de modélisation basés sur le langage *B* pour spécifier des agents situés selon le modèle *Influence/réaction*. Ce modèle fonctionne selon le cycle *percevoir-décider-agir* que nous adoptons également. L'objectif de la formalisation a été de garantir des propriétés de sécurité mais pas des propriétés de convergence et de résilience comme le cas de notre travail.

2.1. Le langage B-événementiel

Le formalisme *B-événementiel* a été proposé par J-R. Abrial (Abrial, 2010) comme une évolution du langage *B*. Il permet un développement *correct par construction* pour les systèmes distribués et réactifs. *B-événementiel* utilise la théorie des ensembles pour la modélisation des systèmes et la preuve de théorèmes pour assurer une vérification formelle. Le concept utilisé pour faire un développement formel est celui de modèle. Un modèle est constitué de composants qui peuvent être de deux types : machine et contexte.

Un contexte est la partie statique du modèle et comprend des ensembles et des constantes définies par l'utilisateur avec les axiomes correspondants. Une machine décrit la partie dynamique du modèle et modélise le comportement du système à concevoir. Une machine est composée d'un ensemble de variables v et un ensemble d'événements ev_i. L'état du système est donné par les valeurs attribuées aux différentes variables déclarées dans la machine. Les événements servent à définir le changement de l'état du système. Ainsi, un événement est défini par un ensemble de paramètres p, une garde qui donne les conditions nécessaires à l'activation de l'événement $G\_evi(p, v)$ et une action $A\_evi(p, v, v')$ qui décrit la relation entre l'ancien état du système (v) et son nouvel état (v'). L'action $A\_evi(p, v, v')$ est aussi appelée un prédicat avant-après.

L'action peut consister en plusieurs affectations qui peuvent être déterministes ou non déterministes. Une affectation déterministe, ayant la forme $x := E(p, v)$, remplace la valeur de la variable x par la valeur de l'expression $E(p, v)$. Une affectation non-

déterministe peut être de deux formes :

1. $x :\in E(p, v)$ choisit arbitrairement une valeur de l'ensemble E et l'affecte à x
2. $x : |Q(p, v, x')$ affecte à x une valeur qui satisfait le prédicat Q.

Les variables sont contraintes par des conditions appelées invariants. L'exécution des événements doit conserver ces invariants. Une machine peut "voir" un ou plusieurs contextes, ce qui lui permet d'utiliser tous les éléments qui y sont définis.

Les obligations de preuve. Les obligations de preuve sont associées aux machines afin de prouver qu'elles satisfont certaines propriétés. A titre d'exemple, nous citons l'OP *INV* qui signifie *préservation d'INVariant* et qui est nécessaire pour prouver que l'exécution de chaque événement préserve les invariants.

Raffinement. Cette technique, permettant une conception *correcte par construction*, consiste à ajouter des détails graduellement en conservant les propriétés initiales du système. Le raffinement relie deux machines : une machine *abstraite* (à raffiner) et une machine *concrète* (résultat du raffinement) et est réalisé en raffinant les événements ainsi que les variables de la machine abstraite. On appellera événement abstrait (respectivement concret) un événement d'une machine abstraite (respectivement concrète) et de même pour les paramètres, les gardes, les variables, etc.

Expression des propriétés de vivacité avec *B-événementiel*. Une machine en *B-événementiel* correspond à un automate à état/transition : les états s sont définis par des vecteurs $\langle \overline{v} \rangle$ représentant les valeurs des variables v, les transitions entre les états sont définies par les événements. Un événement *evt* est dit *activable* (enabled) dans un état s lorsqu'il existe un paramètre p tel que la garde G de l'événement *evt* soit satisfaite à l'état s. Dans le cas contraire, l'événement est dit *non activable* (disabled). Une machine M est dite *bloquée* dans un état s si aucun des événements n'est activable dans cet état.

En se basant sur cette interprétation, (Hoang, Abrial, 2011) ont défini un cadre théorique permettant de raisonner sur les propriétés de vivacité des modèles en *B-événementiel*. Trois propriétés de vivacité ont été considérées dans ce contexte : *l'existence*, *la progression* et *la persistance*. Dans notre travail, nous utilisons seulement la propriété de persistance présentée dans le paragraphe suivant.

La propriété de persistance exprime qu'une propriété finit par rester définitivement vraie. Elle est formalisée pour une machine M par la formule $M \vdash \Diamond\Box P$ [1] et doit être prouvée selon la règle, $LIVE_{\Diamond\Box}$ donnée ci-dessous.

$$\frac{M \vdash \nearrow P \qquad M \vdash \circlearrowleft \neg P}{M \vdash \Diamond\Box P} \; LIVE_{\Diamond\Box}$$

1. $\Box P$, dite "*toujours P*", signifie que P est vraie dans tous les états d'une séquence. $\Diamond P$,dite "*finalement P*", signifie que la propriété P sera vraie dans un état du futur. $\neg P$ se lit *non P* et signifie que P n'est pas vraie. $M \vdash \nearrow P$ signifie que M est divergente en P. $M \vdash \circlearrowleft \neg P$ signifie que M ne se bloque pas en $\neg P$.

Dans cette règle de preuve, la première prémisse dénote le fait que M est divergente en P. Elle garantit, pour une trace ϕ de la machine M, que ϕ se termine avec une séquence infinie d'états vérifiant la propriété P (la divergence). La deuxième prémisse assure que dans le cas où ϕ est finie, elle ne vas pas se bloquer sur un état où la propriété P n'est pas vraie. Ainsi, les deux prémisses garantissent que ϕ (qu'elle soit finie ou infinie) va finir par être dans un état vérifiant la propriété P.

Afin de prouver la première prémisse avec l'outil *Rodin* [2], il est nécessaire de définir un variant $V(v)$, un ensemble fini ou une expression entière, et de démontrer les trois conditions suivantes pour tout événement *evt* de la machine M :

– lorsque la machine est dans un état vérifiant la propriété $\neg P$ et si l'événement *evt* est activable, alors $V(v)$ est non nul dans le cas où le variant est un entier naturel ou un ensemble non vide dans le cas où le variant est un ensemble fini.

– une exécution de l'événement *evt* à partir d'un état vérifiant la propriété $\neg P$ doit nécessairement décrémenter le variant $V(v)$.

– une exécution de l'événement *evt* à partir d'un état vérifiant la propriété P ne doit pas augmenter le variant $V(v)$.

2.2. La logique TLA

La logique *TLA* combine la logique temporelle et la logique des actions pour la spécification et le raisonnement sur des systèmes discrets concurrents et réactifs (Lamport, 1994). Sa syntaxe est basée sur quatre éléments :

– des constantes, des fonctions et des prédicats,

– des formules d'état définies pour le raisonnement sur les états, exprimées sur les variables et les constantes,

– des formules d'action pour raisonner sur les paires d'états (états avant et après) et

– des prédicats temporels pour raisonner sur les séquences d'états ; ces derniers sont construits à partir des éléments déjà définis et les opérateurs temporels $\Box P$ et $\Diamond P$.

Nous terminons cette description de cette logique en détaillant le concept "*d'étape de bégaiement*", les contraintes d'équité ainsi que quelques règles de preuve pour *TLA*.

Étape de bégaiement. Une étape de bégaiement pour une action A et un vecteur de variables f, a lieu lorsque soit l'action A se produit soit les valeurs des variables de f restent inchangées. Nous définissons l'opérateur de bégaiement noté $[A]_f$ comme : $[A]_f \mathrel{\widehat{=}} A \vee (f' = f)$ [3]. De manière duale, $\langle A \rangle_f$ indique que l'action A se produit et au moins une variable dans f change de valeur. $\langle A \rangle_f \mathrel{\widehat{=}} A \wedge (f' \neq f)$.

2. Rodin est une plateforme permettant le développement avec *B-événementiel* ainsi que le raffinement et la preuve. http://www.event-b.org/

3. f' représente les nouvelles valeurs du vecteur f

Contraintes d'équité. L'équité indique que si une action est activable, elle finira par s'exécuter. Deux types d'équité peuvent être distingués selon que l'action est constamment activable ou qu'elle est souvent activable :

– **l'équité faible** pour l'action A notée $WF_f(A)$; affirme que si A est constamment activable, il est alors garanti que l'action A s'exécutera toujours dans le futur.

– **l'équité forte** pour l'action A notée $SF_f(A)$; affirme que si A est souvent activable dans le futur, il est alors garanti que l'action A s'exécutera toujours dans le futur.

Formellement $WF_f(A)$ et $SF_f(A)$ sont définis comme donné ci dessous. Le prédicat $Enabled\langle A\rangle_f$ affirme que l'action A est activable.

$$WF_f(A) \widehat{=} \Diamond\Box Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f$$
$$SF_f(A) \widehat{=} \Box\Diamond Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f$$

Les règles de preuve pour *TLA*. Nous considérons les trois règles de preuve : $WF1$, $SF1$ et $LATTICE$. Dans les règles $WF1$ et $SF1$, P et Q sont des prédicats, A une action et N dénote une disjonction d'actions. P' (respectivement Q') est le prédicat P (respectivement Q) obtenu en remplaçant les variables qu'il contient par leurs nouvelles valeurs après l'exécution d'une action parmi N.

$$\frac{\begin{array}{ll} WF1.1 & P \wedge [N]_f \Rightarrow (P' \vee Q') \\ WF1.2 & P \wedge \langle N \wedge A\rangle_f \Rightarrow Q' \\ WF1.3 & P \Rightarrow Enabled\langle A\rangle_f \end{array}}{\Box[N]_f \wedge WF_f(A) \Rightarrow P \rightsquigarrow Q}\ WF1$$

$$\frac{\begin{array}{ll} SF1.1 & P \wedge [N]_f \Rightarrow (P' \vee Q') \\ SF1.2 & P \wedge \langle N \wedge A\rangle_f \Rightarrow Q' \\ SF1.3 & \Box P \wedge \Box[N]_f \Rightarrow \Diamond Enabled\langle A\rangle_f \end{array}}{\Box[N]_f \wedge SF_f(A) \Rightarrow P \rightsquigarrow Q}\ SF1$$

La règle $WF1$ donne les conditions dans lesquelles une hypothèse d'équité faible pour l'action A est suffisante pour prouver $P \rightsquigarrow Q$[4]. La condition $WF1.1$ décrit une étape faisant passer le système soit à un état vérifiant P, soit un état vérifiant Q après l'exécution d'une action parmi l'ensemble N. La condition $WF1.2$ décrit l'étape inductive où $\langle A\rangle_f$ produit un état vérifiant Q. La condition $WF1.3$ assure que $\langle A\rangle_f$ est toujours activable. La règle $SF1$ donne les conditions nécessaires pour prouver $P \rightsquigarrow Q$ en supposant une équité forte. Les deux premières conditions sont similaires à $WF1$. La troisième condition assure que $\langle A\rangle_f$ va finir par être activable.

4. l'expression $P \rightsquigarrow Q$ est équivalente à l'expression $\Box(P \Rightarrow \Diamond Q)$

La règle *LATTICE* est une règle de preuve inductive. Dans cette règle, F, G, H_c et H_d dénotent des formules TLA, S représente un ensemble donné et $\succ$ est une relation d'ordre partiel bien formé sur l'ensemble S. Informellement, cette règle signifie que pourvu qu'à partir d'un état vérifiant la formule H_c, il est possible de passer à un état vérifiant la formule G ou à un état dans lequel la formule H_d est vérifiée pour une valeur d strictement inférieure à c, il est garanti par induction que la formule G sera atteinte.

$$\frac{F \wedge (c \in S) \Rightarrow (H_c \rightsquigarrow (G \vee \exists d \in S.(c \succ d) \wedge H_d))}{F \Rightarrow ((\exists c \in S.H_c) \rightsquigarrow G)} \; LATTICE$$

Les concepts introduits dans cette section nous serviront dans la prochaine partie pour définir une formalisation des SMA auto-organisateurs et prouver leurs propriétés aussi bien au niveau local qu'au niveau global.

3. Modélisation formelle d'un SMA auto-organisateur

La modélisation formelle que nous proposons repose sur deux niveaux d'abstraction : le niveau micro et le niveau macro. Dans cette section, nous identifions les principales propriétés qui doivent être assurées lors de la conception d'un SMA auto-organisateur selon ces deux niveaux. Nous donnons aussi une stratégie de raffinement permettant de modéliser le comportement des agents du niveau micro.

3.1. Formalisation du niveau micro

La principale préoccupation à ce niveau est la conception du comportement des agents et de leurs interactions. Nous considérons que les agents interagissent via l'environnement. Ainsi, dans un premier temps, la définition de l'environnement est donnée. Dans un second temps, nous définissons ce qu'est un agent puis ce qu'est un SMA auto-organisateur.

3.1.1. Formalisation de l'environnement

Nous considérons que l'environnement est composé d'un ensemble de m éléments notés $l_1, ...l_m$. L'état de l'environnement est décrit par l'état de ses différents éléments. On note par E_{change} les actions de l'environnement ayant comme conséquences des changements sur ses éléments. Formellement, l'environnement est décrit à l'aide de l'automate $E = (SE, SE_{init}, TE, \delta E)$ avec :

– SE est l'ensemble des états de l'environnement.
$SE = \prod_{i:1..m} Sl_i$ avec Sl_i représente l'état de l'élément l_i

– $SE_{init} \in SE$ dénote l'état initial de l'environnement.

– TE est un ensemble d'étiquettes représentant les actions de l'environnement.
$TE = E_{change}$

– δE est l'ensemble des transitions possibles entre les états de l'environnement.
$\delta E \subseteq SE \times TE \times SE$

A l'aide de *B-événementiel*, la dynamique de l'environnement (E_{change}) est formalisée par un ensemble d'événements dont l'action est décrite par le prédicat *avant-après* $EnvironmentChange(l, l')$.

3.1.2. Formalisation d'un agent

D'une manière très abstraite, le comportement de chaque agent se compose de trois étapes : l'étape de perception, de décision et d'action. Nous nous référons à ces étapes par le cycle *percevoir-décider-agir*. Ainsi, un agent est caractérisé par les représentations qu'il possède de son environnement (A_{rep}), un ensemble de règles pour prendre ses décisions (A_{decide}), l'ensemble des actions qu'il peut effectuer ($A_{perform}$) et l'ensemble des opérations qui lui permettent de mettre à jour ses représentations de l'environnement ($A_{perceive}$). Un agent est aussi caractérisé par ses propriétés intrinsèques (A_{prop}), l'état de ses capteurs (A_{sens}) et l'état de ses actionneurs (A_{act}).

Plus formellement, un agent est décrit par l'automate $A = (SA, SA_{init}, TA, \delta A)$ avec :

– SA est l'ensemble des états de l'agent. $SA = A_{rep} \times A_{prop} \times A_{sens} \times A_{act}$

– $SA_{init} \in SA$ dénote l'état initial de l'agent.

– TA est un ensemble d'étiquettes formé par les opérations de perceptions, les règles de décision et les actions qu'un agent peut effectuer.
$TA = A_{perceive} \cup A_{decide} \cup A_{perform}$

– δA est l'ensemble des transitions possibles entre les états de l'agent.
$\delta A \subseteq SA \times TA \times SA$

3.1.3. Formalisation d'un SMA auto-organisateur

Un SMA auto-organisateur formé de n agents $A_1, A_2, ...A_n$ et situé dans un environnement est formalisé par l'automate $SYSTEM = (S, S_{init}, T, \delta)$, avec

– S dénote l'ensemble des états du système. Il est obtenu en fonction des états des agents et de l'environnement. $S = \prod_{i:1..n} SA_i \times SE$

– S_{init} est l'état initial du système. $S_{init} = \prod_{i:1..n} SA_{i,init} \times SE_{init}$ avec $S_{init} \in S$

– T est l'ensemble des étiquettes des transitions sur les états du système. Ces transitions sont celles des agents et celles de l'environnement.
$T = \bigcup_{i:1..n} TA_i \cup TE$

– δ dénote les transitions possibles entre les états du système. $\delta \subseteq S \times T \times S$

Avec *B-événementiel*, les caractéristiques des agents, leurs représentations de l'environnement, les états de leurs capteurs et les états de leurs actionneurs sont modélisés par des variables. Tandis que leurs décisions, leurs actions et les opérations de mise à jour de leurs perceptions sont formalisées par des événements. Ainsi, les décisions

de chaque agent a, sont formalisées par un ensemble de prédicats *avant-après* noté $DecideAct\_i(a, A_{prop}, A'_{prop})$. Un événement d'action est responsable de faire passer l'agent à l'étape de perception. Il permet alors d'activer les capteurs de l'agent. En outre, les actions d'un agent peuvent affecter ses propriétés (A_{prop}) ainsi qu'une partie de son environnement. Les actions sont modélisées par un ensemble de prédicats *avant-après* ayant la forme $PerformAct\_i(a, A_{prop} \cup A_{sens} \cup E, A'_{prop} \cup A'_{sens} \cup E')$. Enfin, l'événement qui permet à un agent de mettre à jour ses perceptions est décrit par le prédicat *avant-après* $PerceiveEnvironment(a, A_{rep}, A'_{rep})$.

L'automate $SYSTEM$, qui modélise un SMA auto-organisateur au niveau local, est traduit en la machine *MicroLevel* donnée par la figure 1.

```
machine MicroLevel
sees ContextMicro
variables   A_prop   A_act   A_sens   A_rep    E

invariants
@inv1 A_prop ∈ Agents → AgentProperties
@inv2 A_sens ∈ Agents → {true,false}
@inv3 A_act ∈ Agents → {true,false}
@inv4 A_rep ∈ Agents → RepresentationProperties
@inv5 E ∈ EnvElements → EnvElementsProperties
events
 event INITIALISATION
 event PerceiveEnvironment
 event DecideAct
 event PerformAct
 event EnvironmentChange
end
```

Figure 1. Modélisation d'un SMA auto-organisateur au niveau micro : la machine MicroLevel

Le comportement local des agents décrit précédemment est dit "correct", si les propriétés suivantes sont satisfaites.

– $LocProp1$: chaque agent fonctionne selon le cycle *Percevoir-décider-agir*.

– $LocProp2$: l'agent ne doit pas être bloqué dans l'étape de décision, c'est-à-dire que la décision doit lui permettre de réaliser une action.

$$LocProp2 \mathrel{\widehat{=}} \forall a \cdot a \in Agents \wedge DecideAct\_i(a, A_{prop}, A'_{prop}) \Rightarrow \exists PerformAct\_i \cdot G\_PerformAct\_i(PerformAct\_i(a, A_{prop} \cup A_{sens} \cup E, A'_{prop} \cup A'_{sens} \cup E'))$$

– $LocProp3$: l'agent ne doit pas être bloqué dans l'étape de perception ; c'est-à-dire que les représentations mises à jour devraient lui permettre de prendre une décision.

$$LocProp3 \mathrel{\widehat{=}} \forall a \cdot a \in Agents \wedge PerceiveEnvironment\_i(a, A_{rep}, A'_{rep}) \Rightarrow \exists DecideAct\_i \cdot G\_DecideAct\_i(DecideAct\_i(a, A_{prop}, A'_{prop}))$$

Dans les deux formules précédentes, $G\_PerformAct\_i()$ et $G\_DecideAct\_i()$ dénotent respectivement les gardes des événements $PerformAct\_i$ et $DecideAct\_i$.

3.2. Formalisation du niveau macro

Au niveau macro, nous focalisons sur la modélisation de l'observation de l'évolution de l'état macroscopique du système en fonction des actions locales des agents, de leurs interactions et des changements de l'environnement. Cette modélisation est une base pour prouver formellement des propriétés globales. L'état macroscopique du système est décrit par un agrégat des états de ses éléments (les agents et l'environnement). Ainsi, le nombre d'agents se trouvant dans un état particulier ou le nombre d'agents exhibant un comportement bien déterminé peuvent décrire l'état macroscopique du système.

Le système peut être soit dans un état fonctionnellement adéquat qui qualifie une situation dans laquelle il assure la fonction pour laquelle il a été conçu, soit dans un état qui nécessite l'emploi de ses mécanismes d'auto-organisation pour faire face aux perturbations et se remettre à nouveau dans un état fonctionnellement adéquat. La figure 2 résume les états du cycle de vie d'un SMA auto-organisateur observé par un observateur externe. On note par $S_{adequate}$ un état fonctionnellement adéquat dans

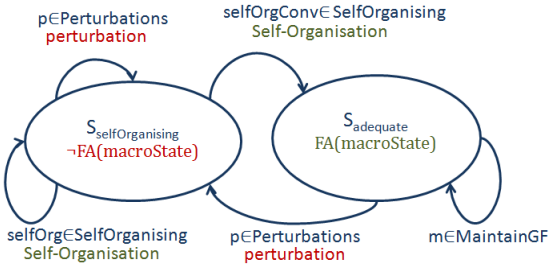

Figure 2. Une formalisation abstraite du comportement du système au niveau macro

lequel le système peut se trouver dès son initialisation ou auquel il doit converger lorsqu'il n'est soumis à aucune perturbation. On note par $FA(macroState)$ le prédicat défini en fonction de l'état macroscopique du système $macroState$ et décrivant un état fonctionnellement adéquat. Dès que la fonctionnalité du système est perturbée, il passe à l'état $S_{selfOrganising}$ dans lequel il emploie ses mécanismes d'auto-organisation pour assurer à nouveau sa fonction. On note par GS l'ensemble des états du système au niveau macro. $GS = S_{adequate} \cup S_{selfOrganising}$

Les transitions observées entre ces différents états sont dues à des changements pouvant être classés parmi les ensembles de changements suivants :

– $Maintain_{GF}$ est formé des changements n'ayant aucun effet sur la fonctionnalité adéquate du système.

– *Perturbations* est l'ensemble des changements qui perturbent le fonctionnement du système et l'empêchent d'assurer sa fonction.

– *SelfOrganising* regroupe les actions que le système peut entreprendre afin de retrouver un état fonctionnellement adéquat.

On note par GT l'ensemble de toutes ces transitions.

$$GT = Maintain_{GF} \cup Perturbations \cup SelfOrganising.$$

Le comportement observé du système est défini par une séquence (pouvant être infinie) alternant des états et des actions $gs_0 \xrightarrow{gt_1} gs_1 \xrightarrow{gt_2} gs_2...$ où pour tout $i > 0$, $gt_i \in GT$ tel que $(gs_{i-1}, \xrightarrow{gt_i}, gs_i) \in G\delta$. On note par $\varepsilon(GS)$ l'ensemble de toutes les traces observables du système. On note par $state(\epsilon, i)$ l'état observable du système à l'instant i dans la trace $\epsilon \in \varepsilon(GS)$. Le comportement observé est ainsi modélisé par la machine *MacroLevel* donnée par le figure 3.

```
machine MacroLevel
variables macroState     FA
invariants
@inv1 FA ∈ STATES → BOOL
@inv2 macroState∈STATES
events
 event INITIALISATION
 event m
 event selfOrg
 event selfOrgConv
 event p
 event observer
    where
      @grd1 FA(macroState)=TRUE
 end
end
```

Figure 3. Modélisation d'un SMA auto-organisateur au niveau macro : la machine MacroLevel

Les définitions données ci-dessus vont servir de base pour spécifier formellement les propriétés de convergence et de résilience. Ces deux propriétés sont définies dans les paragraphes qui suivent.

3.2.1. La convergence du SMA

La convergence (Serugendo, 2009) indique la capacité du système à atteindre son objectif en l'absence de perturbations. Le système converge soit lorsque dès l'initialisation il se trouve dans un état fonctionnellement adéquat soit on garantit qu'il arrive à atteindre un état fonctionnellement adéquat après un certain nombre de transitions. Formellement, la convergence se traduit par la formule suivante :

$$GS_{init} \in S_{adequate} \vee (\forall \epsilon \in \varepsilon(SYSTEM_{MACRO}) \cdot \exists i \cdot state(\epsilon, i) \in S_{adequate}).$$

En utilisant la logique de prédicats et les opérateurs temporels, cette propriété est exprimée par la formule suivante :

$$MacroLevel \vdash \Diamond\Box\ FA(macroState)$$

La preuve de cette formule se fait selon la règle $LIVE_{\Diamond\Box}$ présentée dans la section 2.1 est décrite comme suit :

$$\frac{MacroLevel \vdash \nearrow\ FA(macroState) \quad MacroLevel \vdash \circlearrowleft\ \neg FA(macroState)}{MacroLevel \vdash \Diamond\Box\ FA(macroState)}\ LIVE_{\Diamond\Box}$$

La première prémisse ($MacroLevel \vdash \nearrow\ FA(macroState)$) garantit que toute trace d'exécution de la machine *MacroLevel* se termine par une séquence infinie d'états satisfaisant le prédicat $FA(macroState)$. La preuve de cette prémisse se fait en garantissant les trois conditions suivantes :

– définir le variant *convProg* dans la machine *MacroLevel*. La variable *convProg* modélise la progression du système vers un état fonctionnellement adéquat et est obtenue en fonction de l'état macroscopique du système. Ici, nous supposons que la variable *convProg* est un entier naturel, mais elle peut être un ensemble fini.

```
variables macroState
invariants
@inv3 convProg ∈ STATES → ℕ
variant convProg (macroState)
```

Figure 4. Définition du variant convProg de la machine MacroLevel.

– prouver que l'événement *selfOrgConv* réduit à chaque exécution le variant *convProg*.

– prouver que les événements *m* et *observer* n'incrémentent pas le variant *convProg* à chaque exécution.

La deuxième prémisse ($MacroLevel \vdash \circlearrowleft\ \neg FA(macroState)$) assure que la machine *MacroLevel* ne se bloque pas en un état dans lequel le prédicat $FA(macroState)$ n'est pas vérifié. Prouver cette prémisse revient à ajouter et prouver le théorème de non blocage (figure 5) à la machine *MacroLevel*.

3.2.2. La résilience du SMA

La résilience (Serugendo, 2009) décrit la capacité du système à s'adapter aux changements et aux perturbations qui peuvent avoir lieu. L'analyse de la résilience permet d'évaluer la capacité des mécanismes d'auto-organisation à rétablir l'état du système après des perturbations sans détecter explicitement une erreur.

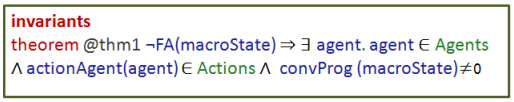

Figure 5. Le théorème de non blocage de la machine MacroLevel dans l'état $\neg FA(macroState)$

On note par p une perturbation provenant de l'environnement ou des agents et provoquant un ensemble ψ de transitions ($\psi \subseteq GS \times p \times GS$). Cette perturbation fait passer le système d'un état fonctionnellement adéquat vers un état d'auto-organisation. Le système est dit résilient à la perturbation p s'il est capable de retrouver un état fonctionnellement adéquat après cette perturbation. On note par $\varepsilon(SYSTEM_{MACRO}, gs)$ les traces observables du système à partir de l'état gs.

Formellement, le système $SYSTEM_{MACRO} = (GS, GS_{init}, GT, G\delta)$ est dit résilient par rapport à la perturbation p si pour toute trace $\epsilon \in \varepsilon(SYSTEM_{MACRO})$ tel qu'il existe $i \geqslant 0$ pour lequel $state(\epsilon, i) = gs$ et pour tout état d'auto-organisation gs' vérifiant $(gs, p, gs') \in \psi$, il vérifie la formule suivante :

$$\forall \epsilon' \in \varepsilon(SYSTEM_{MACRO}, gs') \cdot \exists j > 0 \cdot state(\epsilon', j) \in S_{adequate}.$$

Cette propriété est exprimée à l'aide de la logique de prédicats et des opérateurs temporels par la formule suivante :

$$MAS \vdash \Box(\neg FA(macroState) \Rightarrow \Diamond FA(macroState)). \quad (1)$$

La formule 1 peut s'écrire au moyen de l'opérateur *leads to* (noté $\rightsquigarrow$) comme suit :

$$MAS \vdash \neg FA(macroState) \rightsquigarrow FA(macroState). \quad (2)$$

En supposant que la preuve de la formule 2 se fait à l'aide de la règle $WF1$ et en considérant les éléments suivants :

- $N = selfOrg \vee selfOrgConv \vee p \vee observer \vee m$,
- $f = convProg(macroState)$,

la règle de preuve $WF1$ est réécrite comme suit :

$$\frac{\begin{array}{ll} WF1.1 & \neg FA(macroState) \wedge [N]_f \Rightarrow (\neg FA'(macroState) \vee FA'(macroState)) \\ WF1.2 & \neg FA(macroState) \wedge \langle N \wedge selfOrgConv\rangle_f \Rightarrow FA'(macroState) \\ WF1.3 & \neg FA(macroState) \Rightarrow Enabled\langle selfOrgConv\rangle_f \end{array}}{\Box[N]_f \wedge WF_f(selfOrgConv) \Rightarrow \neg FA(macroState) \rightsquigarrow FA(macroState)}$$

La condition $WF_f(selfOrgConv)$ est formulée comme suit :

$$WF_f(selfOrgConv) \widehat{=} \Diamond\Box Enabled\langle selfOrgConv\rangle_f \Rightarrow \Box\Diamond\langle selfOrgConv\rangle_f \quad (3)$$

L'équation 3 représente une hypothèse d'équité faible sur l'action de l'événement $selfOrgConv$ et suppose qu'il est continuellement activable. En l'absence de perturbations, cette condition est vraie. Mais lorsque le système est soumis à des changements, cette supposition n'est plus valide. Par conséquent, la preuve de la résilience nécessite l'application de la règle $SF1$ basée sur une hypothèse d'équité forte sur l'action de l'événement $selfOrgConv$.

La règle de preuve associée est la suivante :

$$\frac{\begin{array}{ll} SF1.1 & \neg FA(macroState) \wedge [N]_f \Rightarrow (\neg FA'(macroState) \vee FA'(macroState)) \\ SF1.2 & \neg FA(macroState) \wedge \langle N \wedge selfOrgConv\rangle_f \Rightarrow FA'(macroState) \\ SF1.3 & \Box\neg FA(macroState) \wedge \Box[N]_f \Rightarrow \Diamond Enabled\langle selfOrgConv\rangle_f \end{array}}{\Box[N]_f \wedge SF_f(selfOrgConv) \Rightarrow P \rightsquigarrow FA'(macroState)}$$

La condition $SF_f(selfOrgConv)$ est une hypothèse d'équité forte sur l'action de l'événement $selfOrgConv$. Elle est exprimée à l'aide des opérateurs temporels comme suit :

$$SF_f(selfOrgConv) \widehat{=} \Box\Diamond Enabled\langle selfOrgConv\rangle_f \Rightarrow \Box\Diamond\langle selfOrgConv\rangle_f \quad (4)$$

Grâce à la formule 4, on démontre que l'action de l'événement $selfOrgConv$ va toujours finir par s'activer lorsque cet événement est infiniment souvent activable (il n'est pas continuellement activable).

3.3. Stratégie de raffinement

Le développement formel d'un SMA auto-organisateur commence par une machine très abstraite représentant le système comme un ensemble d'agents fonctionnant selon le cycle *Percevoir-décider-agir*. Chacun des événements du modèle initial $Perceive$, $Decide$ et $Perform$ (figure 6) correspond à une étape dans le cycle de vie de l'agent et permet à un agent de passer d'une étape à l'étape suivante. Ce modèle abstrait vérifie que le comportement de chaque agent est conforme au cycle *Percevoir-Décider-Agir* (propriété $LocProp1$). Il constitue le modèle initial d'une série de trois étapes de raffinement.

Le premier raffinement consiste en la définition des différentes actions réalisées par les agents. Ainsi, le raffinement de la machine $Agents0$ par la machine $Agents1$ se fait en divisant l'événement $Perform$ en plusieurs événements dont chacun représente une action. Ce raffinement assure $LocProp2$ (pas de blocage dans l'étape de décision). La figure 7 est un extrait de la machine $Agents1$ et modélise de manière générique l'action d'un agent.

Dans la deuxième étape de raffinement, nous détaillons les événements correspondants aux décisions qu'un agent peut prendre et nous décrivons les règles permettant à l'agent de décider. Nous introduisons également les actionneurs des agents. En utilisant des témoins (*witness*), nous relions les actions introduites dans le raffinement précédent avec les décisions correspondantes définies dans cette étape de raffinement.

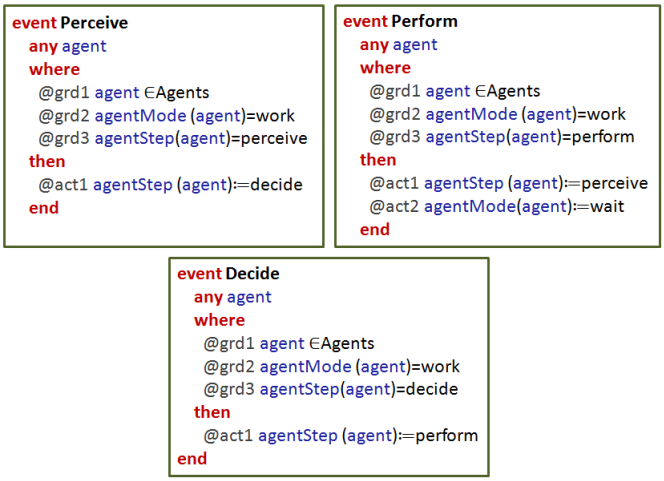

Figure 6. Les événements Perceive, Decide et Perform de la machine initiale Agents0

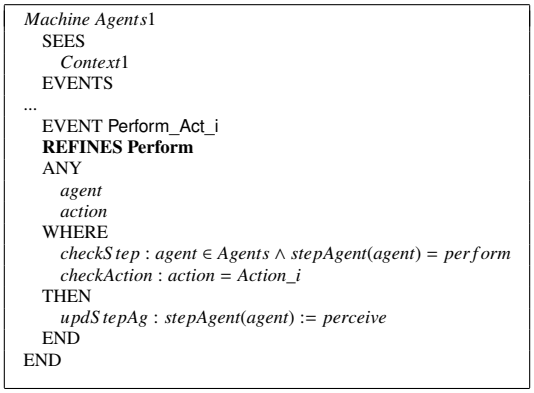

Figure 7. Raffinement de l'événement Perform dans la machine Agents1

La figure 8 décrit de manière générale la manière avec laquelle les événements de décision et d'action sont raffinés.

Dans le troisième raffinement (figure 9), nous introduisons les données concernant l'état du système (*ActualSysState*) ainsi que les représentations de l'agent de son environnement (*rep*) et les capteurs des agents (*sensors*). A ce niveau de raffinement, l'événement *Perceive* est raffiné pour permettre la mise à jour des représentations des agents. Les différents événements liés aux décisions et actions sont aussi raffinés. Ce raffinement garantit la propriété *LocProp*3 (pas de blocage dans l'étape de perception).

EVENT Decide_Act_i **REFINES Decide**
ANY
agent
WHERE
checkStep : $agent \in Agents \wedge stepAgent(agent) = decide$
THEN
updStepAg : $stepAgent(agent) := perform$
updActAg : $actuators(agent) := enabled$
END
EVENT Perform_Act_i **REFINES Perform_Act_i**
ANY
agent
WHERE
checkStep : $agent \in Agents \wedge stepAgent(agent) = perform$
checkActuator : $actuators(agent) = enabled$
WITH
action : $action = Act\_Action\_i \Leftrightarrow actuators(agent) = enabled$
THEN
updStepAg : $stepAgent(agent) := perceive$
END

*Figure 8. Raffinement des événements de décision et d'action dans la machine Agents*2

La figure 9 représente un extrait de la machine *Agents*3 qui raffine *Agents*2. L'invariant *gluInvSensorsPercept* est un invaraint de collage qui permet de relier l'étape de perception de l'agent à l'activation de ses capteurs. Dans le contexte *Context*3, on définit l'aptitude *AbilityToPerceive* (voir son utilisation dans l'événement *Perceive* dans la figure 9) permettant à un agent de déterminer l'état de son environnement local en fonction de l'état global du système.

*Machine Agents*3
SEES
*Context*3
VARIABLES
sensors
rep
ActualSysState
INVARIANTS
defSensorAg : $sensors \in Agents \rightarrow Activation$
defRepAg : $rep \in Agents \rightarrow Value$
defGlobalStateSys : $ActualSysState \in SysStates$
gluInvSensorsPercept : $\forall ag \cdot ag \in Agents \Rightarrow (stepAgent(ag) = perceive \Leftrightarrow sensors(ag) = enabled)$
EVENTS
EVENT PerceiveEnvironment
REFINES Perceive
ANY
agent
WHERE
grdAgent : $agent \in Agents$
grdChekSensors : $sensors(agent) = enabled$
THEN
updStepAg : $stepAgent(agent) := decide$
updRepAg : $rep(agent) := AbilityToPerceive(ActualSysState)$
updSensorAg : $sensors := disabled$
END
END

*Figure 9. Raffinement de l'événement de perception dans la machine Agents*3

4. Application aux fourmis fourrageuses

L'exemple que nous présentons est une formalisation des comportements d'une colonie de fourmis fourrageuses (Topin *et al.*, 1999). Le système est composé de plusieurs fourmis qui se déplacent à la recherche de nourriture dans un environnement qui est un ensemble de cellules reliées entre elles. L'objectif principal des fourmis est de ramener toute la nourriture placée dans l'environnement vers leur nid. Elles n'ont pas d'information sur les emplacements des sources de nourriture, mais elles sont capables de percevoir la nourriture qui est à l'intérieur de leur champ de perception. Les fourmis interagissent entre elles via l'environnement en déposant de la phéromone.

Concernant les propriétés globales, nous nous intéressons à la convergence des fourmis en ramenant toute la nourriture au nid et à leur capacité à détecter une source de nourriture rajoutée à l'environnement.

4.1. Formalisation du comportement local des fourmis

Dans cette section, nous appliquons la stratégie de raffinement décrite dans la section 3.3. Ces étapes de raffinements permettent de dériver le comportement local des fourmis et de vérifier les propriétés du niveau micro.

Modèle abstrait : le modèle initial, appelé *Ants*0, est identique à la machine *Agents*0. Il représente un ensemble de fourmis dont chacune fonctionne selon le cycle *Percevoir-Décider-Agir*.

Premier raffinement : le premier raffinement de *Ants*0 se fait conformément au raffinement de *Agents*0 et donne lieu à la machine (*Ants*1). Ainsi, l'événement $Perform$ est raffiné par les quatre événements suivants :

1. $PerformAntsMove$: se déplacer dans l'environnement,
2. $PerformAntsDropPheromone$: retourner au nid en déposant de la phéromone,
3. $PerformAntsHarvestFood$: récolter de la nourriture et
4. $PerformAntsDropFood$: déposer la nourriture dans le nid

Second raffinement : le second raffinement se fait conformément au raffinement de *Agents*1 et donne lieu à la machine *Ants*2. Ainsi, l'événement *Decide* est raffiné en le divisant en cinq événements :

1. $DecideAntsMoveExplore$: décider d'explorer l'environnement,
2. $DecideAntsMoveBack$: décider de retourner au nid,
3. $DecAntsMoveBackDrop$: décider de retourner au nid en déposant de la phéromone,
4. $DecideAntsHarvestFood$: décider de récolter la nourriture et
5. $DecideAntsDropFood$: décider de déposer la nourriture dans le nid

Nous ajoutons également les actionneurs d'une fourmi : ces pattes (*paw*), la glande qui secrète de la phéromone (*exocrinGland*) et les mandibules lui permettant la récolte

de la nourriture (*mandible*). Nous introduisons aussi la caractéristique *nextLocation* contenant la prochaine cellule vers laquelle la fourmi va se déplacer et est le résultat du processus de décision.

Troisième raffinement : ce troisième raffinement se fait conformément au raffinement de $Agents2$ et donne lieu à la machine $Ants3$. Les représentations des fourmis de leur environnement sont introduites. Chaque fourmi peut percevoir la nourriture ($food$) ainsi que la phéromone (*pheromone*). Nous introduisons également la variable *DePhero* modélisant la distribution de la phéromone dans l'environnement. L'événement *Perceive* est raffiné (ci-dessous) en ajoutant les actions nécessaires pour mettre à jour les représentations d'une fourmi.

EVENT PerceiveAntsEnvironment
REFINES Perceive
ANY
ant, loc, fp, php
WHERE
$grd1 : ant \in Ants \wedge stepAgent(ant) = perceive$
$grd2 : loc = currentLoc(ant)$
$grd3 : fp \in Locations \times Locations \pfun \mathbb{N}$
$grd4 : fp = FPerc(QuantityFood)$
$grd5 : php \in Locations \times Locations \pfun \mathbb{N}$
$grd6 : php = PhPerc(DePhero)$
THEN
$act1 : stepAgentCycle(ant) := decide$
$act2 : food(ant) := \{loc \mapsto fp(loc \mapsto dir) | dir \in Next(loc)\}$
$act3 : pheromone(ant) := \{loc \mapsto php(loc \mapsto dir) | dir \in Next(loc)\}$
END

$FPerc$ (garde $grd4$) et $PhPerc$ (garde $grd6$) modélisent la capacité d'une fourmi à sentir respectivement la nourriture et la phéromone situées dans son champ de perception. L'événement $DecideAntsMoveExplore$ est divisé en trois événements :

1. $DecideAntsMoveRandom$: décider d'aller dans une direction aléatoirement,
2. $DecideAntsMoveFollowFood$: décider d'aller dans la direction contenant le plus de nourriture et
3. $DecideAntsMoveFollowPheromone$: décider d'aller dans la direction contenant le plus de phéromone.

Ce raffinement garantit la propriété $LocProp3$ pour la décision de mouvement. L'événement $PerformAntsMove$ est également raffiné pour tenir compte de ces différentes décisions.

4.2. Formalisation des propriétés globales

Les trois étapes de raffinement décrites dans la section précédente nous ont permis de spécifier un comportement local correct pour les fourmis. Dans cette section, nous nous intéressons à prouver que le comportement modélisé permet d'atteindre les propriétés globales souhaitées.

4.3. Preuve de la convergence

La convergence des fourmis se traduit par leur capacité à récolter toute la nourriture et la ramener au nid. La preuve de cette propriété nécessite la définition de l'événement observateur et la preuve de la terminaison de tous les événements d'action.

4.3.1. L'événement observateur

L'événement observateur est le seul événement responsable de détecter si le système a réussi à réaliser sa fonctionnalité globale. Il s'agit d'un événement particulier sans action et dont la garde décrit l'état du système lorsqu'il atteint sa fonctionnalité globale. Pour l'exemple des fourmis fourrageuses, nous avons considéré que la fonctionnalité globale du système est de ramener la nourriture, initialement dispersée dans l'environnement, vers le nid. Ainsi, la garde de l'événement observateur appelé $AllFoodAtNest$ est décrite par l'expression suivante :

$$\forall loc \cdot loc \in Locations \setminus \{Nest\} \Rightarrow QuantityFood(loc) = 0 \wedge TotalFood(InitFoodDist \mapsto Locations) = QuantityFood(Nest)$$

Dans cette expression, la fonction $QuantityFood$ donne pour chaque position de l'environnement la quantité de nourriture qu'elle contient. La fonction $TotalFood$ permet de calculer la somme des quantité de nourriture dans l'environnement.

4.3.2. La terminaison des événements d'action

La preuve de la terminaison d'un événement nécessite la définition d'un variant [5]. Nous définissons, dans le tableau 1, les variants utilisés pour la preuve de la terminaison des événements d'action. Une fois les variants définis, l'étape suivante consiste à prouver que chacun de ces événements, en s'exécutant, décrémente le variant qui lui correspond. Pour les événements :

- $PerformAntsDropFood$, $PerformAntsHarvestFood$ et
- $PerformAntsDropPheromone$

la preuve est triviale. Elle nécessite le renforcement de leurs gardes et l'ajout des actions qui décrémentent le variant pour montrer leur terminaison. Dans le cas des événements :

- $PerformAntsMoveBack$, $PerformAntsMoveExploreFollowFood$ et
- $PerformAntsMoveExploreFollowPheromone$

la preuve nécessite l'ajout de nouveaux axiomes.

Pour l'événement $PerformAntsMoveExploreRandom$, des hypothèses d'équité forte ont été nécessaires pour montrer qu'il décrémente son variant. La preuve relative à ce dernier événement est faite à l'aide de la logique *TLA*.

5. une expression numérique entière ou un ensemble fini

Tableau 1. Définition des variants nécessaires pour montrer la terminaison des événements d'action

Evénement	Variant
PerformAntsDropFood	V1: l'ensemble des fourmis déposant de la nourriture dans le nid
PerformAntsHarvestFood	V2: la somme totale de nourriture dans l'environnement sauf le nid
PerformAntsDropPheromone	V3: l'ensemble des fourmis déposant de la phéromone
PerformAntsMoveBack	V4: la somme des distances entre les positions des fourmis retournant au nid et le nid
PerformAntsMoveExploreFollowFood	V5: la somme des distances entre les positions des fourmis en quête d'une source de nourriture et la position de la source de nourriture en question
PerformAntsMoveExploreFollowPheromone	V6: la somme des distances entre les positions des fourmis poursuivant de la phéromone et la position de la phéromone en question
PerformAntsMoveExploreRandom	V7: l'ensemble des fourmis faisant un déplacement aléatoire

4.4. Preuve de la résilience

Les modèles obtenus vérifient les propriétés locales de non blocage ainsi que les contraintes locales issues de la description de l'étude de cas. Ces modèles permettent également la convergence vers un état dans lequel toute la nourriture est récoltée. Cette convergence a été prouvée en supposant que l'environnement reste inchangé. Ainsi, les événements relatifs à l'apparition de nouvelles sources de nourriture ou l'apparition de nouveaux obstacles n'ont pas été pris en compte.

Dans cette section, nous nous intéressons à la réaction des fourmis lorsque de nouvelles sources de nourriture sont ajoutées dans l'environnement. Nous voulons prouver que les fourmis sont capables de la détecter. Nous exprimons cette propriété à l'aide de la logique temporelle par la formule 5.

$$ResNewFood \mathrel{\widehat{=}} \Box(\forall loc.(loc \in NewFoodLocations \wedge \exists ant.comeBackAnt(ant) = FALSE \wedge nextLocation(ant) = loc) \Rightarrow \Diamond(loc \in DetectedFoodLocations)) \quad (5)$$

Dans la formule 5, la variable $NewFoodLocations$ désigne l'ensemble des positions dans lesquelles la nourriture est rajoutée. La variable $DetectedFoodLocations$ désigne l'ensemble des sources de nourriture détectée.

La formule 5 signifie que toute source de nourriture (loc) ajoutée dans l'environnement ($loc \in NewFoodLocations$) finira par être détectée.

La condition ($\exists ant.comeBackAnt(ant) = FALSE \wedge nextLocation(ant) = loc$) spécifie qu'il existe une fourmi (ant) en mode exploration ($comeBackAnt(ant) = FALSE$) qui a détecté une source de nourriture en faisant un déplacement dans sa direction. La preuve de cette formule nécessite :

– la modélisation de la perturbation en raffinant l'événement *EnvironmentChange* par l'événement $EnvironmentChangeAddFood$ pour modéliser l'ajout de nourriture dans l'environnement. L'événement concret est donné par la figure 10. L'action $act3$

EVENT EnvironmentChangeAddFood
REFINES EnvironmentChange
ANY
  $newFood$
WHERE
  $grd1 : \forall agent \cdot agent \in Ants \Rightarrow antMode(agent) = wait$
  $grd2 : newFood \subseteq (\{loc \cdot loc \in Locations \wedge QuantityFood(loc) = 0 | loc\} \setminus \{Nest\})$
THEN
  $act1 : antMode : |antMode' = Ants \times \{work\}$
  $act2 : NewFoodLocations := NewFoodLocations \cup newFood$
  $act3 : QuantityFood : |QuantityFood'(newFood) \in 1..QuantityFoodMax \wedge$
    $\forall loc \in Locations \setminus \{newFood\} \Rightarrow QuantityFood'(loc) = QuantityFood(loc)$
END

Figure 10. L'événement EnvironmentChangeAddFood

permet de modifier la quantité de nourriture dans la position concernée par l'ajout en conservant la quantité de nourriture des autres positions.

– la formalisation des actions permettant au système de "corriger" la perturbation par les événements :

- $PerformAntsMoveExploreFollowFood$,
- $PerformAntsMoveExploreFollowPheromone$ et
- $PerformAntsMoveExploreFollowRandom$

Une réflexion sur la manière avec laquelle les fourmis peuvent découvrir de nouvelles sources de nourriture, nous amène à doter les fourmis d'un comportement coopératif leur dictant d'éviter d'aller dans les directions qui contiennent beaucoup de fourmis même si elles contiennent de la nourriture. Ainsi, nous enrichissons l'ensemble des événements, modélisant les actions d'exploration de l'environnement, par l'événement $PerformAntsMoveExploreAvoidCompetition$. Nous jugeons que le rôle de cet événement est primordial pour la résilience du système.

– l'ajout de l'événement $PerformAntsMoveExploreFollowFoodFirstTime$ qui modélise la détection d'une source de nourriture.

– l'introduction de l'événement *ResilienceObserver* qui est un événement sans action dont la garde décrit l'état dans lequel toute la nourriture ajoutée est détectée. Cette garde est donnée par l'expression $NewFoodLocations = \emptyset$.

Les éléments du modèle étant définis, nous passons à la preuve de la propriété *ResNewFood*. Nous considérons les deux prédicats P et $Q_{Detected}$ définis en fonction de la cardinalité de l'ensemble *NewFoodLocations* définis comme suit :

$P \widehat{=} card(NewFoodLocations) = n + 1$, $Q_{Detected} \widehat{=} card(NewFoodLocations) = n$

et nous souhaitons prouver la formule $P \rightsquigarrow Q_{Detected}$.

On définit N et $A_{Detected}$ par :

$N \widehat{=} PerformAntsMoveExploreFollowFoodFirstTime \vee$
$PerformAntsMoveExploreFollowFood \vee$
$PerformAntsMoveExploreAvoidCompetition \vee$
$PerformAntsMoveExploreFollowPheromone \vee$
$PerformAntsMoveExploreRandom \vee$
$EnvironmentChangeAddFood \vee ResilienceOberver$
et
$A_{Detected} \widehat{=} PerformAntsMoveExploreFollowFoodFirstTime.$

En appliquant $SF1$, il est possible de prouver $P \rightsquigarrow Q_{Detected}$:

$$\frac{\begin{array}{ll} SF1.1 & P \wedge [N]_{NewFoodLocations} \Rightarrow (P' \vee Q'_{Detected}) \\ SF1.2 & P \wedge \langle N \wedge A_{Detected} \rangle_{NewFoodLocations} \Rightarrow Q'_{Detected}) \\ SF1.3 & \Box P \wedge \Box[N]_{NewFoodLocations} \Rightarrow \Diamond Enabled\langle A_{Detected} \rangle_{NewFoodLocations} \end{array}}{SF1.H \quad \Box[N]_{NewFoodLocations} \wedge SF_{NewFoodLocations}(A_{Detected}) \Rightarrow P \rightsquigarrow Q_{Detected}}$$

La condition $SF1.1$ décrit une étape durant laquelle le système progresse vers un état vérifiant le prédicat P ou vers un état vérifiant l'état $Q_{Detected}$. La condition $SF1.2$ décrit une étape d'induction durant laquelle l'action $\langle A_{Detected} \rangle_{NewFoodLocations}$ (la détection d'une position contenant de la nourriture ajoutée) donne lieu à un état vérifiant $Q_{Detected}$. La condition $SF1.3$ assure que $\langle A_{Detected} \rangle_{NewFoodLocations}$ finira par être activée.

En supposant que les opérations d'ajout de nourriture dans l'environnement finiront par être arrêtées et en appliquant la règle *LATTICE*, on peut prouver que la cardinalité de l'ensemble *NewFoodLocations* finira par s'annuler ce qui activera l'événement *ResilienceObserver*.

5. Conclusion et perspectives

Nous avons présenté dans cet article une modélisation formelle des SMA auto-organisateurs au moyen de *B-événementiel*. Dans notre formalisation, nous avons considéré le système selon deux niveaux d'abstraction : micro et macro. Cette abstraction a permis de concentrer les efforts de développement sur un aspect particulier du système. Ainsi, nous nous sommes intéressés à spécifier le comportement local des

agents au niveau micro. Cette spécification a été guidée par des étapes de raffinement garantissant la correction, absence de blocage, du comportement généré. Cette stratégie de raffinement a été ensuite étendue afin de prouver des propriétés globales de convergence et de résilience en ayant recours en plus des obligations de preuve à la logique TLA. Notre formalisation a été appliquée à l'étude de cas des fourmis fourrageuses. Nous avons réussi à prouver que toute la nourriture dispersées dans l'environnement sera récoltée et que les fourmis sont capables de détecter et de récolter une nouvelle source de nourriture. Cependant, la preuve de propriétés relatives à l'optimisation comme "Les fourmis sont capables de trouver le plus court chemin entre une source de nourriture et le nid" et celles ayant un aspect quantitatif restent toujours un défi. Une autre limite à signaler est que nous avons supposé que la fonctionnalité émergente du système est connue à l'avance pour pouvoir faire la preuve, chose qui n'est pas toujours vraie avec les SMA auto-organisateurs.

Nos ambitions pour des travaux futurs se résument dans les trois points suivants :

– la définition de patrons de raffinement formels à l'aide du formalisme *B-événementiel*. Notre but est de fournir aux concepteurs de SMA auto-organisateurs des outils leur permettant l'utilisation des techniques formelles pour la preuve de manière plus intuitive dans le processus de développement.

– l'intégration de la formalisation proposée dans des méthodes de développement de SMA auto-organisateurs pour assurer une vérification formelle aux étapes préliminaires du cycle de développement. Cette intégration peut être réalisée par les techniques de l'ingénierie dirigée par les modèles.

– l'introduction des mécanismes d'auto-organisation basés sur la coopération en particulier et l'analyse de leurs impacts sur la résilience du système. Dans le cas des fourmis fourrageuses, l'objectif est d'analyser la capacité des fourmis à améliorer leur rapidité de convergence grâce à une attitude coopérative. Cet objectif peut être atteint en utilisant une approche probabiliste couplée avec *B-événementiel*.

Bibliographie

Abrial J. (2010). *Modeling in Event-B - system and software engineering*. Cambridge University Press. Consulté sur http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521895569

Casadei M., Viroli M. (2009). Using probabilistic model checking and simulation for designing self-organizing systems. In *Proceedings of the 2009 acm symposium on applied computing*, p. 2103–2104. New York, NY, USA, ACM. Consulté sur http://doi.acm.org/10.1145/1529282.1529747

Di Marzo Serugendo G., Gleizes M.-P., Karageorgos A. (2005, juin). Self-organization in multi-agent systems. In *Knowl. eng. rev.*, vol. 20, p. 165–189. New York, NY, USA, Cambridge University Press. Consulté sur http://dx.doi.org/10.1017/S0269888905000494

Gardelli L., Viroli M., Omicini A. (2006). Exploring the dynamics of self-organising systems with stochastic π-calculus: Detecting abnormal behaviour in MAS. In *In mas. in: Fifth*

international symposium from agent theory to agent implementation (at2ai5.

Georgé J.-P. (2004). *Résolution de problèmes par Ã©mergence, Etude d'un Environnement de Programmation émergente*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France. Consulté sur ftp://ftp.irit.fr/IRIT/SMAC/DOCUMENTS/RAPPORTS/TheseJPGeorge_0704.pdf

Gleizes M.-P. (2012). Self-adaptive Complex Systems (regular paper). In M. Cossentino, M. Kaisers, K. Tuyls, G. Weiss (Eds.), *European Workshop on Multi-Agent Systems (EUMAS), Maastricht, The Netherlands, 13/11/2011-16/11/2011*, vol. 7541, p. 114–128. http://www.springerlink.com/, Springer-Verlag. Consulté sur http://www.springer.com/computer/ai/book/978-3-642-34798-6

Hilaire V., Gruer P., Koukam A., Simonin O. (2008). Formal driven prototyping approach for multiagent systems. *IJAOSE*, vol. 2, n° 2, p. 246–266. Consulté sur http://dx.doi.org/10.1504/IJAOSE.2008.017317

Hoang T. S., Abrial J. (2011). Reasoning about liveness properties in Event-B. In *Formal methods and software engineering - 13th international conference on formal engineering methods, ICFEM 2011, durham, uk, october 26-28, 2011. proceedings*, p. 456–471. Consulté sur http://dx.doi.org/10.1007/978-3-642-24559-6_31

Konur S., Dixon C., Fisher M. (2012, février). Analysing robot swarm behaviour via probabilistic model checking. *Robot. Auton. Syst.*, vol. 60, n° 2, p. 199–213. Consulté sur http://dx.doi.org/10.1016/j.robot.2011.10.005

Lamport L. (1994). The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, vol. 16, n° 3, p. 872–923. Consulté sur http://doi.acm.org/10.1145/177492.177726

Méry D., Poppleton M. (2013). Formal modelling and verification of population protocols. In *Integrated formal methods, 10th international conference, IFM 2013, turku, finland, june 10-14, 2013. proceedings*, p. 208–222. Consulté sur http://dx.doi.org/10.1007/978-3-642-38613-8_15

Pereverzeva I., Troubitsyna E., Laibinis L. (2012). Development of fault tolerant MAS with cooperative error recovery by refinement in Event-B. *CoRR*, vol. abs/1210.7035. Consulté sur http://arxiv.org/abs/1210.7035

Serugendo G. D. M. (2009). Robustness and dependability of self-organizing systems - A safety engineering perspective. In *Stabilization, safety, and security of distributed systems, 11th international symposium, SSS 2009, lyon, france, november 3-6, 2009. proceedings*, p. 254–268. Consulté sur http://dx.doi.org/10.1007/978-3-642-05118-0_18

Simonin O., Lanoix A., Scheuer A., Charpillet F. (2011, novembre). Specifying in B the Influence/Reaction Model to Study Situated MAS: Application to vehicles platooning. In *V2CS : First International workshop on Verification and Validation of multi-agent models for complex systems*, p. 15 pages. France. Consulté sur https://hal.archives-ouvertes.fr/hal-00663353

Topin X., Régis C., Gleizes M.-P., Glize P. (1999, octobre). Comportements individuels adaptatifs dans un environnement dynamique pour l'exploitation collective de ressources . In *Intelligence Artificielle Située, cerveau, corps et environnement (IAS'99), Paris, France, 25/10/1999-26/10/1999*. Hermès.