Définition d'un cadre de conception et d'exécution pour la simulation multi-agent

Fabien Badeig [1,2], **Flavien Balbo** [1,2]

1. *Université Paris-Dauphine, LAMSADE*
   Place du Maréchal de Lattre de Tassigny, Paris Cedex 16, France
   prenom.nom@dauphine.fr

2. *INRETS Institute Gretia Laboratory*
   2, rue de la Butte Verte, 93166 Noisy Le Grand, France

RÉSUMÉ. *La simulation multi-agent est utilisée pour comprendre des systèmes complexes par la reproduction de divers scénarios. Dans les plates-formes de simulation actuelles, la politique d'ordonnancement consiste à contrôler l'activation séquentielle des agents qui exécutent systématiquement la boucle perception - décision - action. Cette approche centrée agent est peu efficace en termes de temps d'exécution et de conception. Nous proposons un nouveau cadre conceptuel et opérationnel* EASS *dans lequel les agents mutualisent, au sein de l'environnement, les informations et les traitements nécessaires à la mise en œuvre de la politique d'ordonnancement. Cette externalisation dans l'environnement d'une partie de l'activité des agents liée à leurs activations permet l'activation contextuelle. Les principaux avantages de l'activation contextuelle sont un gain d'efficacité en termes de temps d'exécution, et une meilleure flexibilité de la gestion des comportements des agents en termes de conception.*

ABSTRACT. *Agent-based simulation is used to understand complex systems and to experiment several scenarios. In classical simulation frameworks, a pitfall is the fact that the action phase, based on local agent context analysis, is repeated in each agent at each time cycle during the simulation execution. This analysis inside the agents reduces agent flexibility and limits agent behavior reuse in various simulations. If the designer wants to modify the way the agent reacts to the context, he could not do it without altering the way the agent is implemented because the link between agent context and agent actions is an internal part of the agent. Our proposition, called EASS, is a new agent-based simulation framework, where the context is analyzed by the environment and where agent activation is based on context evaluation. This activation process is called contextual activation. The main advantage of contextual activation is the improvement of complex agent simulation design in terms of flexibility and run-time.*

MOTS-CLÉS : *environnement, framework, simulation mutli-agent, politique d'activation.*

KEYWORDS: *environment, framework, simulation multi-agent, scheduling policy.*

DOI:10.3166/RIA.26.255-280

1. Introduction

La simulation de systèmes complexes constitue un enjeu majeur dans de nombreux domaines de la société comme les sciences sociales (Conte *et al.*, 1998 ; Doniec *et al.*, 2006), les écosystèmes (H. V. D. Parunak, 1999 ; Simonin, 2001 ; Bars *et al.*, 2002), ou les transports (Mandiau *et al.*, 2008). Elle permet de tester des hypothèses sur l'existant, de les exposer et d'en formuler de nouvelles a posteriori (Drogoul *et al.*, 1995 ; Meignan *et al.*, 2006 ; H. Parunak *et al.*, 2008). Ces propriétés font de la simulation un outil d'investigation pertinent quel que soit le domaine considéré.

Afin de modéliser ces systèmes complexes, l'approche fondée sur les systèmes multi-agents (SMA) est de plus en plus utilisée. En effet, la place accordée au paradigme multi-agent est de plus en plus importante dans des domaines très hétérogènes du fait de l'expressivité même de cette approche. Un des principaux résultats est une modélisation intuitive et générale du système. Un SMA est un système composé d'un ensemble d'entités autonomes, les *agents*, qui sont en interaction dans un environnement pour résoudre des problèmes (Ferber, 1999 ; Sycara, 1998). Pour être autonome, un agent doit agir sans intervention directe d'un autre système et il a le contrôle sur ses propres actions et sur son état interne. Ainsi, l'agent possède son propre processus caractérisé par ses trois phases *Perception-Décision-Action*, qui lui permet d'adapter son comportement à l'évolution de son contexte. Par une approche *bottom-up* de la modélisation d'un système, le paradigme multi-agent facilite l'analogie et la compréhension d'une réalité complexe en réifiant les composants du système. Ainsi par construction, une simulation reposant sur une modélisation multi-agent préserve les structures de la réalité simulée, les comportements pro-actifs des entités autonomes, les calculs parallèles, et les scénarios dynamiques de simulation (Davidsson, 2000).

D'après (Millischer, 2000), la simulation consiste à faire évoluer un modèle dynamique dans le temps afin notamment d'en évaluer la pertinence. De ce fait dans une plate-forme de simulation multi-agent, un ordonnanceur contrôle l'exécution d'un modèle multi-agent dans le temps. Ce contrôle est obligatoire afin d'atteindre les objectifs fonctionnels de reproduction, d'évaluation et de compréhension des systèmes complexes. Cependant, ce contrôle est contradictoire avec les fondements d'une modélisation multi-agent qui repose en grande partie sur l'autonomie des agents. Le compromis mis en œuvre par les plates-formes est de limiter ce contrôle à l'activation séquentielle et systématique des agents par l'ordonnanceur selon un modèle temporel qui définit l'évolution du temps dans la simulation. Un agent activé par l'ordonnanceur exécute la boucle *Perception-Décision-Action* comme il le ferait dans un SMA sans la simulation. Grâce à cette boucle, l'agent détermine l'action à exécuter à partir de la perception locale de son contexte. Cette conception centrée sur l'agent repose sur une relation figée entre l'ordonnanceur et les agents et impose trois limites. La première limite est le couplage fort entre le modèle agent et la plate-forme de simulation choisie car la manière dont le contexte sera récupéré et évalué par l'agent dépend de l'architecture de la plate-forme et de l'agent. Dès lors, le concepteur anticipe ce problème en prenant en compte dans son modèle les spécificités de la plate-forme de simulation. La conséquence directe est que le choix de la plate-forme contraint le

choix du modèle agent ce qui n'est pas conceptuellement satisfaisant. La deuxième limite est liée au processus répétitif qui consiste à récupérer et évaluer le contexte local de chaque agent. En effet, selon cette approche le contexte est calculé pour chaque agent à chaque cycle de temps de simulation même si le contexte de l'agent n'a pas évolué ou si une famille d'agents partage le même contexte. La troisième limite est liée à la flexibilité du modèle de simulation produit. L'objectif d'une simulation étant la compréhension d'un système complexe par la reproduction de scénarios, il est important que le concepteur puisse modifier le comportement des agents sans avoir à revenir sur leur implémentation. Le degré de *flexibilité* d'un modèle de simulation est évalué par rapport aux moyens dont dispose le concepteur pour modifier le comportement des agents en ligne ou hors ligne. Actuellement, si le concepteur veut modifier le comportement de sa simulation, il peut le faire de deux manières : soit il modifie l'ordonnanceur global, c'est-à-dire l'ordre d'activation des agents, soit il modifie le comportement des agents, à savoir leur réaction à un contexte, ce qui implique souvent une modification de l'implémentation des agents.

Pour pallier ces limites, nous proposons un nouveau cadre conceptuel et opérationnel pour la simulation multi-agent, appelé *Environment As Active Support for Simulation* (EASS), définissant un processus d'activation que nous appelons *activation contextuelle*. Contrairement aux approches se focalisant sur le processus de l'agent, notre approche place l'*environnement* au centre de la modélisation. Nous proposons en effet de déléguer à l'environnement la politique d'ordonnancement incluant le processus d'activation des agents. Ainsi, un agent est activé directement par l'environnement en fonction d'un contexte spécifique que chaque agent a défini au préalable, pour effectuer l'action qu'il a choisie pour ce contexte. Chaque lien entre un contexte et l'action associée est mutualisé dans l'environnement afin d'être partagé par les agents. La modification d'un lien modifie les conditions d'activation des agents concernés. Notre proposition est par conséquent flexible en termes de conception puisque la simulation peut être modifiée sans que l'implémentation des agents ne le soit. De même, elle est efficace en termes de temps d'exécution puisque le traitement des contextes concerne simultanément plusieurs agents. Les agents conservent leur autonomie en pouvant modifier eux-mêmes dans l'environnement les liens qui les concernent. Afin de modéliser le cadre conceptuel de EASS nous nous sommes fondés sur le principe de coordination, *Property-based Coordination* (PbC) (Zargayouna *et al.*, 2006). Selon ce principe la mutualisation des descriptions des composants du système afin de calculer les différents contextes permet une coordination des agents.

Notre proposition de renforcer le rôle de l'environnement est en cohérence avec le paradigme multi-agent. En effet, dans de nombreux travaux récents dont un état de l'art est présenté dans (Weyns, Parunak *et al.*, 2005), l'environnement est une abstraction du premier ordre pouvant rendre des services aux agents (Weyns *et al.*, 2006). L'environnement, en tant qu'entité dynamique du système, doit être modélisé au même titre que les agents. Nous proposons d'utiliser ses propriétés d'observabilité et d'accessibilité mises en évidence dans ces travaux pour supporter notre modèle. L'environnement est alors utilisé comme structure de mutualisation des descriptions et de calcul des contextes comme demandé par PbC.

L'article est organisé comme suit. La section 2 présente les différentes problématiques concernant la gestion du temps dans une simulation ainsi que les modèles de fonctionnement des principales plates-formes de simulation multi-agent. La section 3 décrit le cadre conceptuel et la section 4 le cadre opérationnel de EASS. La section 5 présente notre démarche expérimentale suivie des premiers résultats. Dans la section 6, nous concluons avec des remarques générales.

2. Positionnement

Notre objectif est d'améliorer la flexibilité et l'efficacité des simulations multi-agents. Afin d'y parvenir, nous proposons un nouveau processus d'activation des agents, l'*activation contextuelle*. Par définition, le processus d'activation dans une simulation est le mécanisme liant l'activité d'un agent à l'évolution du temps. Nous présentons dans cette section les problématiques et modèles relatifs à la gestion du temps dans une simulation puis les solutions proposées par les principales plates-formes de simulation multi-agent.

2.1. Le temps dans la simulation : problématiques et modèles

Lors de l'exécution d'une simulation, la difficulté consiste à assurer une cohérence temporelle, *i.e.* à respecter un modèle temporel. Ainsi chaque action peut être datée et la simultanéité des actions des agents peut être simulée. La politique d'ordonnancement définit les règles d'activation des agents selon le modèle temporel qui spécifie la date des actions.

La politique d'ordonnancement fixe les propriétés d'exécution de la plate-forme de simulation en définissant des règles. Un exemple de politique d'ordonnancement qui est classiquement utilisée dans les plates-formes avec un modèle temporel discret comme NetLogo (Wilensky, 1999) est présenté sur le tableau 1.

Tableau 1. Un exemple de règles définies par une politique d'ordonnancement

Règle 1	un agent ne doit pas pouvoir réaliser plusieurs actions au même moment
Règle 2	plusieurs agents doivent pouvoir agir au même moment
Règle 3	la durée d'une action doit être en cohérence avec le modèle temporel

Dans la littérature de la simulation, trois grandes familles de modèles temporels sont identifiées : les modèles à temps continu, les modèles à temps discret et les modèles à événements discrets.

Les modèles à temps continu sont caractérisés par un intervalle de temps fini où la variable *temps* est un réel, et les variables d'état ont une valeur quel que soit le moment

pris durant cet intervalle de temps. Le formalisme de base associé à ce type de modèle est appelé *Differential Equation System Specification (acronyme DESS)* (Cellier, 1991). Ce type de modèle coïncide naturellement avec la modélisation de phénomènes physiques réels dont la théorie utilise les équations de la physique, comme dans le domaine de la régulation de trafic qui se base sur la théorie des fluides (Lesort, 1995). Dans le cadre des simulations multi-agents, les agents sont par nature des entités qui changent de comportement de manière ponctuelle et donc le changement d'états d'un agent est un événement discret. Cette condition n'implique pas que les agents ne peuvent pas évoluer dans un modèle continu mais rend rares les simulations multi-agents utilisant ce modèle temporel.

Les modèles à temps discrets sont caractérisés par un axe de temps discrétisé suivant une période constante Δt, appelée *pas de temps* ou *cycle*. Tous les changements d'états du système se font à chaque pas de temps et les variables d'état du système sont discrétisées. Ainsi, le changement d'état est instantané et à intervalle régulier. Pour élaborer un modèle discret, il est nécessaire au préalable de définir les fonctions qui calculeront l'état du système à l'instant $t + \Delta t$ à partir de l'état du système à l'instant t. De manière plus formelle, il s'agit de définir une fonction θ telle que si $\sigma(t)$ exprime l'état du système à l'instant t alors $\sigma(t + \Delta t) = \theta(\sigma(t))$. Les conséquences immédiates de ce type de fonction est qu'il faut nécessairement calculer les états intermédiaires à partir de l'état initial pour connaître l'état du système à l'instant t. Il existe plusieurs familles de modèles à temps discret suivant la nature des fonctions qui définissent la dynamique du système. Par exemple, la dynamique du système est modélisée par deux fonctions dans le formalisme *Discret Time System Specification (acronyme DTSS)* (Zeigler *et al.*, 2000). La première fonction est la fonction de transition d'état, notée δ, qui permet de calculer le nouvel état interne du système tandis que la deuxième fonction est la fonction de sortie, notée λ qui détermine la valeur des variables en sortie du système. Ce modèle est le plus couramment utilisé dans les modèles multi-agents et nous y reviendrons en section 2.2.

La troisième famille de modèles est la famille des modèles à événements discrets où les changements d'état du système se font à des instants précis et de manière instantanée mais où l'axe temporel est continu, c'est-à-dire que la variable t représentant le temps est un réel. L'instant où les variables d'état évoluent est appelé un *événement*. La caractéristique des modèles à événements discrets est la manière dont l'évolution du temps est calculée à l'aide d'un échéancier qui stocke la date des événements futurs. La date de l'événement le plus récent déterminera la valeur suivante de la variable liée au temps. Classiquement, les modèles à événements discrets sont formalisés à partir de *Discret Event System Specification (acronyme DEVS)* présentées dans (Zeigler *et al.*, 2000). Les principes de DEVS ont pour origine les mathématiques discrètes. Il existe deux modèles fondés sur DEVS : le modèle *atomique* et le modèle *couplé*. L'un des intérêts de DEVS est de pouvoir coupler plusieurs modèles DEVS entre eux qu'ils soient *atomiques* ou eux-mêmes *couplés*, sachant qu'un modèle couplé est composé de modèles DEVS. Urhmacher (Uhrmacher, Schattenberg, 1998) et Duboz (Duboz, 2004) modélisent leurs problèmes à l'aide des systèmes multi-agents en utilisant le formalisme DEVS et parallel-DEVS. Ils présentent de manière plus approfondie et

complète le formalisme DEVS, et proposent également une méthodologie pour modéliser un SMA en utilisant ce formalisme et le couplage avec un système d'équations différentielles. L'idée élémentaire d'un simulateur par ordonnancement d'événements est de connaître au préalable la date des événements futurs. Cependant il n'est pas toujours aisé, voire impossible, de planifier les événements à venir. Une solution, appelée *analyse d'activités*, consiste à conditionner le déclenchement et l'exécution d'événements à partir de l'état du système. Le principe est de pouvoir déclencher des événements dont on ne connaît pas au préalable la date d'activation, par exemple lors de l'élaboration d'un simulateur de trafic on ne peut pas prévoir a priori quand une collision aura lieu. Par contre, on peut parfaitement définir le contexte de déclenchement d'un tel événement.

Notre proposition qui consiste à définir un nouveau processus d'activation, l'*activation contextuelle*, se situe dans la continuité de ces travaux. Alors que les modèles événementiels utilisent le contexte pour déterminer l'évolution du temps selon un échéancier d'évènements prédéfinis, nous conservons pour la gestion du temps la facilité des modèles à temps discret, et nous reprenons l'utilisation du contexte afin d'activer les agents selon les évènements de la simulation.

2.2. Politique d'ordonnancement dans les principales plates-formes de simulation multi-agent

Le paradigme multi-agent est fondé implicitement sur la composition de comportements individuels simultanés. L'implémentation de la simulation des concurrences a un impact direct sur l'évolution du modèle multi-agent. De ce fait, on ne peut concevoir une simulation multi-agent sans s'intéresser au préalable à la conception et l'analyse de l'ordonnancement. Selon le modèle à temps discret, le principe est d'un point de vue formel le suivant : l'évolution de l'état du monde d'un instant t à $t + \Delta t$ résulte de la composition des actions $A_1, A_2, ..., A_n$ des agents du système à l'instant t, selon une fonction de temps dynamique D (Michel *et al.*, 2001).

La difficulté de cette représentation réside dans la définition du terme *action* et dans l'implémentation de la composition des actions sur le plan de la simultanéité. La politique d'ordonnancement la plus utilisée consiste à activer les agents chacun leur tour de façon séquentielle car cette démarche est facile à mettre en œuvre. L'activation de l'ensemble des agents correspond à un pas de temps pour la simulation. Dans (Payet *et al.*, 2006), les auteurs proposent une amélioration de ce processus regroupant les agents selon des slots temporels afin de n'activer pour chaque slot que les agents concernés. À chaque activation, chacun des agents analyse son contexte pour déterminer l'action à réaliser en fonction de ses objectifs. À un pas de temps donné, le contexte d'un agent inclus le résultat des actions des agents qui ont été activées avant lui, ce qui accentue le rôle de l'ordonnanceur sur le comportement de la simulation. En effet, l'ordre d'activation des agents va impacter le résultat de la simulation. Une amélioration consiste à introduire une étape de validation des actions entre chaque transition du temps. Cette approche est une mise en œuvre du modèle influ-

ence/réaction proposé dans (Ferber, Muller, 1995 ; Michel, 2007). Le principe est de *figer* le monde à l'instant t afin que tous les agents aient la même perception de l'état du monde, noté $\sigma(t)$. Pour ce faire, un état du monde temporaire est créé où toutes les actions des agents sont reportées dans des variables temporaires sans conséquence sur l'état courant. Ensuite la synthèse de l'ensemble des actions est calculée en résorbant les conflits et le nouvel état est obtenu.

Les avantages de cette gestion des activations des agents sont que les conflits dûs aux accès aux variables de l'environnement sont évités et que le système garantit que tous les agents agissent une fois par cycle. Cependant la mise en œuvre de cette gestion au sein de plates-formes multi-agents présentent des limites. En effet, dans ces plates-formes, l'ordonnanceur est un composant spécifique et *extérieur* au système qui contrôle l'activation des agents. Par exemple, dans les plates-formes CORMAS (Bousquet *et al.*, 1998) et MASON (Luke *et al.*, 2004), l'ordonnanceur active une même méthode pour chaque agent. Pour spécialiser le comportement des agents, le concepteur doit alors surcharger cette méthode. Dans les plates-formes multi-agents de type LOGO, tels que l'outil de simulation *TurtleKit* de la plate-forme agent MADKIT (Ferber, Gutknecht, 2000) ou la plate-forme STARLOGO [1], un agent a un automate de comportements qui détermine l'action suivante à exécuter. Le concepteur doit alors mettre en œuvre pour chaque comportement une méthode correspondante, et définir les liens entre chacun des comportements de manière statique. Ces différentes plates-formes de simulation multi-agent sont évaluées et comparées dans (Tobias, Hofmann, 2004).

Dans tous les cas, à chaque activation un agent doit calculer son contexte, décider de l'action la plus adéquate puis exécuter cette action. Ces traitements sont répétitifs et obligent le concepteur à revenir sur la conception de l'agent s'il veut modifier son comportement, c'est-à-dire sur la phase de décision qui sépare le calcul du contexte et l'exécution de l'action. De notre point de vue, ces limites sont liées à la nature statique du lien entre un agent et le processus de simulation exécuté par l'ordonnanceur.

3. Un cadre conceptuel pour la simulation multi-agent

Notre objectif est d'améliorer l'efficacité et la flexibilité du modèle de simulation produit par le concepteur. Pour ce faire, nous externalisons de l'agent l'évaluation de son contexte et la sélection du comportement associé. Ces informations et traitements deviennent des composantes de l'environnement qui peut activer les agents. Pour permettre aux agents de conserver leur autonomie, ils doivent pouvoir manipuler ces composantes. Nous définissons ainsi un processus dynamique et souple d'activation des agents où un agent peut modifier simplement son lien à la simulation en cours d'exécution. Cette modification consiste à changer la relation entre un comportement spécifique de l'agent et le contexte associé. L'avantage est de séparer clairement ce qui relève du modèle agent (la définition de ses comportements) du processus de si-

1. http://education.mit.edu/StarLogo/

mulation (activation et ordonnancement des comportements des agents). Dans la section 3.1, nous présentons un exemple que nous utilisons tout au long du document afin d'illustrer nos propos. La section 3.2 introduit le principe *Property-Based Coordination* (PbC) et donne la formalisation du modèle EASS.

3.1. Un exemple illustratif de robots collaboratifs

Pour illustrer notre proposition, nous utilisons un exemple de simulation de robots à base d'agents inspirée du *Packet World* (Weyns, Helleboogh, Holvoet, 2005). L'intérêt de cette simulation est la mise en œuvre des principes suivants : la perception active, la prise de décision d'agents situés et la coordination entre agents. Dans cet exemple, les agents *robot* et les objets *caisse* sont situés sur une *grille* représentant un environnement spatial à deux dimensions. Les agents *robot* évoluent dans cet environnement et doivent coopérer pour déplacer des caisses (figure 1). Chaque robot a un champ de perception qui limite sa perception de l'environnement et des autres robots. La particularité des agents *robot* est d'avoir une compétence : soit *porter*, soit *soulever*, et pour déplacer une caisse, il faut deux robots avec des compétences différentes.

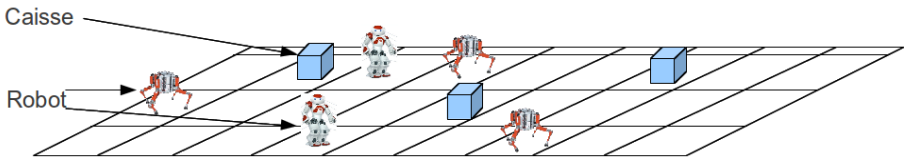

Figure 1. Un exemple de robots collaboratifs pour déplacer des caisses sur une grille à deux dimensions

Par défaut, un agent *robot* recherche des caisses dans son environnement. Pour ce faire, il se déplace aléatoirement d'une case jusqu'à ce qu'il détecte une caisse dans son champ de perception. Dans ce cas, à partir de la localisation de la caisse, il se déplace dans sa direction. Lorsque le robot se trouve à proximité de la caisse, deux situations sont identifiées :

1. soit un robot de compétence complémentaire est également à proximité de cette caisse et celle-ci peut être déplacée,
2. soit il est seul, alors l'agent *robot* attend l'arrivée d'un autre agent de compétence complémentaire pour déplacer cette caisse.

3.2. Formalisation des composants de l'environnement

Notre approche est fondée sur le principe de coordination *PbC* (Zargayouna *et al.*, 2006). L'objectif explicité pour PbC est : *de représenter les composants multi-agents par des composants symboliques et observables, et de gérer leur traitement à des fins de coordination*. Deux catégories de composants symboliques sont alors définies. La première catégorie est la description des composants réels du système multi-agent : les agents, les messages et les objets. La deuxième catégorie de composants symboliques

est liée aux éléments abstraits du système comme les composants de coordination. Les composants de coordination, que nous appelons des *filtres*, réifient le lien entre un contexte d'activation et une action de l'agent. Un contexte d'activation est un ensemble de contraintes sur des descriptions des composants du SMA. Lorsqu'un ensemble de descriptions vérifie les contraintes d'un filtre, le contexte exprimant un besoin de l'agent est constitué. L'ensemble de ces informations est géré par l'environnement du SMA.

Selon le modèle EASS, l'environnement est défini par le n-uplet :

$$\langle \Omega, \mathcal{D}, \mathcal{P}, \mathcal{F} \rangle$$

avec :

– $\Omega = \{\omega_1, ..., \omega_m\}$ l'ensemble des entités. Ces entités concernent les agents $\Omega_A \subset \Omega$ avec Ω_A l'ensemble des entités agents, les messages $\Omega_{MSG} \subset \Omega$ avec Ω_{MSG} l'ensemble des entités messages et les objets $\Omega_O \subset \Omega$ avec Ω_O l'ensemble des entités objets,

– $\mathcal{D} = \{d_1, ..., d_m\}$ l'ensemble des domaines de description des propriétés,

– $\mathcal{P} = \{p_1, ..., p_n\}$ l'ensemble des propriétés,

– $\mathcal{F} = \{f_1, ..., f_k\}$ l'ensemble des filtres.

3.2.1. Entité

Les entités sont les méta-informations sur le SMA qui sont gérées par EASS. Une *entité* ω_i est définie par un couple $\langle e_r, e_d \rangle$ où e_r est la référence sur un composant réel du système multi-agent (agents, messages, objets), et e_d la description enregistrée dans l'environnement de ce composant. La description d'une entité est définie par un ensemble de couples $\langle propriete, valeur \rangle$ qui est ensuite exploitée pour identifier les contextes d'activation des agents.

Une entité fait le lien entre le monde réel du SMA et le monde modélisé pour la simulation. Alors que la description d'un nouveau message est ajoutée de manière automatique dans l'environnement, la description d'un agent est laissée à l'initiative de l'agent qui a son propre processus de décision. L'agent doit maintenir à jour une description qui le représente dans l'environnement.

3.2.2. Propriété

Une propriété donne accès à une information sur une entité. Par définition, une propriété $p_i \in \mathcal{P}$ est une fonction dont le domaine de description $d_j \in \mathcal{D}$ peut être quantitatif, qualitatif, ou un ensemble fini de données. Elle s'écrit :

$$p_i : \Omega \rightarrow d_j \cup \{inconnu, vide\}$$

La valeur $inconnu$ est utilisée si la valeur de la propriété de l'entité existe mais n'est pas renseignée. La valeur $vide$ implique que cette propriété n'existe pas pour cette entité.

Un agent a au minimum dans sa description les propriétés id et $temps$ qui sont obligatoirement renseignée avec :

– $id : \Omega \rightarrow \mathbb{N}$, identifie de manière unique un agent.

– $temps : \Omega \rightarrow \mathbb{N}$, donne la prochaine date à laquelle l'agent souhaite être activée. L'utilisation de cette valeur sera précisée section 4.2.

Considérons l'exemple de simulation de robots décrit dans la section 3.1, la description d'un robot est complétée par un ensemble de propriétés qui sont détaillées dans le tableau 2.

Tableau 2. Les propriétés constituant la description des agents robot

nom	domaine de définition	rôle
$champ_{perception}$	$\Omega \rightarrow \mathbb{N} \cup \{inconnu, vide\}$	profondeur du champs de perception
id_{caisse}	$\Omega \rightarrow \mathbb{N} \cup \{inconnu, vide\}$	identifiant du paquet pris en compte par le robot
$competence$	$\Omega \rightarrow \{soulever, porter\} \cup \{inconnu, vide\}$	compétence du robot
$position$	$\Omega \rightarrow \mathbb{N} \times \mathbb{N} \cup \{inconnu, vide\}$	position du robot sur l'échiquier
$comportement$	$\Omega \rightarrow \{opportuniste, altruiste\} \cup \{inconnu, vide\}$	comportement du robot

A partir du tableau des propriétés ci-dessus, un agent *robot* peut par exemple correspondre aux informations suivantes :

$$\begin{array}{lccl} \langle & (\ id & , & 3\ ), \\ & (\ champ_{perception} & , & 4\ ), \\ & (\ id_{caisse} & , & inconnu\ ), \\ & (\ competence & , & porter\ ), \\ & (\ position & , & (6,8)\ ), \\ & (\ comportement & , & opportuniste\ ), \\ & (\ temps & , & 5\ ) \quad \rangle \end{array}$$

Cet agent *opportuniste* (propriété $comportement$) est défini par l'identifiant unique 3 (propriété id) perçoit les descriptions des autres composants du système qui sont situés à moins de 4 cases de lui (propriété $champ_{perception}$). Cet agent est localisé à la position $(6, 8)$ (propriété $position$) sur la grille et ne manipule aucune caisse car la propriété id_{caisse} est à $inconnu$. Cette description est enregistrée dans l'environnement, et sa mise à jour est assurée par l'agent.

3.2.3. Filtre

Un filtre identifie les entités selon leur description (e_d) et réalise l'interaction entre les objets concrets (e_r). Ici une interaction est comprise dans son sens premier c'est-à-dire une réaction réciproque entre deux objets. Un filtre $F_j \in \mathcal{F}$ est un n-uplet $F_j = \langle f_a, [f_m], [f_C], n_f \rangle$ avec n_f le nom du filtre et :

– $f_a : \Omega_A \rightarrow \{true, false\}$ une assertion qui identifie les agents concernés c'est-à-dire subissant l'interaction,

– $f_m : \Omega_{MSG} \rightarrow \{true, false\}$ une assertion optionnelle qui identifie les messages concernés,

– $f_C : P(\Omega) \rightarrow \{true, false\}$ un ensemble optionnel d'assertions identifiant les autres entités du contexte.

Un filtre est validé pour tout triplet $\langle agent[, message][, contexte]\rangle$ tel que $f_a(agent) \wedge f_m(message) \wedge f_C(contexte)$ est vrai. Cette définition sera complétée dans la section suivante afin de prendre en compte les contraintes d'ordonnancement. Par conséquent, chaque agent a dont la description valide les conditions de f_a, peut être activé pour subir l'interaction. Nous proposons deux types de filtres selon la nature de l'interaction : les filtres de communication et les filtres d'activation. Dans le cas d'un filtre de communication, l'agent a reçoit dans sa boîte aux lettres le message m dont la description valide f_m (obligatoire dans ce cas particulier). Dans le cas d'un filtre d'activation, l'action associée au filtre peut être toute action que l'agent a connaît et qu'il peut alors exécuter. Pour chacun de ces filtres il faut qu'il existe un sous-ensemble d'entités $contexte$ dont les descriptions valident f_C.

Considérons un filtre $\langle f_a, vide, f_C, F_e\rangle$ qui active un agent pour qu'il se déplace vers une caisse particulière. La valeur $vide$ pour l'assertion f_m indique dans ce cas que l'assertion f_m est inutilisée. Ce filtre doit se déclencher si un agent est inoccupé (conditions de f_a) et qu'il perçoit une caisse dans son champ avec un agent de compétence complémentaire en attente d'un autre agent pour déplacer la caisse (conditions de f_C). Le tableau 3 résume la conception du filtre F_e.

Tableau 3. Définition des trois assertions composant le filtre F_e

L'agent est inoccupé	$f_a : [position(?a) = ?pos_a]$ $\wedge[competence(?a) = ?comp_a]$ $\wedge[champ_{perception}(?a) = ?champ_a]$ $\wedge[id_{caisse} = inconnu]$
Son contexte est constitué de deux assertions	$f_C : f_{caisse}(?p) \wedge f_{robot}(?b)$
la caisse est perçue	$f_{caisse}(?p) : [position(?p) = ?pos_p$ $\& : \lvert ?pos_a - ?pos_p\rvert < ?champ_a]$ $\wedge[id(?p) = ?id_p]$
il existe un robot avec la compétence complémentaire qui souhaite déplacer la caisse	$f_{robot}(?b) : [id_{caisse}(?b) = ?id_p]$ $\wedge[competence(?b) \neq ?comp_a]$

Le symbole "?" précédant une lettre déclare une variable et "=" est l'opérateur de comparaison. Les opérateurs de comparaison définis dans notre modèle sont déterministes, *i.e.* dont le résultat est soit 0, soit 1. Ce filtre est validé s'il existe dans l'environnement un n-uplet d'entités $\langle a \in \Omega_A, \emptyset, \{p, b\} \subset \Omega\rangle$ dont les descriptions vérifient ces conditions. Dans la simulation de robots, si un agent perçoit une caisse dans son champ de perception, il veut se diriger vers cette caisse. Le contexte lui permettant

d'effectuer cette action est l'ensemble des informations sur la caisse. Ces informations sont accessibles grâce aux descriptions observables correspondant aux objets *caisse*. Quand un filtre est déclenché, l'action associée est activée pour l'agent concerné avec en paramètre de l'activation les paramètres relatifs au contexte. Cela signifie que dans la phase de modélisation, les relations entre les contextes et les actions doivent être construites. Le contexte d'un agent est calculé grâce aux informations disponibles et accessibles dans l'environnement.

Dans notre exemple de simulation de robots 3.1, cinq contextes sont identifiés. Le premier contexte correspond au déplacement d'une caisse ($deplacement_{caisse}$). Le deuxième contexte est relatif à la recherche d'une caisse ($recherche_{caisse}$) par un agent *robot*. Le troisième contexte correspond à la proximité d'un agent *robot* à une caisse ($proximite_{caisse}$). Le quatrième et cinquième contexte correspondent au cas où un agent *robot* détecte une caisse dans son champ de perception ($decouverte_{caisse\ proche}$ et $decouverte_{caisse\ manipulee}$). Le tableau 4 détaille chacun des contextes.

Tableau 4. Illustration de l'ensemble des contextes identifiés pour les agents robot

nom du contexte	description du contexte
$deplacement_{caisse}$	deux agents *robot* sont prêts à déplacer une caisse. Ce contexte nécessite l'accès aux informations suivantes : 1) les compétences des agents *robot*, 2) la position des agents *robot* et de l'objet *caisse*. Ce contexte est vérifié lorsque deux robots avec des compétences complémentaires (*soulever* et *porter*) sont à proximité d'une caisse.
$recherche_{caisse}$	un robot n'a détecté aucune caisse dans son champ de perception
$proximite_{caisse}$	un robot est à proximité d'une caisse et aucun robot avec la compétence complémentaire est proche de cette caisse
$decouverte_{caisse\ proche}$	un robot perçoit la caisse la plus proche dans son champ de perception
$decouverte_{caisse\ manipulee}$	un robot perçoit la caisse la plus proche qui est en cours de traitement par un agent avec la compétence complémentaire.

Les deux contextes liés à la découverte de caisses ($decouverte_{caisse\ proche}$ et $decouverte_{caisse\ manipulee}$) permettent d'avoir deux comportements différents des agents *robot* :

1. les robots opportunistes choisissent la caisse la plus proche,

2. les robots altruistes choisissent la caisse pour laquelle un autre agent est en attente pour la déplacer.

Nous avons défini des agents *robot* avec la compétence *porter* ou *soulever* et avec le comportement *altruiste* ou *opportuniste*. La figure 2 fournit le cadre qui permet de

modéliser des robots opportunistes ou altruistes. La différence entre les deux types de robots résulte du contexte qui déclenchera le comportement $deplacement_{direction}$.

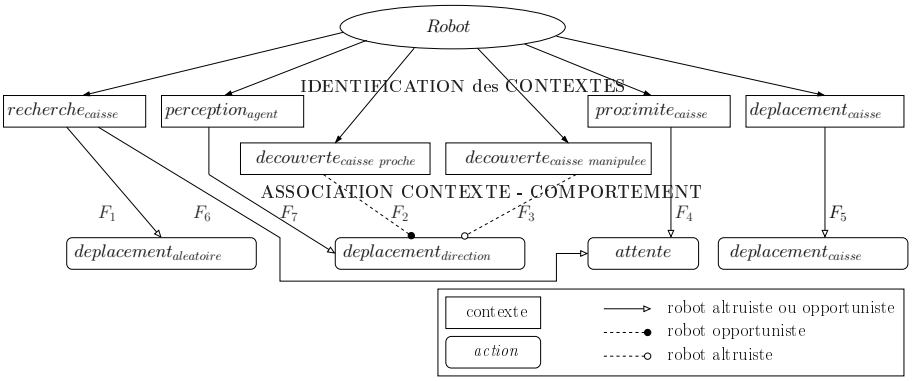

Figure 2. Lien entre les comportements d'un agent et ses contextes

3.3. Modélisation de l'activité d'un agent

A partir du modèle agent produit (figure 2), le concepteur instancie son propre modèle agent d'un robot en ajoutant à l'agent un automate de comportements. Cet automate caractérise les différents états que l'agent peut prendre en fonction des contextes perçus. Chaque filtre active un comportement et correspond à une transition dans l'automate. L'automate doit donc préciser les filtres nécessaires à l'agent ainsi que le comportement attendu de l'agent en réaction à l'activation de ces filtres. l'automate de la figure 3 indique que cet agent utilise les filtres F_1, F_2, F_4, F_5 et que l'agent exécute le comportement $deplacement_{aleatoire}$ puis $deplacement_{direction}$ par l'activation du filtre f_2. Pour concevoir un agent opportuniste, il suffit de remplacer le filtre F_2 par le filtre F_3. Ainsi, le comportement $deplacement_{direction}$ d'un agent *robot* est identique que ce soit pour les agents altruistes ou opportunistes ; par contre le contexte qui va permettre d'exécuter ce comportement diffère impliquant une modification du comportement des agents.

Le concepteur peut avec cette même bibliothèque concevoir d'autres comportements comme celui illustré par l'automate de la figure 4. Dans ce cas, l'agent reste en attente (F_6) tant qu'il ne perçoit pas un agent qui a une compétence complémentaire et qui se dirige vers une caisse (F_7). Puis il traite la caisse et se remet en attente. L'objectif de cet agent est de limiter ses déplacements pour, par exemple, économiser son énergie.

4. Un cadre opérationnel pour la simulation multi-agent

Dans notre approche, l'environnement gère les filtres issus de la phase de modélisation, ainsi que les informations relatives aux composants de la simulation (agents,

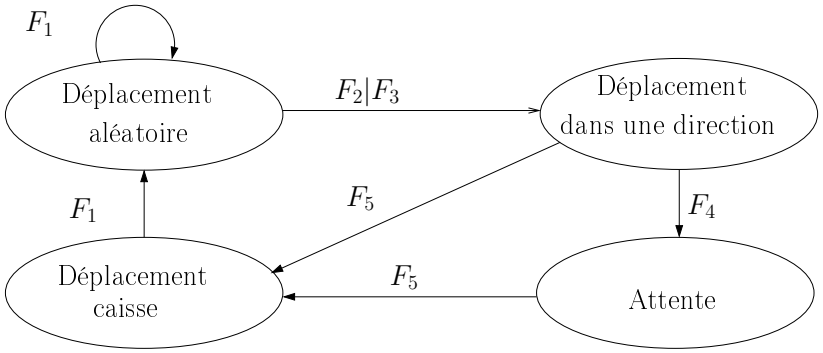

Figure 3. Automate des comportements d'un agent robot

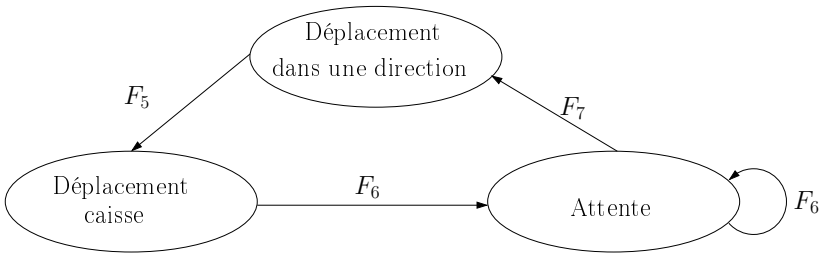

Figure 4. Comportement alternatif d'un agent robot

objets, messages). Le cadre conceptuel nous permet de modéliser le comportement des agents et nous montrons dans cette section comment une simulation peut ainsi être exécutée. Dans la sous-section 4.1, nous montrons les liens qui se forment entre les agents et leur environnement. La sous-section 4.2 présente comment l'ordonnancement assure l'activation des agents qui le souhaitent. Dans la sous-section 4.3, nous décrivons le fonctionnement général de notre ordonnanceur.

4.1. Activation des agents par l'environnement

Le rôle de l'ordonnanceur intégré à l'environnement est d'assurer l'*activation contextuelle* des agents. Pour ce faire, l'environnement doit manipuler la description des entités du système et les filtres des agents. Ainsi, l'ordonnanceur vérifiera l'adéquation entre les descriptions des entités et les filtres.

Chaque nouvel agent de la simulation s'enregistre à l'environnement ce qui consiste à s'identifier et à y ajouter sa description. Puis, il dépose l'ensemble des filtres composant son automate des comportements. Chaque filtre exprime un lien d'activation pour un contexte donné et un comportement précis et donc l'ordonnanceur peut activer un agent lorsque le contexte est évalué. Au cours de la simulation, l'agent met à jour sa description, exécute les actions correspondant aux filtres déclenchés le concernant et maintient à jour son automate des comportements. La mise à jour de sa descrip-

tion consiste à modifier la valeur des propriétés qui ont été modifiées par son action comme, par exemple, la propriété $position$ des agents robot suite à un déplacement. Un autre agent ne peut pas modifier ces valeurs. L'exécution des actions se fait selon le principe de l'activation contextuelle et est contrôlée par l'ordonnanceur comme nous le décrivons dans la section suivante. Chaque agent peut modifier dynamiquement son lien à la simulation en ajoutant ou supprimant des filtres à l'ordonnanceur géré par l'environnement. Uniquement les filtres contenus dans l'environnement participent au processus de simulation, *i.e.* sont pris en compte par l'ordonnanceur global. Le concepteur peut ainsi instancier un agent qui a initialement le comportement correspondant à l'automate de la figure 3 mais qui modifie son comportement pour celui de l'automate de la figure 4 dès qu'il n'a plus d'énergie. Pour cela, il ne doit plus être concerné par les filtres $\{f_1, f_2, f_4\}$ mais par les filtres $\{f_6, f_7\}$.

Les filtres présents dans l'environnement sont partagés par tous les agents dont la description peut correspondre à l'assertion f_a du filtre. L'ajout et le retrait effectif des filtres de l'environnement par un agent auraient donc des conséquences sur le comportement des autres agents. Pour pallier ce problème, nous avons mis en œuvre un mécanisme d'*abonnement* aux filtres. La définition initiale d'un filtre est complétée par une composante supplémentaire que nous appelons *agentsPotentiels*. Un filtre s'écrit alors :

$$\langle f_a, [f_m], [f_C], n_f, agentsPotentiels \rangle$$

où $agentsPotentiels$ est l'ensemble des agents qui adhèrent au filtre. Ainsi, un agent qui souhaite modifier son activation contextuelle, utilise les primitives de l'environnement qui permet d'ajouter ou de supprimer des filtres en modifiant l'appartenance de l'agent à la liste *agentsPotentiels*. Sa décision est sans conséquence sur le fonctionnement des autres agents.

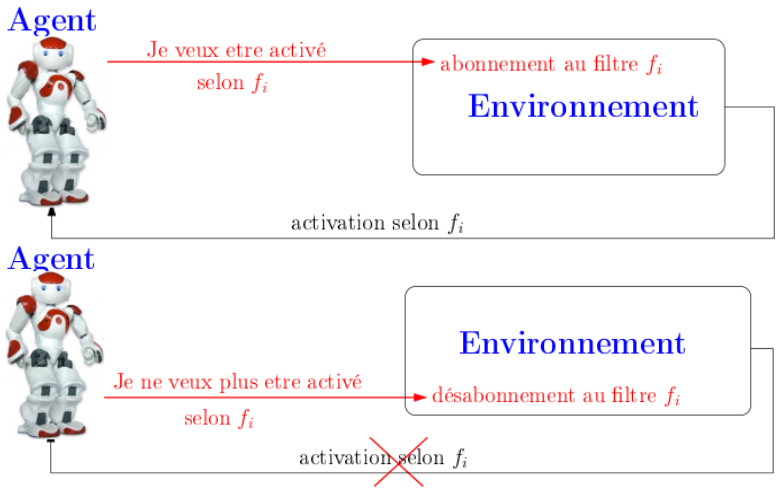

Figure 5. Dynamique d'activation entre un agent et son environnement d'exécution

Durant la phase de modélisation, un concepteur peut modifier sa simulation en changeant le lien entre les contextes d'activation et les actions des agents. L'avantage est que ces modifications sont réalisées sans modifier le code de l'agent concerné. Ainsi, le comportement de l'agent peut être modifié de deux façons : 1) soit en changeant l'action associé à un filtre ce qui traduit que le concepteur modifie la réaction d'un agent à un contexte précis, 2) soit en changeant le contexte d'un filtre associé à une action, dans ce cas le concepteur modifie la situation qui déclenche une action précise. Cette flexibilité de conception des agents est complétée par la flexibilité lors de la phase d'exécution où l'agent ajoute ou supprime dynamiquement un lien d'activation au cours de la simulation activant ou désactivant une réaction à un contexte spécifique. Ce mécanisme permet à l'agent de s'adapter à l'évolution de son environnement et assure la dynamique du système. La figure 5 illustre cette notion.

4.2. Contrôle de l'activation d'un agent

L'environnement doit assurer le respect des règles de la politique d'ordonnancement. Selon le principe de l'activation contextuelle, chaque agent peut être activé à chaque cycle en fonction du contexte qu'il a choisi et qui est évalué par l'environnement. Ainsi, les règles qui doivent être mises en œuvre sont les suivantes :

1. chaque agent est activé au plus une fois par cycle de temps ce qui correspond à vérifier la propriété d'*unicité* de l'action d'un agent ;

2. tout agent devant être activé, est effectivement activé, ce qui correspond à vérifier la propriété de *complétude* des activations lors d'un cycle de simulation.

4.2.1. Assurer l'unicité

La difficulté pour la première règle de la politique d'ordonnancement est double. En effet, il faut sélectionner à chaque cycle de temps pour chaque agent une action parmi l'ensemble des actions possibles, et une fois l'action exécutée par l'agent, le désactiver jusqu'au cycle de temps suivant même si de nouveaux contextes deviennent possibles. Afin de résoudre ce problème de conflits d'activation, nous avons complété la définition d'un filtre avec la propriété *priorité*. Un filtre s'écrit alors :

$$\langle f_a, [f_m], [f_C], n_f, agentsPotentiels, priorite \rangle$$

où $priorite$ donne le niveau de priorité du filtre. Cette valeur est identique pour tout agent inscrit dans la liste *agents potentiels*. Nous définissons ainsi un ordre entre les filtres. Un agent est toujours activé par le filtre le plus prioritaire parmi l'ensemble des filtres correspondant aux contextes possibles. Afin d'assurer qu'il ne sera pas réactivé dans le même cycle de temps, nous utilisons la propriété $temps$ de l'agent dont la valeur correspond à la date à laquelle l'agent veut de nouveau être activé.

Pour que la gestion du temps de la simulation soit cohérente, nous imposons la contrainte de fonctionnement suivante : lorsqu'un agent est activé, il exécute l'action associée à l'activation et il met automatiquement à jour la valeur de sa propriété $temps$. Cette propriété caractérise l'horloge interne de l'agent. De la même manière,

une description de l'environnement d'exécution est maintenue et accessible aux agents où le temps de la simulation est enregistré dans une propriété T_E. La mise à jour de cette propriété est de la responsabilité de l'environnement. La comparaison des valeurs relatives à la propriété $temps$ des agents et à la propriété T_E au moment de l'évaluation des filtres d'activation assurent le contrôle de l'unicité du déclenchement d'un agent pour un cycle de temps. Ainsi, pour chaque filtre d'activation, nous imposons la contrainte temporelle $[temps(?a) < T_E]$ avec $?a$ l'agent subissant l'interaction. Cette contrainte empêche un même agent d'être activé plusieurs fois dans un même cycle de temps et limite également les filtres à tester car le déclenchement d'un filtre désactive les autres filtres pour ce même agent.

4.2.2. Assurer la complétude

La deuxième règle de la politique d'ordonnancement consiste à n'activer que les agents qui le souhaitent et de ne pas en oublier. Il est en effet possible avec notre proposition de permettre à un agent de ne pas être activé lors d'un cycle ou de simuler des comportements complexes qui doivent s'exécuter sur plusieurs cycles tout en améliorant l'efficacité de la simulation.

La condition minimale de chaque filtre d'activation qui compare le temps interne de l'agent (propriété $temps$) avec le temps de la simulation (propriété T_E), permet en effet à un agent de ne pas être considéré pendant plusieurs cycles de simulation en indiquant un temps interne supérieur au temps de la simulation. Le contraire peut également se produire, c'est-à-dire un agent qui souhaite être activé (temps interne inférieur au temps de la simulation) mais pour lequel aucun contexte ne correspond. Pour résoudre ce problème, nous proposons l'ajout systématique d'un filtre $\langle [temps(?a) < T_E], null, null, F_{defaut}, \Omega_A, 1 \rangle$ avec seulement la contrainte temporelle entre les propriété $temps$ et T_E dont la priorité est minimale (1). Ainsi, ce filtre se déclenche pour tout agent vérifiant la contrainte temporelle (donc souhaitant être activé) avant que le temps de la simulation ne soit modifié comme nous le verrons dans la section suivante.

L'avantage de notre gestion de l'activation est que le nombre d'agents activés dans un cycle de temps de simulation est variable et il correspond aux agents qui doivent réellement être activés. Lorsque tous les agents ont un temps interne supérieur au temps de la simulation, tous les agents devant être activés ont été activés et le cycle de temps de la simulation est terminé. Le temps de la simulation est alors mis à jour de t à $t + \delta t$.

4.3. Politique d'ordonnancement

L'environnement gère la politique d'ordonnancement qui consiste, dans notre approche, à organiser l'ensemble des filtres et à gérer le temps de la simulation. Notre but est d'avoir un même formalisme et outil pour définir et gérer la politique d'ordonnancement et le temps de la simulation. Pour ce faire, nous utilisons le formalisme proposé pour les filtres de communication et d'activation. Par conséquent, l'environnement

contient un nouveau type de filtre lié à la gestion de la simulation. Pour obtenir une exécution cohérente de la simulation, chaque évaluation de filtre doit être planifiée. Avec notre approche, il existe deux niveaux de planification : le niveau d'ordonnancement global qui commande l'exécution de la simulation, et le niveau d'ordonnancement local qui contrôle l'activation contextuelle des agents qui a été présentée à la section précédente.

Au niveau de l'ordonnancement global, nous proposons de contrôler l'exécution de la simulation par un automate d'états que nous appelons *automate d'exécution*. Un état de l'automate est un ensemble de filtres homogènes, *i.e.* dont la nature d'action est similaire : communication, activation ou propre à la simulation. Les transitions entre les états de l'automate d'exécution sont conditionnées par la valeur d'une propriété $etat$ qui est enregistrée dans la description de l'environnement. Le filtre, nommé selon la convention $F_{etatX \rightarrow etatY}$ modifie la valeur de la propriété $etat$ de $etatX$ à $etatY$. La transition d'un état X à un état Y de l'automate d'exécution est réalisée si tous les filtres de l'état X ne peuvent plus être déclenchés. Un cycle de simulation correspond alors à l'évaluation de tous les filtres qui sont dans chaque état de l'automate d'exécution. L'ordre des états de l'automate d'exécution est important car il détermine le groupe de filtres qui sera évalué en premier. Par conséquent, si le concepteur veut modifier le comportement de simulation et tester d'autres scénarios, il peut également modifier l'ordre des états de l'automate d'exécution.

Par défaut, l'automate d'exécution possède un seul état correspondant à la gestion de la simulation qui contient au moins deux filtres F_{defaut} qui active les agents par défaut, et F_{temps} qui met à jour le temps de la simulation. Avec ce seul état dans l'automate et ces deux filtres, notre simulateur émule la politique d'ordonnancement classique des simulations multi-agents à temps discret. Le filtre F_{temps} assure une gestion discrète du temps de la simulation. Le filtre F_{defaut} déclenche l'action $defaut\ Action$ qui doit être surchargée par le concepteur pour définir le comportement de l'agent. Dans ce cas, à chaque cycle de temps, tous les agents sont activés sans évaluation a priori du contexte autre que la comparaison entre le temps interne de l'agent et le temps de la simulation par le filtre F_{defaut}. Dans la méthode $defautAction$, le concepteur doit alors prévoir de récupérer et évaluer le contexte pour déterminer le comportement que l'agent doit adopter.

Dans la simulation de robots, l'automate d'exécution est composé de deux états. Le premier état contient les filtres associés aux contextes : *découverte caisse*, *proximité caisse* et *deplacement caisse* (respectivement F_2 ou F_3, F_4 et F_5). Le deuxième état contient les filtres F_1 et F_{temps}. F_1 est le filtre par défaut des agents robots qui consiste à rechercher une caisse en se déplaçant aléatoirement et dont la condition d'activation porte uniquement sur la comparaison entre le temps interne de l'agent et le temps de la simulation.

La notion d'activation contextuelle présente plusieurs avantages : 1) deux niveaux de contrôle de la simulation sont possibles, un niveau d'ordonnancement global et un niveau d'ordonnancement local, il en résulte davantage de flexibilité en permettant de modifier facilement le comportement de la simulation ; 2) l'activation est contextuelle

et elle est réalisée par l'environnement ce qui évite un calcul répétitif local à chaque agent ; 3) cette stratégie de contrôle empêche un agent d'être activé plus d'une fois dans un cycle de temps et permet à un agent d'être inactif pendant une certaine période de temps.

Un exemple d'exécution d'une simulation de robots

Cette section illustre la politique d'ordonnancement et donne un scénario d'exécution d'une simulation de robots (section 3.1). Au cours d'une phase d'initialisation, les composants du système (les agents *robot* et les objets *caisse*) s'inscrivent dans l'environnement. Pour chaque inscription, une description est créée où les couples $(id, valeur)$ et $(temps, 0)$ sont générés et ajoutés à cette description.

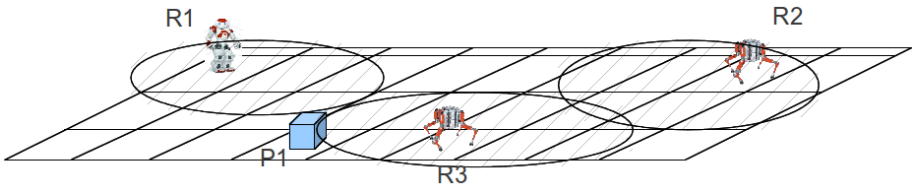

Figure 6. Initialisation des agents robot R1, R2, R3, et de la caisse P1

Lorsque l'environnement a été initialisé, la simulation peut démarrer. Le temps de simulation est $T_E = 1$ au démarrage. Le premier état de l'automate d'exécution est activé et les filtres le composant sont évalués et déclenchés en fonction de l'adéquation avec les descriptions contenues dans l'environnement. Prenons le cas où deux agents *robot* R_1 et R_2 ne perçoivent pas de caisse et l'agent *robot* R_3 détecte la caisse P_1. R_3 est activé par F_2 pour effectuer l'action *déplacement dans une direction* dont l'orientation est calculée par le contexte vérifiant les conditions du filtre, *i.e.* $position$ et id de la description de la caisse P_1. L'horloge interne de R_3 est mise à jour à 1. Quand l'environnement a activé tous les agents pouvant agir en fonction de l'évaluation des filtres composant l'état 1 de l'automate d'exécution, le filtre $F_{etat1 \rightarrow etat2}$ est activé ce qui permet de passer à l'état 2 de l'automate d'exécution. Le premier filtre déclenché est F_1. Donc, R_1 et R_2 sont activés séquentiellement pour effectuer l'action *déplacement aléatoire*. Le concepteur doit définir les règles topologiques de l'environnement pour ensuite les traduire dans la partie prémisse des filtres. Par exemple, si plusieurs robots ne peuvent pas être sur la même case alors il suffit de ne laisser percevoir que les case disponibles et les plus pertinentes pour le déplacement. Leur horloge interne est mise à jour à 1. Tous les agents ont leur temps interne supérieur ou égal à T_E ($T_E = 1$). Par conséquent, le filtre F_{temps} est déclenché et le temps de la simulation T_E est incrémenté de 2, puis le filtre $F_{etat2 \rightarrow etat1}$ permet de revenir sur le premier état de l'automate d'exécution.

Maintenant, R_3 est à proximité de la caisse P_1, R_1 perçoit la caisse P_1 et R_2 ne perçoit aucune caisse. Le temps de simulation est $T_E = 2$ et le premier état de l'automate d'exécution est activé. En respectant la priorité des filtres, R_3 est activé par

le filtre F_4 pour effectuer l'action *attente*. Son horloge interne est mise à jour à 2. Puis, R_1 est activé par le filtre F_3 pour effectuer l'action *déplacement dans une direction* avec les informations provenant de la description de la caisse P_1. Son horloge interne est mise à jour à 2. Tous les agents pouvant être activés par les filtres de l'état 1 de l'automate d'exécution ont agi, le cycle actuel de la simulation passe à la deuxième phase correspondant à l'état 2 de l'automate d'exécution. R_2 est activé par le filtre F_1 pour effectuer l'action *déplacement aléatoire* et son horloge interne est mise à jour à 2. Tous les agents *robot* ont leur temps interne supérieur ou égal à T_E ($T_E = 3$). Par conséquent, le filtre de mise à jour du temps de la simulation est déclenché et le temps de la simulation T_E est incrémenté.

5. Expérimentations et premiers résultats

Afin d'étudier la faisabilité de notre proposition, un prototype a été développé comme plug-in de la plate-forme multi-agent MADKIT (Ferber, Gutknecht, 2000). MADKIT est une plate-forme modulaire et évolutive écrite en Java. Elle permet une forte hétérogénéité des architectures des agents et des langages de communication. Elle permet également un haut degré de personnalisations. L'architecture logicielle de MADKIT est basée sur des plug-ins et accepte facilement l'ajout ou la suppression d'autres plug-ins pour s'adapter à des besoins spécifiques. Notre plug-in est composé d'un composant environnement avec une API qui permet aux agents d'ajouter/retirer/modifier leurs descriptions et des filtres. Pour respecter la dynamique de notre approche, nous avons choisi de la mettre en œuvre à l'aide d'un système à base de règles (RBS) fondé sur l'algorithme bien connu RETE (Forgy, 1982). L'instanciation des composants du système en RBS est intuitive : les descriptions sont les faits et les filtres sont les règles. L'évaluation et le déclenchement des règles dépendent de l'algorithme RETE qui est un algorithme efficace et largement étudié.

Nous avons entrepris de valider notre proposition selon deux critères : 1) la flexibilité des modèles de simulation générés, 2) l'efficacité de l'activation contextuelle. La flexibilité est un critère difficile à mesurer et nous avons choisi de le faire selon le principe suivant : un modèle de simulation est flexible s'il peut être manipulé afin d'obtenir de nouveaux scénarios sans être directement modifié. Par "manipuler" nous comprenons la possibilité de prendre en compte ou de supprimer des comportements en ligne ou hors ligne. Par "sans être directement modifié" nous comprenons qu'il n'y a pas de nouveaux développement à réaliser. Afin d'évaluer ce critère, nous avons étudié la possibilité avec un même modèle agent de générer différents scénarios qui impactent le comportement général de la simulation. L'efficacité de notre modèle de simulation mettant en œuvre le principe d'activation contextuelle est mesurée par comparaison avec un modèle d'activation classique où l'évaluation du contexte est réalisée localement pour chaque agent.

Nous rappelons que précédemment l'exemple a été simplifié à 3 robots et une caisse alors que pour les expérimentations, nous testons avec 20 robots et 20 caisses sur une grille de 150×150. L'exemple est décrit en détail dans (Badeig *et al.*, 2007).

5.1. Évaluation de la flexibilité

Nous avons testé six scénarios de simulation ($S1$ à $S6$) au moyen de neuf filtres : cinq filtres pour l'activation contextuelle (F_1 à F_5, voir figure 3), trois filtres pour la communication ($F_{recept1}$, $F_{recept2}$ et F_{accept}) et un filtre pour l'activation classique F_{defaut}. Ce dernier filtre qui participe au scénario $S1$, $S3$ et $S5$, permet de comparer notre modèle avec un modèle intégrant un processus d'activation classique. Les scénarios $S1$ à $S6$ sont définis dans le tableau 5, $S1$ et $S2$ sont deux scénarios sans communication, $S3$ à $S6$ sont quatre scénarios avec communication.

Tableau 5. Les scénarios sur la simulation de robots pour l'évaluation de notre modèle EASS

Scénario	activation	communication
$S1$	$\{F_{defaut}\}$ + Analyse du Contexte Local à l'Agent (ACLA)	$\emptyset$
$S2$	$\{F_1, F_2, F_3, F_4, F_5\}$	$\emptyset$
$S3$	$\{F_{defaut}\}$ + ACLA	$\{F_{recept1}, F_{accept}\}$
$S4$	$\{F_1, F_2, F_3, F_4, F_5\}$	$\{F_{recept1}, F_{accept}\}$
$S5$	$\{F_{defaut}\}$ + ACLA	$\{F_{recept2}, F_{accept}\}$
$S6$	$\{F_1, F_2, F_3, F_4, F_5\}$	$\{F_{recept2}, F_{accept}\}$

$S1$ illustre un scénario classique avec une phase d'activation et une analyse du contexte locale à chaque agent (ACLA) ; $S2$ est un scénario avec une activation contextuelle issue de l'environnement ; les scénarios $S3$ et $S5$ sont une extension du scénario $S1$ avec deux modes de communication resultant de l'utilisation des filtres $F_{recept1}$ ou $F_{recept2}$; $S4$ et $S6$ illustrent la modélisation unifiée de l'activation contextuelle et de la communication.

Ces scénarios ont permis de tester la flexibilité de notre modèle où, en ajoutant des filtres pour la communication sans modifier le modèle agent et en changeant seulement le filtre de réception $F_{recept1}$ de $S3$ et $S4$ par $F_{recept2}$ de $S5$ et $S6$, nous avons pu générer trois types de scénarios et deux politiques de communication différentes. La première politique correspond à un mode de communication de type *broadcast* limité, et la seconde est une mise en œuvre du protocole Contract Net (CNP). Ces expérimentations sur un exemple simple montrent la flexibilité de notre approche EASS, *i.e.* la facilité avec laquelle il est possible de générer des simulations différentes, sans modifier l'architecture des agents.

5.2. Évaluation de l'efficacité

Nous avons exécuté trois jeux de simulations en fonction de plusieurs paramètres : le nombre d'agents *robot*, le nombre d'objets *caisse* et la taille du champ de perception. Pour chaque jeu de simulations, nous avons comparé notre modèle avec l'approche classique en fonction du temps moyen d'exécution sur 50 simulations avec les mêmes paramètres. Sur la première série, nous avons testé avec 5 robots possédant la

compétence *porter*, 5 robots avec la compétence *soulever* et 5 caisses, sur la seconde série avec 15 robots avec la compétence *soulever*, 15 robots avec la compétence *porter* et 15 caisses, et sur la troisième série avec 20 robots avec la compétence *soulever*, 20 robots avec la compétence *porter* et 20 caisses. Nous avons utilisé les deux scénarios $S1$ et $S2$, qui ne tiennent pas compte des coûts de communication, $S1$ étant un scénario basé sur un processus d'activation classique avec une analyse locale du contexte de l'agent (ACLA), $S2$ étant le scénario correspondant à notre approche avec une activation contextuelle. Comme on peut le voir sur la figure 7, le scénario $S2$ est plus efficace que le scénario $S1$. Pour un champ de perception compris entre 8 et 30, la courbe en pointillé relative au temps d'exécution de $S2$ est en dessous de la courbe en trait plein correspondant au temps d'exécution de $S1$. Comme on s'y attendait, un champ de perception plus large améliore l'efficacité des agents *robot* pour le scénario $S2$. En effet, lorsque le champ de perception des agents augmente de 5 à 30, le temps d'exécution des simulations diminue de $4\ 000$ à 800 secondes. Au contraire, dans le cas du scénario $S1$ avec un processus d'activation classique, le coût de l'analyse du contexte locale à chaque agent augmente linéairement avec la taille du champ de perception. Ainsi, l'amélioration de la perception des caisses en augmentant le champ de perception compense à peine le coût de l'analyse locale du contexte.

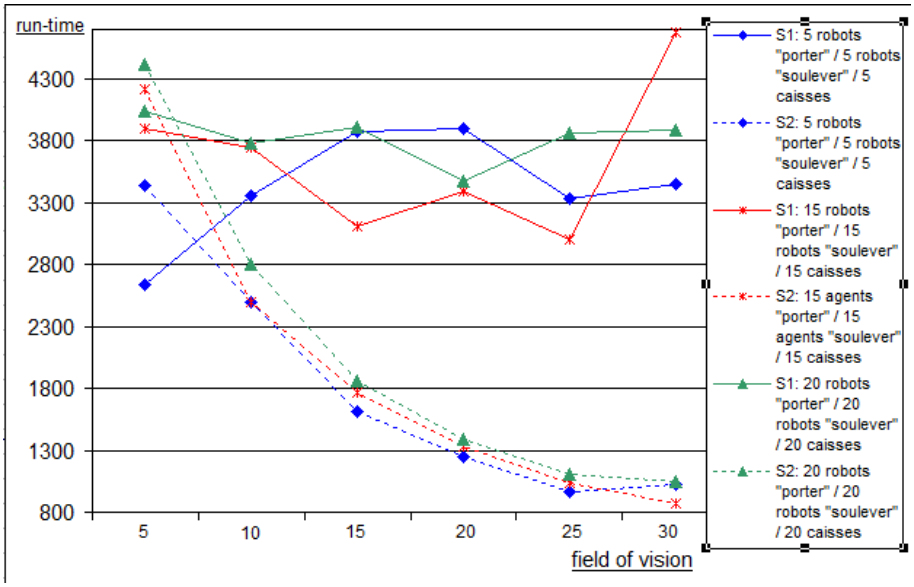

Figure 7. Temps d'exécution des simulations pour les scénarios S_1 et S_2

Nous avons également comparé sur la figure 8 les scénarios sans communication (S_1 et S_2) et avec communication (S_3, S_4, S_5 et S_6) en fixant le champ de perception des robots à 15 et en faisant varier le nombre de robots et de caisses dans l'environnement. Lorsque les agents ont recours aux modes de communication pour localiser plus rapidement les caisses dans l'environnement, le temps des simulations est amélioré. En effet, pour les scénarios mettant en œuvre l'activation classique avec une

analyse locale aux agents de leurs contextes, les scénarios avec communication S_3 et S_5 donnent de meilleurs résultats que le scénario sans communication S_1. Ce résultat est pertinent par rapport à l'implémentation car la communication améliore la perception des agents et surtout évite un traitement local du contexte étant donnée que la position de la caisse est directement identifiée dans le message. De la même manière, les filtres de communication des scénarios S_4 et S_6 vont modifier l'état de l'agent pour le diriger vers la caisse localisée dans le message ce qui désactive l'évaluation des filtres de plus basse priorité. Ainsi, les scénarios S_4 et S_6 fournissent des temps d'exécution plus rapides que le scénario S_2. La deuxième analyse des résultats de la figure 8 concerne la comparaison entre scénarios mettant en œuvre l'approche classique et l'activation contextuelle. On constate que les conclusions formulées à partir des résultats de la figure 7 restent valables pour les scénarios avec communication. En effet, le scénario S_4 (resp. S_6) avec communication et activation contextuelle est plus rapide en temps d'exécution que le scénario S_3 (resp. S_5).

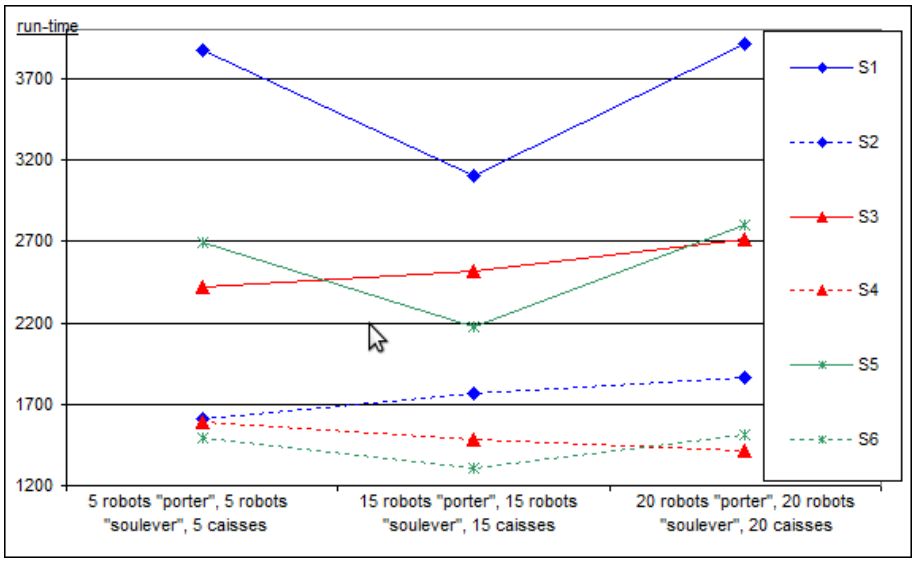

Figure 8. Temps d'exécution des simulations pour les scénarios S_1, S_2, S_3, S_4, S_5 et S_6 en fonction du nombre de robots et du nombre de caisses, le champ de perception étant fixé à 15

6. Conclusion

Dans cet article, nous avons défini un cadre de conception et d'exécution pour les simulations multi-agents, appelé *Environment as Active Support for Simulation* (EASS). L'originalité de notre approche est de déléguer à l'environnement un ensemble de services dont une partie de l'activité des agents relative à l'évaluation de son contexte local. Notre proposition est de représenter explicitement le lien entre les actions des agents et leurs contextes d'activation et ainsi de pouvoir manipuler librement

cette réification de l'activation des agents. De fait, ces nouveaux liens d'activation modifient le processus classique exécuté par le moteur d'ordonnancement des simulations multi-agents tout en gardant les propriétés d'autonomie, de réactivité et de proactivité propres aux agents. Nous avons appelé ce nouveau processus d'activation, l'*activation contextuelle*. Notre proposition fondée sur le principe de coordination *PbC* défend le fait que les entités d'un système multi-agent doivent être partiellement observables à travers un ensemble de propriétés.

Notre proposition a été implémentée et testée en utilisant la plate-forme MADKIT. Nous avons entrepris deux types de validation sur l'efficacité de notre approche et sur sa flexibilité. En ce qui concerne l'efficacité, notre approche est en moyenne meilleure qu'une approche utilisant un processus d'activation classique. En ce qui concerne la flexibilité, notre approche permet une gestion fine de différents scénarios de simulation en modifiant en ligne ou hors ligne les relations entre les agents et leur ordonnanceur (l'environnement). Nous avons l'intention de tester notre plate-forme sur un problème complexe, réaliste et à plus large échelle qui est le problème de gestion de crises dans les transports (Badeig *et al.*, 2008 ; Badeig, Balbo, 2008).

Un certain nombre de points difficiles à différents niveaux de modélisation restent à traiter pour évaluer notre approche dans son ensemble. La première concerne la formalisation des filtres et plus précisément des opérateurs de comparaison car nous n'utilisons pour l'instant que des opérateurs de comparaison déterministes ($=$, $>$, $<$,...) ne permettant pas de simuler des incertitudes dans la perception des agents. Nous envisageons de prendre en compte l'incertitude de l'évaluation du contexte pour l'activation des agents en étendant l'ensemble des opérateurs de comparaison aux opérateurs flous. Un autre point difficile qui n'est actuellement pas pris en compte est la robustesse de notre approche concernant l'incohérence de l'ensemble des filtres. En effet, chaque agent dispose d'un ensemble de filtres cohérents. Cependant, les éventuelles incohérences liées à la mise en commun de ces ensembles doivent être étudiées. Un autre point délicat dans notre approche est l'impact du nombre de filtres dans l'environnement pour évaluer la dépendance de notre modèle par rapport à la dynamique de simulation.

Bibliographie

Badeig F., Balbo F. (2008). La gestion de crise dans les transports, un simulateur multi-agent centré environnement. In *Atelier systèmes d'information en transport,* INFORSID*'08.*

Badeig F., Balbo F., Pinson S. (2007). Contextual activation for agent-based simulation. In *Proceedings of* ECMS*'07*, p. 128–133.

Badeig F., Balbo F., Scémama G., Zargayouna M. (2008). Agent-based coordination model for designing transportation applications. In *Proc. of the 11th int. ieee conf. on intelligent transportation systems*, p. 402–407. IEEE Computer Society.

Bars M. L., Attonaty J.-M., Pinson S. (2002). An agent-based simulation for water sharing between different users. In *Proceedings of the first international joint conference on autonomous agents and multiagent systems*, p. 211–212. ACM.

Bousquet F., Bakam I., Proton H., Page C. L. (1998). CORMAS : Common resources and multi-agent systems. In Springer-Verlag (Ed.), *Proceedings of the 11th international conference on industrial and engineering applications of artificial intelligence and expert systems*, p. 826–837.

Cellier F. E. (1991). *Continuous system modeling*. Springer-Verlag.

Conte R., Gilbert N., Sichman J. (1998). Mas and social simulation : A suitable commitment. In *Proceedings of the first international workshop on multi-agent systems and agent-based simulation*, p. 1–9. Springer-Verlag.

Davidsson P. (2000). Multi-agent-based simulation: Beyond social simulation. In *Proceedings of mabs*, p. 97–107.

Doniec A., Mandiau R., Piechowiak S., Espié S. (2006). L'anticipation comme modèle d'interaction : application à la ccordination multi-agent en simulation. In *Proceedings of the 15ème congrès francophone reconnaissance des formes et intelligence artificielle (*RFIA *2006)*, p. 1443–1454.

Drogoul A., Corbara B., Lalande S. (1995). MANTA: New experimental results on the emergence of (artificial) ant societies. In N. Gilbert, R. Conte (Eds.), *Artificial societies: The computer simulation of social life*, p. 190–211. London, UCL Press.

Duboz R. (2004). *Intégration de modèles hétérogènes pour la modélisation et la simulation de systèmes complexes : Application au transfert d'échelles en écologie marine*. Thèse de doctorat non publiée, Université du Littoral - Côte d'Opale.

Ferber J. (1999). *Multi-agent systems : An introduction to distributed artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc.

Ferber J., Gutknecht O. (2000). Madkit: A generic multi-agent platform. In *4th international conference on autonomous agents*, p. 78–79.

Ferber J., Muller J.-P. (1995). Influences and reaction: A model of situated multiagent systems. In *Proceedings of the international conference on multi-agent systems.*

Forgy C. L. (1982). Rete : A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, vol. 19, p. 17–37.

Lesort J. (1995). Modélisation des réseaux de trafic : de la planification à l'exploitation. In *Annales des ponts et chaussées*, p. 34–48.

Luke S., Cioffi-Revilla C., Panait L., Sullivan K. (2004). Mason: A new multi-agent simulation toolkit. In *Swarmfest workshop.*

Mandiau R., Champion A., Auberlet J.-M., Espié S., Kolski C. (2008). Behaviour based on decision matrices for a coordination between agents in a urban traffic simulation. *Applied Intelligence*, vol. 28, n° 2, p. 121–138.

Meignan D., Simonin O., Koukam A. (2006). Multiagent approach for simulation and evaluation of urban bus networks. In *Proceedings of the 4th workshop on agents in traffic and transportation*, p. 50–56.

Michel F. (2007). Le modèle irm4s de l'utilisation des notions d'influence et de réaction pour la simulation de systèmes multi-agents. In *Revue d'intelligence artificielle (ria).*

Michel F., Gutknecht O., Ferber J. (2001). une méthodologie pour la conception de simulateur multi-agents basée sur l'organisation. In *Plate-forme* AFIA *2001 - atelier* SMA.

Millischer L. (2000). *Modélisation individu centrée des comportements de recherche des navires de pèche*. Thèse de doctorat non publiée, ENS Agronomique de Rennes. (Thèse en Halieutique)

Parunak H., Belding T., Hilscher R., Brueckner S. (2008). Understanding collective cognitive convergence. In *Proccedings of the 9th international workshop on multi-agent-based simulation (mabs)*, p. 1–15.

Parunak H. V. D. (1999). "go to the ant": Engineering principles from natural multi-agent systems. In *Annals of operations research.*

Payet D., Courdier R., Ralambondrainy T., Sébastien N. (2006). Le modèle à temporalité : pour un équilibre entre adéquation et optimisation du temps dans les simulations agents. In *Actes des journées francophones des systèmes multi-agents (*JFSMA*'06)*, p. 63–76.

Simonin O. (2001). *Le modèle satisfaction-altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique*. Thèse de doctorat non publiée, Université de Montpellier II.

Sycara K. (1998). Multiagent systems. In *Ai magazine*, p. 79–92.

Tobias R., Hofmann C. (2004). Evaluation of free java-libraries for social-scientific agent based simulation. *Journal of Artificial Societies and Social Simulation*, vol. 7, n° 1, p. 31.

Uhrmacher A. M., Schattenberg B. (1998). Agents in discrete event simulation. In *Proceedings of the 10th european simulation symposium "simulation in industry – simulation technology: Science and art" (ess'98)*, p. 129–136.

Weyns D., Bouck N., Holvoet T. (2006). Gradient field-based task assignment in an agv transportation system. In *Aamas '06: Proceedings of the fifth international joint conference on autonomous agents and multiagent systems*, p. 842–849. ACM Press.

Weyns D., Helleboogh A., Holvoet T. (2005). The packet-world: A test bed for investigating situated multiagent systems. In *Software agent-based applications, platforms and development kits, whitestein series in software agent technology*, p. 383–408.

Weyns D., Parunak H. V., Michel F., Holvoet T., Ferber J. (2005). Environments for multiagent systems, state-of-the-art and research challenges. In D. Weyns, H. V. Parunak, F. Michel (Eds.), *Proceedings of the 1st international workshop on environments for multi-agent systems (lecture notes in computer science, 3374)*, p. 1–47.

Wilensky U. (1999). *Netlogo (and netlogo user manual)*. Rapport technique. Center for Connected Learning and Computer-Based Modeling, Northwestern University. (http://ccl.northwestern.edu/netlogo/)

Zargayouna M., Trassy J. S., Balbo F. (2006). Property based coordination. In I. J. Euzenat, J. Domingue (Eds.), *Artificial intelligence: Methodology, systems, applications*, vol. 4183, p. 3–12. Springer Verlag.

Zeigler B. P., Praehofer H., Kim T. G. (2000). *Theory of modeling and simulatin - integrating discrete event and continuous complex dynamic systems* (Second Edition éd.). Academic Press.