Approche formelle pour la modélisation et la simulation à évènements discrets de systèmes multi-agents

Romain Romain Franceschini

Université de Corse – Pasquale Paoli
école doctorale environnement et société
umr cnrs 6134 (spe)

Thèse présentée pour l'obtention du grade de

Docteur en Informatique

Soutenue publiquement par

Romain Franceschini

le 11 décembre 2017

Approche formelle pour la modélisation et la simulation à évènements discrets de systèmes multi-agents.

Directeurs

Mr. Paul-Antoine Bisgambiglia, Professeur, Université de Corse
Mr. Paul-Antoine Bisgambiglia, Maître de conférences, Université de Corse

Rapporteurs

Mr. Hans Vangheluwe, Professeur, Université d'Anvers
Mr. Jean-Christophe Soulié, Cadre de Recherche (HDR), CIRAD Montpellier

Jury

Mme. Evelyne Vittori, Maître de conférences, Université de Corse
Mr. David R. C. Hill, Professeur, Université Clermont Auvergne
Mr. Hans Vangheluwe, Professeur, Université d'Anvers
Mr. Jean-Christophe Soulié, Cadre de Recherche (HDR), CIRAD Montpellier
Mr. Dominique Federici, Professeur, Université de Corse

Membres invités

Mr. Paul-Antoine Bisgambiglia, Professeur, Université de Corse
Mr. Paul-Antoine Bisgambiglia, Maître de conférences, Université de Corse

REMERCIEMENTS

Je tiens à remercier chaleureusement mon directeur de thèse, le Professeur Paul-Antoine Bisgambiglia pour avoir encadré mes travaux, pour la confiance qu'il m'a accordé, pour son sens du détail, pour ses encouragements, ainsi que pour ses conseils.

C'est en toute amitié que je remercie son homonyme, Paul-Antoine Bisgambiglia (MCF), pour avoir co-encadré cette thèse. À travers sa disponibilité, son soutien et les nombreuses discussions que nous avons partagées, il a su m'encadrer de près tout en m'accordant une grande liberté dans mes travaux. En m'encourageant à faire preuve d'initiative, il a largement contribué à mon épanouissement scientifique.

Je leur exprime toute ma gratitude pour m'avoir initié au monde académique et pour m'avoir accompagné dans l'apprentissage de ses subtilités.

Je remercie très sincèrement Monsieur Hans Vangheluwe et Monsieur Jean-Christophe Soulié, qui m'ont fait l'honneur de rapporter cette thèse. Leurs lectures respectives, à la fois positives et critiques m'ont permis de prendre confiance en mon travail tout en m'offrant le recul nécessaire pour apprécier ses limites.

Je tiens à remercier chaleureusement Monsieur David Hill pour avoir accepté de présider mon jury de soutenance. Également membre de mon comité de pilotage, ses remarques et nos échanges ont grandement enrichis ma culture scientifique et ont joués un rôle certain dans l'orientation de mes travaux de thèse.

Je remercie également Madame Evelyne Vittori et Monsieur Dominique Federici pour avoir accepté de faire partie de mon jury de soutenance, ainsi que pour l'intérêt qu'ils ont porté à mon travail.

Je tiens également à remercier Monsieur Gauthier Quesnel et Monsieur Eric Ramat, tous deux membres de mon comité de pilotage, pour le temps qu'ils m'ont accordé et pour tous leurs conseils.

Je remercie tous les membres de l'équipe informatique du laboratoire, pour leur accueil et pour leur bienveillance.

Merci également à toute l'équipe pédagogique de l'Université de Corse qui m'a conduit jusqu'ici.

Je tiens à remercier mes collègues, qui m'ont permis d'oublier temporairement ma thèse, et dont certains sont devenus des amis. Puisque j'ai occupé deux bureaux différents au sein du laboratoire, je commencerais par remercier ceux que j'ai pu côtoyer dans le premier. Merci à Hélène pour sa joie de vivre communicative. Merci à Andréa, Ingrid, Lætitia B.,

Sandrine, Johan et Stéphanie pour leur bonne humeur et pour leur gentillesse.

Un grand merci à Damien, avec qui j'ai beaucoup partagé durant cette thèse. D'abord scientifiquement, à travers nos discussions qui ont permis d'améliorer la qualité de mon travail, et humainement, à travers les nombreux moments d'amitié que nous avons partagés autour d'une table, en montagne ou au pan d'escalade. Merci également à lui pour son immense soutien en fin de rédaction et pour ses relectures accélérées.

Merci à Marie-Laure Nivet pour m'avoir accueillie dans ce qui a été mon second bureau. Si je la côtoie aujourd'hui comme collègue, j'ai eu la chance de l'apprécier comme enseignante charismatique, pédagogue et dévouée.

Merci à Gauthier, qui a toujours su nous réunir avec ses excellents breuvages, chauds et froids, consommés au bureau et ailleurs. Fidèle compagnon de rédaction, il m'a permis de rester motivé jusqu'au bout.

Merci à Mohamed pour les discussions passionnantes que nous avons partagées, pour avoir su satisfaire ma curiosité et pour les efforts de vulgarisation dont il a fait preuve.

Merci aux autres doctorants de l'étage pour avoir égayé nos trop peu nombreuses pauses café.

Merci à Briac, Mohamed, Gauthier et Damien qui se sont portés volontaires pour imprimer et relier mon manuscrit afin de me faire gagner un temps précieux en fin de rédaction.

Merci à Roland pour son aide précieuse lors de la traduction en Corse du résumé vulgarisé de cette thèse.

Il m'est difficile de trouver les mots justes pour remercier tous mes proches, famille et amis, qui ont tous joué une part importante dans mon parcours.

Merci à mes parents et à Thomas, mon frère, pour leur soutien, leur patience, leurs encouragements, leurs relectures et pour avoir cru en moi.

Merci à Lætitia qui a eu le courage de partager toute cette aventure avec moi, qui a su composer avec mes humeurs et qui a su me faire oublier mon travail quand il le fallait. Merci également à elle pour son soutien indéfectible, pour ses relectures et pour son précieux sens de la logistique lors de la soutenance.

Merci à mes amis de longue date, que j'ai trop souvent oublié ces derniers temps, et qui ont répondu présent physiquement ou moralement pour me soutenir le jour J.

Enfin, merci à tous ceux qui se sont déplacés parfois de très loin pour assister par un jour pluvieux à ma soutenance et pour avoir partagé ce moment avec moi.

Corte, le 8 février 2018.

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE

Contexte

Aujourd'hui, la simulation numérique est devenue un outil essentiel pour étudier, comprendre et anticiper des phénomènes naturels. Utilisée à la fois pour élargir nos connaissances sur ces phénomènes et pour apporter une aide à la décision, la simulation prend une place de plus en plus importante dans notre quotidien. Ces systèmes, qualifiés de complexes, sont constitués d'un grand nombre d'entités en interaction. L'étude de leur comportement se caractérise par les nombreux aspects qui peuvent échapper à notre compréhension. La science des systèmes complexes permet de mieux appréhender leur étude à travers l'élaboration d'abstractions, appelées modèles. Ainsi, le domaine transdisciplinaire de la modélisation et la simulation (M&S) connaît un développement croissant, motivé en particulier par des enjeux globaux : écologie, gestion de ressources, *etc*.

Afin d'appréhender les systèmes complexes et d'en capturer les subtilités, l'activité de modélisation est habituellement guidée par un schéma de pensée faisant consensus au sein d'une communauté de modélisateurs, appelé paradigme de modélisation. Un nombre important de paradigmes de modélisation existent, le plus connu étant sans doute le paradigme objet. Parmi ces schémas de pensée, le paradigme agent constitue une approche de modélisation particulièrement adaptée à la description de phénomènes complexes naturels. Il encourage l'adoption d'un niveau d'analyse fin, pour considérer le système étudié selon des entités hétérogènes qui sont appelées agents. Sa capacité à saisir la complexité des interactions entre ces agents et leur environnement constitue un atout certain, particulièrement pour observer des phénomènes émergents.

Le paradigme agent est issu de deux domaines. Il est au carrefour de l'Intelligence Artificielle Distribuée et de la Vie Artificielle (Ferber, 1995). Selon Maes (1995), les deux domaines partagent le même objectif : celui de réaliser des agents autonomes et adaptatifs. L'Intelligence Artificielle Distribuée est une branche de l'Intelligence Artificielle (IA) apparue à l'aube des années 80 aux Etats-Unis avec les travaux de Kornfeld (1979) et d'Erman et al. (1980) sur la communication par partage d'informations, et de Lenat (1975) sur la résolution collective de problèmes. Alors que les travaux en Intelligence Artificielle Distribuée se concentrent sur l'organisation de systèmes et des processus de raisonnement pour la résolution collective de problèmes, le domaine de la Vie Artificielle, impulsé par Langton (1989), s'attache à la compréhension et à la modélisation de systèmes vivants. La

Vie Artificielle étudie notamment les processus d'évolution, de survie, d'adaptation, de reproduction, et d'apprentissage ou plus généralement, le comportement du vivant et de son interaction avec l'environnement.

Les systèmes multi-agents (SMA) permettent d'étudier des systèmes hétérogènes et de résoudre des problématiques complexes. Ils sont au cœur de nombreuses thématiques de recherche (Oliveira et al., 1999), qui incluent notamment : les différentes politiques et stratégies de résolution collective de problèmes (coordination, interaction, négociation) ; la communication entre agents (protocoles de communication, langages) ; l'adaptativité et l'apprentissage des agents ; ou encore des problématiques d'ingénierie logicielle incluant l'architecture des agents et des SMA, ou la conception de langages de programmation basés sur le paradigme agent. Au delà des thématiques de recherche, les domaines d'applications concernés sont nombreux puisque les SMA sont utilisés en robotique (Kitano et al., 1997 ; MacKenzie et al., 1997), en éthologie (Corbara et al., 1993 ; Drogoul, 1993), en écologie et en biologie (Bousquet et Le Page, 2004 ; Bousquet et al., 1994 ; Coquillard et Hill, 1997), ou encore en sciences sociales (Deffuant et al., 2002 ; Richiardi et al., 2006).

Conscient de leurs atouts, le Laboratoire Sciences Pour l'Environnement (UMR CNRS 6134 SPE), dans lequel cette thèse s'inscrit, porte un intérêt grandissant aux systèmes multi-agents (Urbani, 2006) dans le but d'étudier des systèmes naturels complexes. Ce laboratoire est une Unité Mixte de Recherche pluridisciplinaire dont le projet scientifique repose sur la maîtrise, la gestion et l'exploitation des ressources naturelles ainsi que sur la compréhension de la dynamique des systèmes naturels complexes, notamment à travers leur modélisation et leur simulation.

Parmi les projets portés au sein du laboratoire, les domaines d'applications concernent la gestion halieutique ou la propagation des feux de forêts. À ce titre, la simulation de systèmes multi-agents permettrait d'envisager l'étude de stratégies de lutte incendie ou encore l'étude de différentes politiques de pêche. Si des modèles sont développés pour ces applications, leurs résultats pourraient éventuellement être utilisés comme support d'aide à la décision. Cependant, la conception et la description de tels modèles est difficile, et leurs comportements aussi complexes que les phénomènes qu'ils sont censés représenter (Duboz et al., 2012).

Les enjeux liés aux résultats de simulation et aux modèles SMA pouvant être importants, il est indispensable d'inscrire les activités de M&S au sein d'un protocole scientifique rigoureux. Puisque la simulation est utilisée comme une expérience numérique, elle doit se conformer aux pratiques de toute science expérimentale, et être assujettie aux mêmes contraintes. Il s'agit de décrire les modèles de manière non ambigüe afin de permettre son partage avec la communauté scientifique afin d'en assurer sa reproductibilité, et donc la validation par les pairs.

Nos objectifs principaux seront de :

1. proposer une approche visant à améliorer la reproductibilité des expériences numériques dédiées aux systèmes multi-agents.

2. fournir au laboratoire un outil de modélisation et de simulation dédié aux systèmes multi-agents, en adéquation avec les problématiques environnementales.

Propositions

Afin de répondre au premier objectif qui vise à permettre la description non ambigüe de modèles SMA, nous adoptons dans ce manuscrit une approche formelle, afin de tenter de décrire précisément la structure et le comportement d'un modèle SMA. Pour cela, nous nous orientons vers la communauté de la Théorie de la Modélisation et de la Simulation (TMS), qui semble offrir les outils permettant de remplir notre objectif. Nous ferons également la proposition de méthodes permettant d'accompagner le modélisateur dans le processus de M&S, pour tenter de le guider dans la définition de modèles SMA.

Nos contributions peuvent être perçues comme une tentative visant à concrétiser la vision d'autres chercheurs, qui partagent notre volonté de réunir la communauté de la TMS et des SMA. Particulièrement, nous plaçons ces travaux comme la suite logique de ceux initiés par Duboz et al. (2012), en fournissant les « aspects techniques et purement formels » qui découlent de leur réflexion. Nous couvrons également une partie des perspectives de travail de Michel (2004), qui consiste à élaborer un formalisme qui permet d'englober les notions développées par l'auteur, associée à des méthodes de conception.

Pour répondre à notre deuxième objectif, nous proposons de mettre en œuvre les abstractions permettant de faciliter le développement de SMA, d'après les concepts définis par l'approche formelle.

Organisation du mémoire

Dans le chapitre I, nous posons le cadre de la modélisation et de la simulation (M&S) afin de définir les concepts fondamentaux que nous manipulons tout au long de ce document. Ce chapitre est également l'occasion de poser certaines définitions associées au paradigme agent et d'étudier leur conception du point de vue de l'ingénierie logicielle, afin d'offrir un environnement permettant de concevoir et de simuler des SMA. Nous mettons ensuite en avant les problématiques liées à la simulation de SMA avant de mettre en évidence l'utilité des méthodes formelles pour leur modélisation.

Comme nous le verrons, la spécification formelle est un facteur déterminant pour la reproductibilité d'une expérience numérique. Dans le chapitre II, nous réalisons un état de l'art des différents outils formels issus du domaine de la théorie de la modélisation et de la simulation. Il cherche à proposer une vue d'ensemble des possibilités offertes par ces outils, ce qui permet de guider et d'orienter notre choix quant au formalisme le plus à même de représenter le paradigme agent. Il sera montré que le formalisme PDEVS rassemble selon nous les différentes qualités permettant d'exprimer un modèle basé sur le paradigme agent. Celui-ci ayant déjà été utilisé pour représenter des systèmes multi-agents, nous abordons les

spécificités et les objectifs des différents travaux pour spécifier des SMA. En vue de remplir nos objectifs, nous discutons de l'intérêt de proposer une nouvelle spécification formelle des SMA.

Le chapitre III constitue le cœur de nos travaux. Nous y proposons une spécification formelle du paradigme agent, baptisée DPDEMAS pour *Dynamic Parallel Discrete Event Multi-Agent Specification*, pour tenter d'aiguiller le modélisateur dans la conception d'un modèle SMA, tout en étant suffisamment générique pour envisager de construire une bibliothèque de composants réutilisables. Associée à cette spécification, nous proposons une méthode en deux phases couvrant une partie du processus de conception de la M&S. La première phase permet de guider la spécification informelle d'un système multi-agent. Cette spécification constitue un prérequis à la seconde méthode, qui permet de guider le modélisateur dans la spécification formelle d'un modèle SMA basé sur la spécification générique DPDEMAS.

Le chapitre IV décrit le développement de Quartz, un *framework* (cadriciel) mettant en œuvre nos travaux de formalisation. Cet environnement de M&S répond à l'objectif n° 2 visant à faciliter le développement de modèles SMA basés sur notre spécification DPDEMAS. Nous décrivons son architecture, que nous avons voulue modulaire et extensible, et qui permet d'exécuter des modèles basés sur différentes extensions du formalisme PDEVS. Une bibliothèque offrant des abstractions concernant le paradigme agent est également proposée. Nous décrivons aussi ses différentes fonctionnalités, qui offrent des perspectives de vérification des modèles. Nous présenterons également les optimisations que nous avons effectuées afin de fournir des performances d'exécution satisfaisantes à notre outil.

Avant de conclure et de donner les perspectives de nos travaux, nous illustrons dans le chapitre V la mise en œuvre de nos différentes contributions à travers la réalisation de deux modèles d'exemple, de la spécification informelle à l'implémentation du modèle.

Chapitre I

SYSTÈMES MULTI-AGENTS ET SIMULATION

Sommaire

1 *Introduction*

Comme nous l'avons souligné dans l'introduction générale, les domaines d'applications des systèmes multi-agents sont très larges. Notre intérêt se porte sur la modélisation du comportement de systèmes vivants. La modélisation et la simulation des processus d'évolution, de survie, d'adaptation, de reproduction, et d'apprentissage ou plus généralement, le comportement du vivant et de son interaction avec l'environnement, doit nous permettre d'acquérir de nouvelles connaissances à propos des phénomènes complexes étudiés. Cette

recherche de connaissances doit répondre à de fortes exigences, notamment lorsqu'il s'agit de produire des résultats qui peuvent être le support de décisions. Pour cela, l'usage de la modélisation et de la simulation doit être régie par les mêmes contraintes qu'une expérience scientifique. Le respect de la méthode scientifique impose la mise en place d'un protocole d'expérimentation rigoureux offrant la possibilité de reproduire une expérience numérique ainsi que ses résultats, afin de permettre sa validation, et *in fine*, une transparence des décisions qui peuvent en découler.

L'objectif de ce chapitre est de donner les définitions indispensables à la compréhension de nos travaux, tout en soulignant, comme mis en évidence par Duboz et al. (2012), qu'un rapprochement de la communauté SMA avec celle de la théorie de la modélisation et la simulation peut être hautement bénéfique (Duboz et al., 2012), particulièrement pour remplir nos objectifs de reproductibilité. Pour cela, nous débutons ce chapitre en définissant les concepts associés aux systèmes complexes, ainsi que les concepts liés à leur représentation, à travers les notions d'observation, de modélisation, de simulation, d'analyse et de validation. Une fois ces définitions données, nous décrivons plus en détail les raisons qui ont conduit à l'apparition du paradigme agent. Nous donnons les définitions associées à ce paradigme, puis proposons une étude des différentes architectures logicielles et méthodologies de conception qui ont été proposées pour la modélisation de systèmes multi-agents. Nous discutons également dans cette partie de l'intérêt des méthodologies de conception existantes afin de déterminer si elles sont adaptées au processus de M&S. Enfin, après avoir exposé ces bases théoriques, nous reviendrons sur la problématique de la reproductibilité, sur ses apports et les éléments de solution que nous souhaitons appliquer, notamment l'intérêt d'utiliser des méthodes formelles pour modéliser des SMA.

2 *Terminologie*

Pour comprendre la modélisation et la simulation (M&S), il est nécessaire d'établir la relation entre le système réel, qui représente l'objet d'étude, et le modèle qui est la représentation du système réel. C'est la théorie générale des systèmes qui réalise ce lien et sur laquelle est basée la théorie de la modélisation et de la simulation. Puisque notre travail concerne et s'appuie sur la modélisation et la simulation, il convient de clarifier et de définir les concepts que nous utilisons tout au long de ce document. Ainsi, nous dégageons dans un premier temps la notion de système, de système complexe et de système dynamique avant d'étudier ce qu'est un modèle, la simulation d'un modèle ou encore un paradigme de modélisation.

2.1 *Système*

Il existe tellement de définitions différentes du concept de système qu'il est difficile de s'arrêter sur l'une d'entre elles. La définition proposée par l'Association Française d'Ingénierie Système [1] (AFIS) couvre, selon nous, l'ensemble des propriétés qui forment un

1. https://www.afis.fr

système :

> *« Un système est décrit comme un ensemble d'éléments en interaction entre eux et avec leur environnement, intégré pour rendre à son environnement les services correspondants à sa finalité. Un système présente donc des propriétés nouvelles résultant des interactions entre constituants : si l'on intègre des éléments pour faire un système, c'est bien pour bénéficier des effets de synergie résultant de leurs interactions. »*

Par système, on désigne un phénomène naturel complexe (physique, biologique, chimique, météorologique, ...) ou plus généralement un système réel dont on souhaite comprendre le fonctionnement (l'objet d'étude). D'après la définition de l'AFIS, les aspects essentiels d'un système sont :

1. L'*interaction* entre les entités qui composent le système. Les entités, éléments possédant une dynamique propre, évoluent à travers des échanges d'informations, de matière ou d'énergie. Ces flux forment la notion d'interaction.
2. L'*environnement*, avec lequel le système entretient une relation. Si les entités interagissent entre elles, le système est situé dans un environnement et échange avec lui des flux d'informations.
3. Sa *finalité*, la raison pour laquelle un système peut interpeller, susciter l'intérêt et être questionné. Elle caractérise ce que le système apporte à son environnement.
4. La *synergie* résultant de l'interaction de l'ensemble des entités qui composent le système. Le système possède une dynamique globale qui émerge de l'ensemble des entités et de leurs interactions. C'est la propriété d'*émergence* du système.

De Rosnay (1975), vient appuyer cette définition puisqu'il voit un système comme un ensemble d'éléments en interaction, organisés en fonction d'un but. Un système est donc *organisé*, *finalisé*, *constitué* d'éléments et de leurs relations et enfin, il forme un *tout* qui dépasse la somme de ses parties.

Dans la suite, nous allons voir que cette vision du concept de système s'inscrit dans le courant de pensée de la *systémique*, qui permet d'appréhender des phénomènes complexes. Nous dégageons ensuite les nuances qui se cachent derrière les termes de systèmes complexes et systèmes compliqués, car bien qu'ils puissent sembler équivalents, nous allons voir que ces termes évoquent des courants de pensée bien distincts. Enfin, nous nous intéressons à l'évolution dans le temps des systèmes.

2.1.1 *Systémique*

La définition de système que nous avons donnée vient de l'approche *systémique*, nourrie par les travaux de Wiener (1948), de Von Bertalanffy (1968), de Le Moigne (1977) ou encore de Forrester (1980). Elle permet d'appréhender des sujets *complexes* issus de domaines scientifiques différents et considère un système à la fois comme un tout et comme un ensemble de parties indépendantes dont l'interaction entraîne l'émergence d'un comportement global. C'est précisément sur ce point que l'approche *réductionniste*, répandue dans la communauté scientifique à l'époque où la systémique est née, trouve ses limites en décomposant le réel en éléments individuels simples sans faire apparaître l'effet de synergie du système global. Les

comportements individuels ne permettant pas d'expliquer le comportement de l'ensemble des composants du système. La propriété d'émergence a d'abord été mise en évidence avec l'approche *holistique*, qui considère que le système global ne se réduit pas uniquement à la somme des sous-systèmes qui le composent. Il existe en fait deux évolutions successives au sein du courant de pensée systémique, comme le souligne Ferber (1995). Même si la systémique a toujours considéré qu'un système est organisé en un réseau complexe d'interactions, c'est notamment avec Morin (1977), Atlan (1979) ou encore Varela (1989) que la notion d'auto-organisation et d'émergence à travers les interactions est mise en avant.

2.1.2 *Simple, compliqué et complexe*

Il est important de clarifier la notion de complexité lorsqu'on parle de systèmes complexes. Système *complexe* et système *compliqué* sont des notions souvent confondues. Or, elles sont fondamentalement différentes comme le distingue Le Moigne (1977). En effet, la notion de système *compliqué* s'apparente beaucoup à l'approche *réductionniste* : plusieurs sous-systèmes simples forment un système compliqué et un système est simple lorsque son fonctionnement et son utilisation sont des évidences au sens cartésien du terme. Un système *complexe*, lui, est lié à la pensée *systémique* : un système est complexe lorsqu'il devient difficile de saisir tous les aspects de son fonctionnement, ou qu'il est difficile d'anticiper son comportement.

Selon Le Moigne (1999), établir un modèle d'un système complexe permet d'aider à sa compréhension. Il est ainsi possible de comprendre à travers l'expérimentation. La complexité est davantage liée à l'intrication des interactions entre les sous-systèmes plutôt qu'à la façon dont ces derniers sont organisés et combinés. Si un système compliqué est décomposable en un ensemble d'éléments plus simples, un système complexe est également caractérisé par les relations qu'entretiennent ses sous-parties et la synergie qui en résulte. Par exemple, la complexité des interactions neuronales permettent de qualifier le cerveau de système complexe.

2.1.3 *Système dynamique*

Un système dynamique décrit la façon dont un système évolue dans le temps à travers certaines règles. Pour cela, la théorie des systèmes distingue généralement deux aspects d'un système :

1. sa *structure interne*, qui englobe son état et sa dynamique. L'état est tout ce qui compose le système et la dynamique est la manière dont évolue cet état.
2. son *comportement*, c'est-à-dire tout ce que le système offre d'observable d'un point de vue externe. Ce comportement est la résultante de la dynamique du système.

Un système dynamique est très souvent représenté par une boîte composée d'entrées et de sorties (figure 1.1). Lorsque la manière dont fonctionne la structure interne du système et toutes ses caractéristiques sont connues, le système est désigné par le terme de *boîte blanche*. À l'inverse, on parle de *boîte noire* quand le fonctionnement interne n'est pas connu et qu'il est uniquement possible de s'intéresser aux manifestations externes du système. Dans ce

cadre, la relation qui existe entre les entrées et les sorties du système est primordiale. C'est par cette relation qu'il est possible de caractériser son comportement.

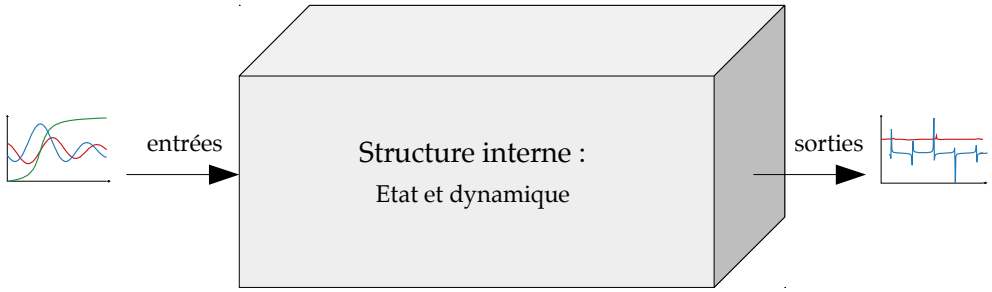


Figure 1.1 Représentation d'un système dynamique.

Un système dynamique évolue en fonction d'une base de temps. Sa structure interne est généralement définie par un ensemble de variables d'états qui représentent l'état interne dans lequel le système se trouve à un instant précis, et par un mécanisme d'évolution au cours du temps. Ce mécanisme est décomposable en deux phases :

1. le changement d'état des variables internes, défini par une fonction de transition d'état ;
2. la génération des sorties associées à l'état courant, définie par une fonction de sortie.

L'évolution de l'état du système en fonction du temps définit sa dynamique. Ses caractéristiques peuvent être vues comme dénombrables ou non. On parle alors d'évolution discrète ou continue. Selon que les changements d'états s'opèrent de manière discrète ou continue, que le temps prenne une infinité de valeurs dans un intervalle fini, ou encore que les valeurs des variables d'états soient finies ou non, un système est considéré comme discret ou non. La variation de l'intensité lumineuse perçue par un individu au cours de la journée est un phénomène continu. À l'inverse, le déclenchement d'une alarme à incendie est un phénomène discret.

2.2 *Modèle*

La notion de modèle est directement liée à notre manière d'appréhender le réel. Pour comprendre un système, nous construisons une abstraction de celui-ci. Cette abstraction est souvent simplifiée car conditionnée par l'aspect qui nous intéresse à propos du système. On retrouve cette idée dans l'ouvrage « Théorie Générale des Modèles » de Stachowiak (1973) qui estime qu'un modèle possède les trois propriétés suivantes :

1. La *représentation*. Les modèles sont toujours des modèles de quelque chose, des représentations d'*originaux* naturels ou artificiels qui peuvent eux-mêmes être des modèles.
2. La *réduction*. En général, les modèles ne possèdent pas la totalité des caractéristiques des originaux qu'ils représentent, mais seulement celles qui semblent pertinentes pour ceux qui les créent ou les utilisent.

3. Le *pragmatisme*. Les modèles ne correspondent pas de manière exacte à leurs originaux. Ils remplissent leur fonction de remplacement dans un contexte particulier. Un modèle est conçu dans un certain but, influencé par une pensée particulière à une certaine période.

Ainsi, un modèle est une représentation souvent incomplète voire inexacte d'un système, construit pour répondre à un certain nombre d'interrogations portées sur ce dernier. On retrouve cette idée dans beaucoup de définitions, notamment celle de Minsky (1965) :

> « *To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A.* »[2]

La notion de modèle est donc étroitement liée à l'observation du système, qui est conditionnée par les objectifs à atteindre, eux-mêmes déterminés par l'hypothèse formulée. Les hypothèses et les conditions d'observation du système forme ce que Zeigler et al. (2000) appelle le cadre expérimental du modèle.

Il est important de garder à l'esprit que les interrogations portées sur le système varient en fonction du concepteur. En effet, la discipline, les connaissances dont ce dernier dispose, le courant de pensée qu'il adopte, et le contexte historique font que le modèle a une légitimité limitée dans le temps. Comme Hill (2010) le fait remarquer très justement :

> *« Il s'agit là du processus même de l'avancement de la science. [...] L'ensemble des observations ultérieures du système réel en fonctionnement permettra la validation ou l'invalidation du modèle. »*

En ayant à l'esprit la définition de *système* que nous avons donnée dans la section 2, un modèle peut tout à fait être vu comme un système. En effet, un modèle peut interagir avec d'autres modèles ou avec son environnement. De ce point de vue, un modèle est un système possédant également les propriétés que nous venons d'énoncer. C'est un système conçu spécifiquement pour aider à répondre aux interrogations portées sur le système originel.

Selon Fishwick (1995), « modéliser, c'est décrire la réalité sous la forme d'un système dynamique, à l'aide d'un langage de description, à un certain niveau d'abstraction. » Cette définition permet de résumer les différentes étapes nécessaires à la conception d'un modèle :

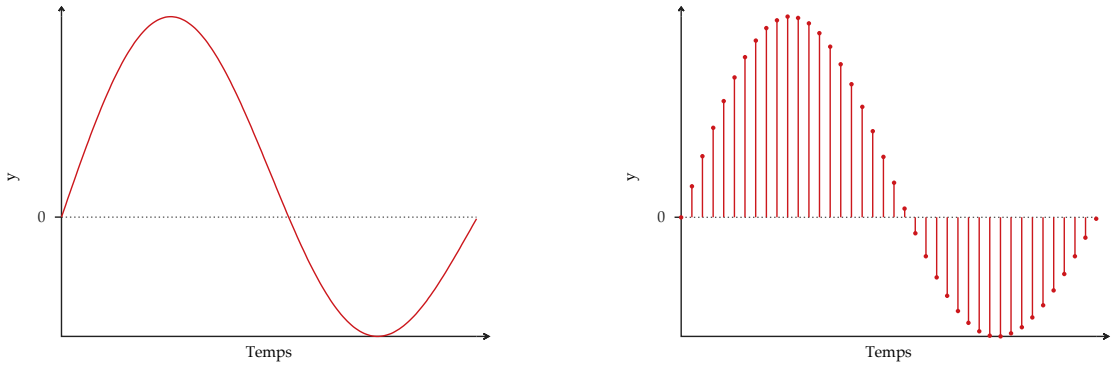
1. définition du cadre expérimental (Zeigler et al., 2000) ;
2. choix d'un paradigme de modélisation permettant de représenter le système réel ;
3. choix des éléments pertinents pour l'étude en fonction du cadre expérimental ;
4. décomposition du système en composants et identification de leurs interactions ;
5. choix d'un formalisme pour la description des états et des transitions d'états.

2.2.1 *Représentation du temps*

Si un système dynamique évolue en fonction d'une base de temps, il est logique de plonger également sa représentation - le modèle - dans le temps. Il existe plusieurs façons

2. « Pour un observateur B, l'objet A* est un modèle de l'objet A dans la mesure où A* aide B à répondre aux questions qu'il se pose sur A. » (traduit de Minsky (1965)).

de représenter l'écoulement du temps, que l'on peut classer en deux catégories : le *continu* et le *temps discret*. La figure 1.2 montre pour chacune d'elles l'évolution d'une variable d'état y au cours du temps.



(a) Modèle continu (b) Modèle à temps discret

Figure 1.2 Evolution d'une variable y dans un modèle à temps continu (a) et un modèle à temps discret (b).

Modèles continus Il y a une infinité de changements d'états dans un intervalle de temps fini (fig. 1.2(a)). Généralement, ce type de modèle repose sur des équations différentielles, ou sur des modèles à évènements discrets. Dans un modèle à évènements discrets, la représentation du temps est continue, mais les changements d'états s'effectuent de manière discrète, à des dates particulières.

Modèles à temps discrets Le temps est discrétisé, il évolue suivant une période de temps constante (figure. 1.2(b)). Les changements d'états s'effectuent de manière discrète, d'une période de temps à la suivante.

2.2.2 *La simulation*

Indissociable de la modélisation, la simulation désigne la mise en action des modèles. Elle plonge les modèles dans le temps et fait évoluer leurs états de manière automatique à travers un programme informatique dans le cadre d'une simulation numérique. L'analyse des résultats produits par la simulation permettra la compréhension du système étudié ou fournira une aide à la décision. Nous retenons deux définitions de la simulation qui soulignent la triade modèle-dynamique-analyse, trépied sur lequel la simulation repose. La première est proposée par Fishwick (1995) :

> « *Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output.* »[3]

La deuxième définition, proposée par Hill (1993), insiste sur les objectifs de la simulation :

3. « La simulation informatique est la discipline de conception d'un modèle d'un système réel ou théorique, de l'exécution du modèle par un ordinateur, et l'analyse de ses sorties. » (traduit de Fishwick (1995)).

« La simulation consiste à faire évoluer une abstraction d'un système au cours du temps afin d'aider à comprendre le fonctionnement et le comportement de ce système et à appréhender certaines de ses caractéristiques dynamiques dans l'objectif d'évaluer différentes décisions. »

Afin de synthétiser les différentes relations qui existent entre les concepts de système, de modèle et de simulation, nous reprenons l'illustration proposée par Zeigler et al. (2000) en figure 1.3, laquelle permet de visualiser l'activité de modélisation. L'hypothèse formulée

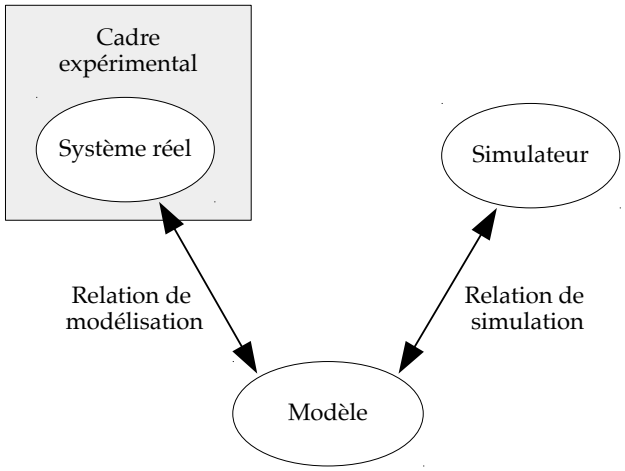


Figure 1.3 L'activité de modélisation, permettant d'illustrer les relations qu'entretiennent un système, son modèle et le simulateur (Zeigler et al., 2000).

à propos du système et les conditions d'observation de ce dernier forment la notion de cadre expérimental. Le modèle, qui est conçu à partir du cadre expérimental, entretient avec ce dernier une relation qui implique que le comportement du modèle soit en accord avec le comportement du système. Le simulateur permet de faire évoluer le comportement du modèle dans le temps, et le modèle permet de générer les données de simulation. La relation entre le simulateur et le modèle implique que le simulateur stimule correctement le modèle, en terme d'entrées et de temps. Les résultats produits par la simulation du modèle permettent de confronter les résultats numériques aux données observées du modèle, et potentiellement de les valider.

2.3 *Paradigme de modélisation*

Un paradigme de modélisation désigne une façon de concevoir la modélisation d'un système, qui correspond à un courant de pensée particulier, à des méthodes et des objectifs associés à un domaine particulier et dont l'utilisation fait l'objet d'un consensus, au moins pendant une certaine période. De manière générale, les paradigmes évoluent et de nouveaux paradigmes font leur apparition à mesure que nos connaissances évoluent. Dans le cadre de la M&S, il existe un certain nombre de paradigmes de modélisation qui permettent de concevoir un système complexe. Par exemple, les différentes représentations du temps au sein d'un modèle (*cf.* section 2.2.1) sont en réalité différents paradigmes de modélisation du temps.

Ici, nous nous intéressons à certains paradigmes en les étudiant d'un point de vue historique afin de montrer ce qui, petit à petit, a conduit à imaginer et à considérer le paradigme agent. Jusqu'à présent, nous n'avons pas abordé la notion de niveau d'analyse dans la modélisation. Le niveau d'analyse dépend de la « hauteur » à laquelle nous nous plaçons vis à vis du système à modéliser. En s'intéressant à une dynamique de groupe sans se soucier de l'individualité, on considère le modèle qui en résulte comme *macroscopique*. À l'inverse, en étudiant l'individu, le modèle se situera à une échelle *microscopique*. Bon nombre de paradigmes de modélisation dépendent directement de cette distinction.

2.3.1 *Modèles macroscopiques*

Historiquement, les modèles macroscopiques sont les premiers à faire leur apparition. Ils voient le jour au début du xx^{e} siècle, notamment avec les travaux de Hamer (1906) qui propose le premier modèle mathématique épidémiologique (SI) puis avec les travaux en écologie de Volterra (1926, 1928) et de Lotka (1925) sur les dynamiques de populations animales, plus connus aujourd'hui sous le nom de modèle « proie-prédateur ». Ces deux modèles permettent de suivre l'évolution de groupes de population au cours du temps en décrivant les relations mathématiques entre ces groupes. Ils se situent donc à une échelle macroscopique puisqu'ils s'intéressent à l'évolution d'une caractéristique globale du système. Cette approche classique offre une représentation continue du temps et des états du système, basée sur un ensemble d'équations différentielles. L'approche de conception de ce type de modèle macroscopique, conçu en analysant des propriétés observables du système, est dite « descendante » (ou *top-down* en anglais). Ces premiers modèles déterministes basés sur des formules mathématiques et qui dépendent de paramètres fixés sont les plus simples. En effet, la notion de déterminisme nous renvoie directement à la notion de causalité : pour les mêmes paramètres d'entrée, la sortie du modèle est toujours la même.

En réalité, concevoir de tels modèles n'est pas toujours possible. Rappelons que l'un des objectifs de la simulation, en dehors de l'aide à la décision qui n'intervient que sur des modèles dont le comportement est maîtrisé, est l'aide à la compréhension du fonctionnement et du comportement d'un système. Il est donc parfois nécessaire d'évaluer la sensibilité d'un modèle à certaines hypothèses ou paramètres, ou encore d'explorer son comportement. Ceci peut être motivé par le besoin de représenter l'incertitude dans le comportement du modèle, ou lorsqu'une partie des paramètres sont influencés par des variations aléatoires qu'il est impossible d'associer a une propriété du phénomène étudié. La raison pour laquelle il est parfois nécessaire de faire appel au hasard dans un modèle est à mettre en relation avec la notion de complexité. Dans ce cas, il est commode d'assimiler ces incertitudes en utilisant des variables stochastiques. Ainsi, les premiers modèles *stochastiques* font leur apparition avec les méthodes de simulation Monte Carlo (Metropolis et Ulam, 1949), qui permettent de simuler des modèles déterministes avec des paramètres ou des entrées stochastiques.

Malgré l'introduction de l'incertitude dans les modèles macroscopiques, Ferber (1995) met en évidence le fait qu'une telle approche, utilisée pour la modélisation d'entités individuelles comme avec un modèle proie-prédateur, ne permet pas de prendre en compte la complexité des interactions, ou le caractère individuel des actions. Cette problématique a

conduit à s'intéresser à un niveau d'analyse plus fin, en considérant l'entité individuelle plutôt que le groupe d'individus ; l'échelle microscopique plutôt que l'échelle macroscopique.

2.3.2 *Modèles microscopiques*

Le besoin d'approfondir les connaissances de certains systèmes au niveau microscopique associé à l'augmentation des capacités de calcul des ordinateurs à amené de nouveaux paradigmes de modélisation. À l'inverse des modèles macroscopiques qui considèrent le groupe dans son intégralité, les modèles microscopiques se concentrent sur la définition de l'entité elle-même, de son comportement et de ses interactions. Cette approche de conception, dans laquelle un système est analysé en se concentrant directement sur le comportement d'entités élémentaires et leur manière d'interagir plutôt que d'essayer de cerner le système dans sa globalité, est dite « ascendante » (ou *bottom-up* en anglais).

Automates cellulaires

Les automates cellulaires (Von Neumann et Burks, 1966), par exemple, représentent une première forme de paradigme de modélisation microscopique. Un automate cellulaire est constitué par un réseau d'automates à états finis, chaque automate a un ensemble de voisins finis et son état est calculé en fonction de l'état des automates situés dans son voisinage. Le comportement de l'ensemble des automates est identique, seule la situation spatiale diffère. Ils sont généralement représentés par des tuiles formant un pavage périodique à partir de formes géométriques. Ainsi, le comportement de l'automate et ses interactions sont au centre de la modélisation. À ce titre, il représente un paradigme intéressant pour l'étude de systèmes complexes, puisqu'il permet à partir de règles individuelles simples de faire apparaître des phénomènes émergents. Cependant, les automates cellulaires décrivent uniquement des entités spatiales fixes. Ainsi, ils sont parfaitement adaptés pour décrire une dynamique environnementale mais sont beaucoup moins intuitifs lorsqu'il s'agit de représenter des organismes vivants capables de se mouvoir au sein de leur environnement.

Modélisation individu-centrée

À l'inverse, le paradigme de modélisation individus-centrés, ou *Individual-Based Models* (IBM), traite l'individu comme une entité de modélisation à part entière. Ce paradigme prend racine en écologie et a été utilisé dès les années 1970, afin d'étudier la variabilité individuelle et a permis de répondre à la problématique des modèles macroscopiques, qui agrègent les variables d'états au niveau de la population (Grimm, 1999). Il s'agit donc de modéliser le comportement et les caractéristiques individuelles et de simuler l'ensemble des individus en interaction. Ainsi, chaque entité est un individu au sens biologique, avec ses caractéristiques intrinsèques. À l'inverse d'un automate cellulaire, la notion d'espace n'est pas centrale dans ce paradigme. Les caractéristiques d'un individu peuvent comprendre des propriétés relatives à sa position dans l'espace, mais pas de façon systématique.

Modélisation orientée agent

Le paradigme agent, qui se situe au centre de nos travaux, considère également l'agent comme une entité de modélisation à part entière. En cela, il rappelle le paradigme individus-

centrés. Pourtant, les modèles orientés agents sont bien distincts des modèles individus-centrés. En effet, les IBM sont intimement liés aux domaines de la biologie et de l'écologie et s'attachent essentiellement à modéliser des populations d'individus tandis que le paradigme agent permet de représenter d'autres types de systèmes. Les systèmes multi-agents peuvent dans certains cas être considérés comme des systèmes individus-centrés, mais pas systématiquement, comme le souligne Millischer (2000) :

> « *Un SMA est un IBM dès lors que les agents mis en œuvre modélisent des individus réellement observables. Un IBM est un SMA dès lors qu'il est spatialement explicite, qu'il cherche à modéliser des interactions entre plusieurs individus, et qu'il tient compte de l'environnement de ces individus, ce qui n'est pas forcément le cas.* »

D'autres chercheurs sont arrivés à la même conclusion, et c'est notamment celle de Duboz (2004), qui affirme :

> « *Pour les écologues, les SMAs sont perçus comme une technique pour implémenter des IBMs. Pour un chercheur en informatique, les IBMs sont vus comme une utilisation possible des SMAs.* »

2.4 *Le paradigme agent*

Les systèmes multi-agents (SMA) constituent un paradigme tout à fait pertinent pour aborder la conception de systèmes complexes, et cela pour des domaines d'applications variés. La diversité des thématiques de recherche ouvertes par les systèmes multi-agents, associée à l'engouement que ce paradigme a suscité, a entraîné des visions différentes à propos des définitions des concepts mêmes du paradigme. Nous proposons dans cette section de présenter les nuances associées à ces concepts tout en donnant des définitions fondamentales qui semblent aujourd'hui faire consensus.

2.4.1 *Qu'est-ce qu'un agent ?*

La notion d'intelligence est à la base de la définition d'un agent. Il existe deux visions différentes de l'intelligence d'une entité au sein d'un SMA. Si une entité est capable d'agir de manière réfléchie pour résoudre un problème, on parle d'approche *cognitive*. Si elle se résume à un système simple doté de réflexes, on parle d'approche *réactive* :

- L'approche *cognitive* vient essentiellement du domaine de l'IA distribuée, qui s'appuie sur les travaux menés en sociologie pour ajouter une dimension interactionnelle aux outils classiques de l'intelligence artificielle à travers des problématiques de communication, de coopération ou de négociation (Bond et Gasser, 1988). Chaque entité dispose de l'ensemble des informations et du savoir-faire nécessaire à l'accomplissement de sa tâche (Ferber, 1995).
- À l'inverse, l'approche *réactive* avance qu'une entité n'est pas nécessairement capable de raisonnement pour faire émerger un comportement global intelligent. Dénuée de représentation de son environnement, sans buts explicites et incapable de planifier quoi que ce soit, ce type d'entité est seulement régie par des mécanismes « réflexes » face à certaines situations. Pourtant, l'ensemble de leurs actions et de leurs interactions

indirectes permettent de résoudre des problèmes complexes. Les travaux qui ont été effectués en éthologie par Drogoul (1993) sur les colonies de fourmis sont un bon exemple d'organisation émergente (Corbara et al., 1993). L'origine de cette approche vient en partie de l'IAD, à travers les travaux de Foerster (2003) sur l'auto-organisation, du domaine de la Vie Artificielle (Langton, 1989), de la cybernétique (Wiener, 1948) et de la robotique (Brooks, 1983).

Retenir une seule définition du concept d'*agent* conciliant l'approche cognitive et l'approche réactive est difficile. Néanmoins, Wooldridge et Jennings (1995) mettent en évidence des caractéristiques communes à travers les propriétés suivantes :

- l'*autonomie* : les agents évoluent sans intervention directe de l'Homme ou d'autres agents, et disposent du contrôle de leur état et de leurs propres actions.
- la *sociabilité* : l'agent est capable d'interagir avec d'autres agents (et éventuellement des humains) grâce à un mode de communication.
- la *réactivité* : l'agent perçoit son environnement (qui peut être physique, formé par un ensemble d'autres agents, ou un réseau social) et réagit aux changements qui s'y produisent.
- la *proactivité* : les agents ne se contentent pas de réagir en fonction de leur environnement, ils sont aussi capables de manifester un comportement motivé par des buts.

La figure 1.4 donne une vision schématique de l'agent. Celle-ci est à mettre en parallèle avec une définition d'agent assez minimaliste, proposée par Weiss, Gerhard (1999) :

> *« Agents are autonomous, computational entities that can be viewed as perceiving their environment through sensors and acting upon their environment through effectors. »* [4]

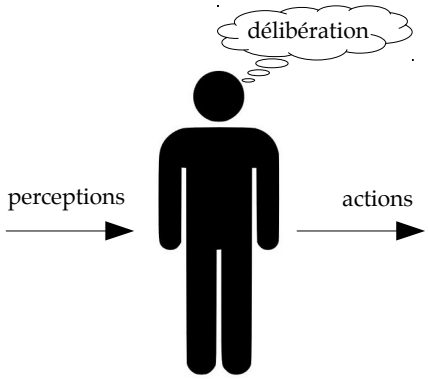


Figure 1.4 Représentation d'un agent, *percevant* son environnement et agissant sur son environnement à travers ses *actions* selon le processus à 3 phases *perception*, *délibération* et *action* identifié par Pnueli (1986).

D'après cette définition, similaire à celle proposée par Russel et Norvig (2002), et des caractéristiques des agents que nous avons énumérées, il est possible d'établir un certain nombre de points communs entre un système dynamique et un agent. À l'instar d'un système dynamique, un agent est en interaction avec ses pairs et avec son environnement.

4. « Les agents sont des entités computationnelles autonomes que l'on peut considérer comme percevant leur environnement à travers des capteurs et agissant sur leur environnement à travers des effecteurs. » (traduit de Weiss, Gerhard (1999)).

Nous accompagnons la définition de Weiss, Gerhard (1999) d'une définition plus complète, proposée par Ferber (1995), qui considère un agent comme « *une entité physique ou virtuelle :*

1. *qui est capable d'agir dans un environnement ;*
2. *qui peut communiquer avec d'autres agents ;*
3. *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser) ;*
4. *qui possède des ressources propres ;*
5. *qui est capable de percevoir (mais de manière limitée) son environnement ;*
6. *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune) ;*
7. *qui possède des compétences et offre des services ;*
8. *qui peut éventuellement se reproduire ;*
9. *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.* »

Ferber (1995) accompagne cette définition générale de deux autres définitions plus spécialisées, destinées aux agents « purement communiquant » (école cognitive) et aux agents « purement situés » (école réactive).

2.4.2 *Qu'est-ce qu'un environnement ?*

Le terme d'environnement dans un système multi-agents peut faire référence à plusieurs notions distinctes. Weyns et al. (2005b) en distinguent trois :

- l'infrastructure *physique* (matérielle ou réseau) sur laquelle repose l'exécution d'un agent ;
- l'environnement d'*exécution logiciel* (système d'exploitation, middleware) sur lequel les agents sont exécutés ;
- l'environnement *d'application*, qui est le modèle conceptuel d'environnement tel qu'il est défini au sein d'une plateforme SMA.

Cette distinction de l'environnement peut être appliquée à un logiciel conçu à travers le paradigme agent et déployé pour rendre service à des utilisateurs. Dans le cadre de la simulation multi-agent, Klügl et al. (2005) distinguent plus simplement l'*environnement de simulation* de l'*environnement simulé*. L'environnement de simulation fournit l'infrastructure permettant d'exécuter la simulation tandis que l'environnement simulé fait parti du modèle du système multi-agent, il permet aux agents simulés de « vivre » dans une abstraction d'un environnement original. C'est à cette dernière notion, celle d'environnement *simulé* (ou d'*application*), à laquelle nous faisons référence dans toute la suite de ce document lorsque nous employons le terme d'environnement, sauf mention particulière.

L'environnement forme avec l'agent un couple indivisible, comme le soulignent Weyns et al. (2005b), il est un composant essentiel d'un système multi-agents. D'après Odell et al.

(2003a), les agents ont besoin d'exister et de fonctionner au sein d'un environnement. La définition de Weyns et al. (2006), que nous retenons vient appuyer ce postulat :

> « *The environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources.* »[5]

Les auteurs accompagnent leur définition des précisions suivantes. Premièrement, l'environnement est une brique indépendante avec des responsabilités bien établies. Deuxièmement, il est une entité essentielle quel que soit le SMA : il n'y a pas de SMA sans environnement, c'est ce dernier qui permet l'existence des agents. Son rôle d'intermédiaire le définit comme une entité active à laquelle on peut assigner des tâches spécifiques.

Deux grands types d'environnements d'application sont généralement distingués (Odell et al., 2003a) :

- les environnements *physiques*, qui fournissent les règles et contraintes qui régissent et permettent l'existence des agents et de ressources éventuelles ;
- les environnements *sociaux*, qui permettent d'organiser les agents pour former une société virtuelle, où chaque agent peut jouer des rôles au sein de groupes préétablis.

Dans les deux cas, l'environnement représente un support de communication entre agents, le premier est plutôt vecteur de communication indirecte tandis que le second est plutôt vecteur de communication directe. Ces deux types d'environnements nous renvoient une fois encore sur des courants de pensée différents : les SMA centrés agents (ACMAS[6]) et les SMA centrés organisation (OCMAS[7]). Les approches centrées agents se concentrent uniquement sur l'aspect individuel de l'agent et de ses actions, tandis que dans les approches organisationnelles, les agents sont organisés en formant des groupes et des hiérarchies.

Un certain nombre de propriétés mises en évidence par Russel et Norvig (1995) permettent de caractériser les environnements. Nous en présentons ici quelques unes. L'environnement peut être :

- *totalement observable* si les agents ont accès à l'ensemble de l'état de l'environnement ou *partiellement observable* si au contraire, les agents n'ont qu'une vision restrictive de l'état de l'environnement ;
- *déterministe* si le changement d'état de l'environnement est uniquement déterminé par son état courant et les actions des agents ou *stochastique* dans le cas contraire ;
- *dynamique* si l'environnement peut évoluer de manière endogène ou *statique* s'il change d'état uniquement à travers les actions des agents ;
- *discret* s'il existe un nombre fini de perceptions ou d'actions possibles ou *continu* à l'inverse. À titre d'exemple, un environnement cellulaire est discret tandis qu'un espace euclidien est continu.

5. « L'environnement est une abstraction de première classe qui fournit les conditions environnantes pour que les agents puissent exister et qui sert d'intermédiaire d'interaction entre les agents et d'accès aux ressources. » (traduit de Weyns et al. (2006)).
6. ACMAS : *Agent-Centered Multi-Agent Systems*.
7. OCMAS : *Organisation-Centered Multi-Agent Systems*.

2.4.3 *Le système multi-agents*

De manière simplifiée, un système multi-agents est composé d'un ou plusieurs environnements et d'un ensemble d'agents évoluant au sein du/des environnement(s) dans le(s)quel(s) ils sont plongés. La relation entre l'environnement et les agents est illustrée par la Figure 1.5.

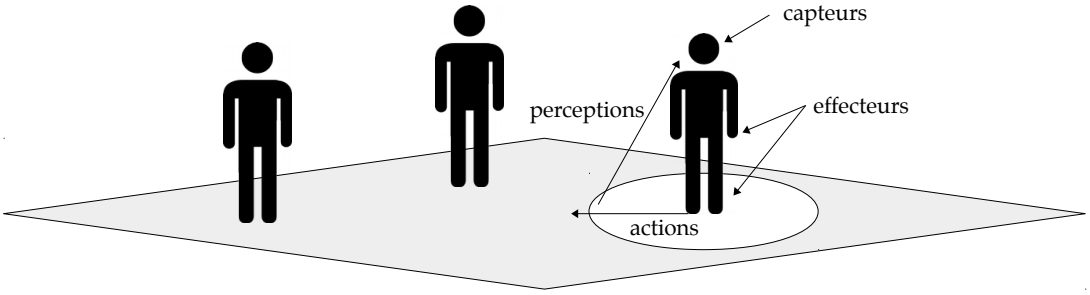


Figure 1.5 Représentation schématique d'un système multi-agents, inspirée de Russel et Norvig (2002).

Ferber (1995) propose la définition suivante : « *un système multi-agents est un système composé des éléments suivants :*

1. *Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.*
2. *Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que pour tout objet, il est possible à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.*
3. *Un ensemble A d'agents qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système.*
4. *Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.*
5. *Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer, et manipuler des objets de O.*
6. *Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.* »

Contrairement à ce qui est suggéré dans cette définition, rappelons que l'environnement n'est pas simplement un espace doté d'une métrique. Weyns et al. (2006) considèrent que l'environnement est une entité de *structuration* des agents sous trois différentes formes :

- la structuration *physique* qui fait référence à la dimension spatiale, à la topologie et à la distribution ;
- la structuration des *communications* qui fait référence aux infrastructures permettant aux agents de communiquer entre eux ;
- la structuration *sociale* qui fait référence à la manière dont les agents sont organisés pour former une société virtuelle.

Notons que Soulié (2001), donne la possibilité pour un agent d'exister et d'évoluer au sein de plusieurs environnements, contrairement à la définition que nous venons de donner.

Ainsi, un agent peut appartenir à la fois à un environnement physique et par ailleurs exister au sein d'un environnement social, au même titre que nous pouvons agir physiquement à travers notre corps tout en ayant la possibilité d'entretenir une présence sur Internet. Cette vision du système multi-agent peut également permettre de faire coexister deux environnements physiques dont les lois ou les propriétés diffèrent (*e.g.* milieu aquatique et milieu atmosphérique).

Comme nous allons le voir dans la section suivante, les concepts développés dans cette section laissent beaucoup de libertés lorsqu'il s'agit de concevoir et d'implémenter un système multi-agents. C'est dans un contexte d'ingénierie logicielle que nous nous plaçons maintenant, en étudiant les différents métamodèles proposés pour la conception d'un SMA.

3 *Conception des systèmes multi-agents*

Tout un pan de la communauté SMA s'est particulièrement intéressé à l'ingénierie logicielle orientée agent. Le paradigme agent permet de faciliter la compréhension et la modélisation de systèmes complexes (Gleizes et al., 2009). Plusieurs méta-modèles et méthodologies permettant de faciliter l'analyse, la conception et l'implémentation des systèmes multi-agents ont été proposés dès la fin des années 90.

Avant de parler de manière plus détaillée de ces différents travaux, nous proposons de présenter un état de l'art organisé selon les aspects de conception identifiés par l'approche VOYELLES (Demazeau, 1995) et par l'approche INGENIAS (Gomez-Sanz et Pavón, 2002) : l'*agent*, l'*environnement*, l'*interaction* et l'*organisation*. Les objectifs de l'approche cognitive et l'approche réactive étant distincts, ils ont conduit à des architectures différentes pour chacun de ces axes.

3.1 *Architecture interne de l'agent*

Pnueli (1986) décrit un agent comme un processus composé de trois phases : *perception*, *délibération* et *action* (*cf.* Figure 1.4). En fonction de ce que l'agent perçoit, l'agent choisit l'action à effectuer pendant sa phase de délibération. Selon le type de SMA modélisé, il existe différentes architectures d'agent permettant de matérialiser cette phase de délibération. Nous présentons ici deux architectures : l'architecture de subsomption et l'architecture *Beliefs-Desire-Intentions* (BDI). La première est surtout utilisée pour modéliser des agents réactifs tandis que la seconde est utilisée pour modéliser des agents cognitifs.

3.1.1 *Architecture de subsomption*

L'approche réactive considère que les agents n'ont pas de représentation explicite de l'environnement. Le comportement de l'agent est uniquement décrit par des mécanismes directs associant les perceptions aux actions. Leurs buts ne sont pas explicites, ils les réalisent de manière « instinctive » à travers leurs « réflexes ».

L'architecture la plus répandue pour définir le comportement d'un agent réactif est l'architecture de *subsomption*, proposée par Brooks (1983). Chaque agent est capable d'effectuer un ensemble de tâches, plus ou moins prioritaires. Chacune d'elles est déclenchée par une perception particulière. Un exemple de ce processus est donné par la figure 1.6 à travers le comportement d'un agent fourmi.

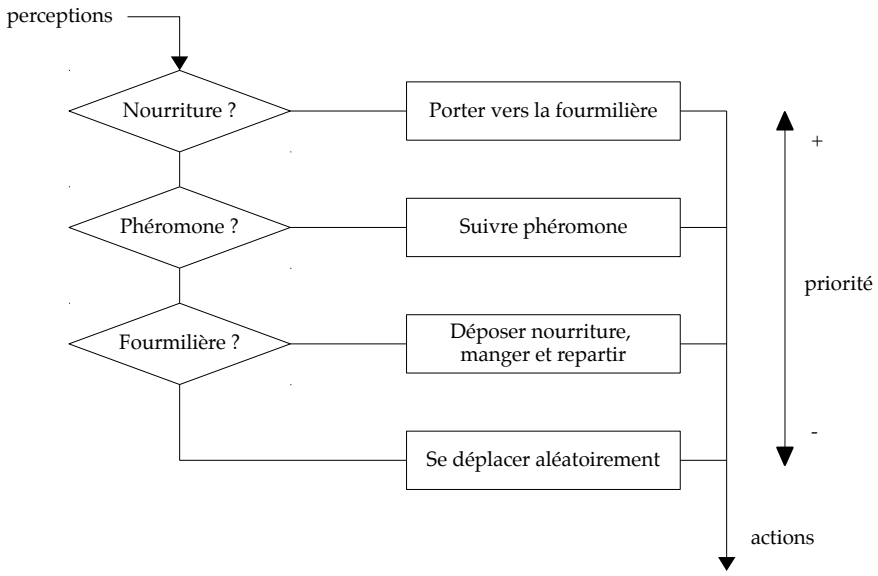


Figure 1.6 Exemple de comportement d'un agent « fourmi » à travers l'architecture de subsomption sous forme de diagramme d'activité.

Malgré le fait que l'agent fourmi n'a aucune connaissance de son environnement, et n'agit qu'en réaction à la perception, le comportement d'un ensemble d'agents permet de résoudre des problèmes complexes à travers l'émergence d'une intelligence collective (Drogoul, 1993).

3.1.2 *Architecture Beliefs-Desire-Intentions*

Également basée sur le processus décisionnel à trois phases, l'approche cognitive suggère une architecture d'agent qui permet une représentation explicite de l'environnement et des agents qui l'entourent. Ainsi l'agent est capable de raisonner à partir de ses connaissances avant de prendre une décision. L'une des architectures les plus connues est celle de Rao et Georgeff (1992), appelée *Beliefs-Desire-Intentions* (BDI) dans laquelle un agent agit rationnellement à partir de ses croyances sur l'état du monde, de ses désirs et de ses intentions. Cette architecture est basée sur la théorie du raisonnement proposée par Michael Bratman, que les auteurs ont formalisée en logique temporelle. De manière simplifiée, le processus de délibération, représenté par la Figure 1.7, fonctionne de la manière suivante : à partir des perceptions de son environnement, l'agent peut réviser ses croyances sur l'état du monde qui l'entoure. Une seconde étape consiste à prendre en compte les nouvelles croyances et les intentions courantes de l'agent afin de produire des désirs. L'agent produit ensuite de nouvelles intentions en fonction de ses croyances, de ses intentions courantes et de ses désirs.

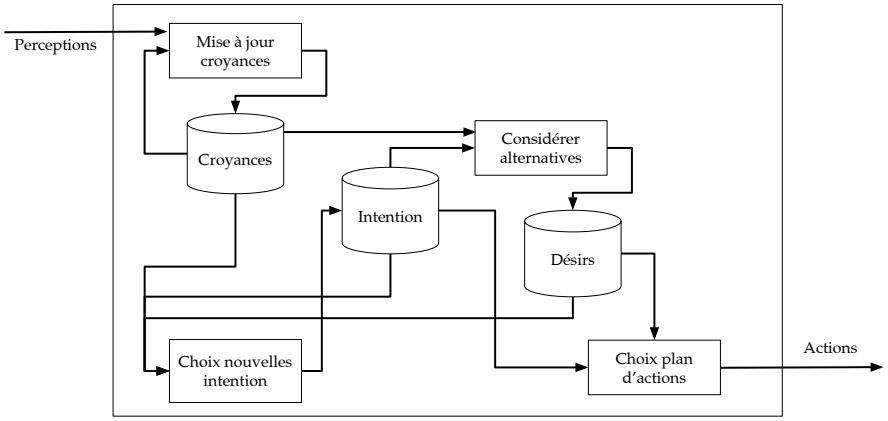


Figure 1.7 Représentation schématique de l'architecture BDI, basée sur l'interpréteur BDI donné dans Rao et Georgeff (1992).

Enfin, les actions sont produites à travers la génération d'un plan d'actions, basé sur les désirs et les intentions de l'agent.

Il existe également des approches hybrides qui mêlent l'approche réactive et l'approche cognitive. Certaines actions sont alors effectuées par une partie réflexe tandis que d'autres actions sont effectuées de manière plus « réfléchie ». Ces deux approches peuvent tout à fait être complémentaires et permettent de concevoir une architecture d'agent adaptée à certaines problématiques pour gagner en efficacité. L'architecture INTERRRAP (Müller, 1996) en est un exemple.

3.2 *Environnement*

Comme nous l'avons vu dans la section 2.4.2, un système multi-agents implique forcément la présence d'un environnement dans lequel les agents évoluent. Les rôles de l'environnement, en revanche, n'ont pas toujours été clairement identifiés au sein de la communauté SMA. Les travaux d'état de l'art de Weyns et al. (2005b) permettent à la fois de les synthétiser et d'établir l'importance de l'environnement.

Bien que nous ayons déjà mentionné l'existence d'environnement sociaux, nous restreignons volontairement cette sous-section à l'étude des propriétés générales de l'environnement. Nous traitons les aspects sociaux dans la section suivante.

En ce qui concerne la conception de l'environnement, Weyns et al. (2005b) distinguent les aspects structurels de l'environnement et ses aspects dynamiques. Au même titre, nous développons dans un premier temps les aspects structurels (topologie, distribution, entités passives) avant d'aborder les aspects dynamiques (dynamique endogène, perception, action).

3.2.1 *Aspects structurels*

Dans le cadre de la simulation, l'environnement est un espace virtuel qui a un rôle de structuration physique pour les entités qui le composent (agents et entités passives). Cette notion évoque donc les notions de métrique, de topologie ou encore de distribution. Pour que le modélisateur puisse utiliser un environnement avec une topologie adaptée à sa problématique, plusieurs formes de structuration spatiale ont vu le jour au sein des plate-formes de simulation multi-agents. Il est possible de classer ces différentes topologies selon la propriété environnementale *discret* / *continu* distinguée par Russel et Norvig (1995).

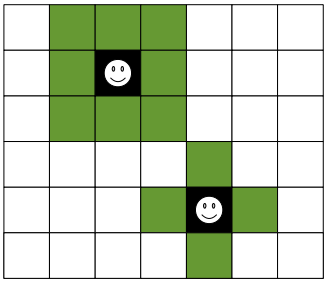
Environnements discrets

Avec une approche discrète, l'environnement est formé par un ensemble de zones qui entretiennent une relation de voisinage entre elles. La zone est l'unité spatiale qui définit la portée de la perception et de l'action. Nous donnons ici deux exemples concrets d'environnements discrets :

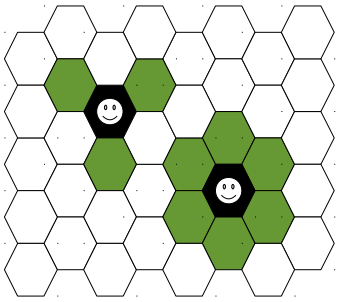
- les environnements *cellulaires*, très courant dans le domaine des automates cellulaires, est formé par une grille de cellules généralement représentées par des *tuiles* formant un pavage périodique à partir de formes géométriques. Les cellules peuvent être organisées en une ou plusieurs dimensions ($2D$, $3D$). La figure 1.8 illustre trois environnements cellulaires à deux dimensions formés avec un pavage différent. Pour chacun d'eux, nous illustrons différents types de relations de voisinage.
- les environnements de type *graphe*, formés par un ensemble de nœuds liés entre eux par des arêtes (orientées ou non), a l'avantage de pouvoir représenter une topologie adaptée à la notion de réseau. Bien que nous parlons ici des environnements physiques, le graphe est une structure de données qui peut avoir d'autres utilités dans le cadre d'un SMA, notamment dans l'état interne d'un agent pour représenter un réseau social par exemple. Quand on y réfléchit, n'importe quel environnement cellulaire pourrait être représenté à l'aide d'un graphe. L'avantage de ce dernier réside néanmoins dans la possibilité d'avoir un rayon d'action/perception conditionné par les arêtes qui unissent l'ensemble des sommets. La figure 1.9 donne la représentation d'un environnement graphe dans lequel la relation de voisinage est matérialisée par les arêtes et l'unité du rayon est le nœud.

Environnements continus

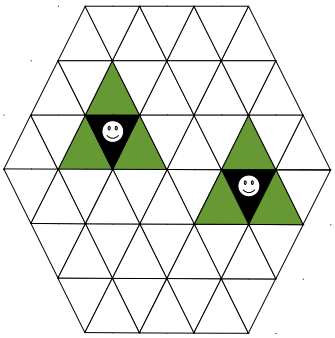
À l'inverse de l'approche discrète, l'environnement continu permet une granularité beaucoup plus fine puisqu'il ne restreint pas la portée de la perception et de l'action à une unité déterminée par la topologie de l'environnement. Cette approche, bien que plus complexe à mettre en œuvre permet de traiter de manière individuelle le rayon d'action/perception, et éventuellement d'avoir une portée propre à chaque sens d'un même individu. Imaginons par exemple que deux classes d'agents soient modélisées au sein du même système multi-agents, l'une de ces classes possédant une acuité visuelle plus importante que l'autre. Un environnement continu permettra alors une plus grande justesse dans la modélisation. Il permettra également de prendre en compte plus aisément le fait qu'un agent ait de meilleures facultés auditives que visuelles. Dans un environnement cellulaire, certains types de voisinage d'environnements cellulaires permettent également



(a) Environnement cellulaire formé de tuiles carrées. Pour chaque agent, un type de voisinage possible est représenté par les cellules vertes autour de lui. De gauche à droite, un voisinage de Moore d'ordre 1 (8 voisins) et un voisinage de von Neumann (4 voisins).



(b) Environnement cellulaire formé de tuiles hexagonales. Les voisinages possibles avec cette forme de tuile sont illustrés avec les cellules vertes autour des agents. 3 ou 6 voisins immédiats



(c) Environnement cellulaire formé de tuiles triangulaires. Ici, le rayon perception/action est uniquement constitué de 3 voisins immédiats.

Figure 1.8 Exemples de représentation d'environnements cellulaires à deux dimensions avec des tuiles différentes (carrées (a), hexagonales (b) et triangulaires (c)), permettant plusieurs possibilités de rayon d'action et de perception pour un agent.

d'étendre le rayon de perception/action. C'est le cas du voisinage de Moore qui peut passer de 8 voisins immédiats à 24 voisins, voire d'avantage. Il permet donc également de représenter des rayons plus ou moins importants, mais uniquement avec des multiples d'unités.

Un environnement continu peut être représenté à l'aide d'un espace euclidien. La plate-forme GAMA (Grignard et al., 2013), par exemple, utilise un espace euclidien pour représenter l'environnement. La Figure 1.10 est une capture d'écran de l'environnement d'un modèle épidémiologique (SI) réalisé avec GAMA, dans lequel les agents se déplacent au sein d'une ville, de bâtiments en bâtiments à travers les routes. Bien que nous ayons classé les environnements de type graphe dans la catégorie des environnements discrets, un graphe peut tout à fait être considéré comme environnement continu à partir du moment où des contraintes physiques sont appliquées aux arêtes. Ces contraintes peuvent se matérialiser avec des poids représentant une distance associée à une vitesse de déplacement dans le milieu par exemple.

Topologie hybride et distribution

Il est tout à fait possible de combiner les avantages d'un environnement discret et d'un environnement continu. Considérons un environnement cellulaire dans lequel chaque zone est considérée comme un espace euclidien qui entretient une relation de voisinage avec d'autres zones. Un environnement hybride de ce type permet à certains agents de profiter de la facilité que représente l'unité d'un environnement discret et à d'autres d'avoir une granularité plus fine. La plate-forme GAMA par exemple, offre la possibilité de représen-

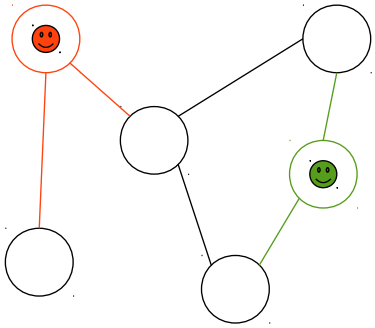


Figure 1.9 Exemple d'environnement de type graphe. Chaque agent est situé dans un nœud, et peut se déplacer/percevoir en fonction des arêtes qui lient les différents nœuds du graphe.

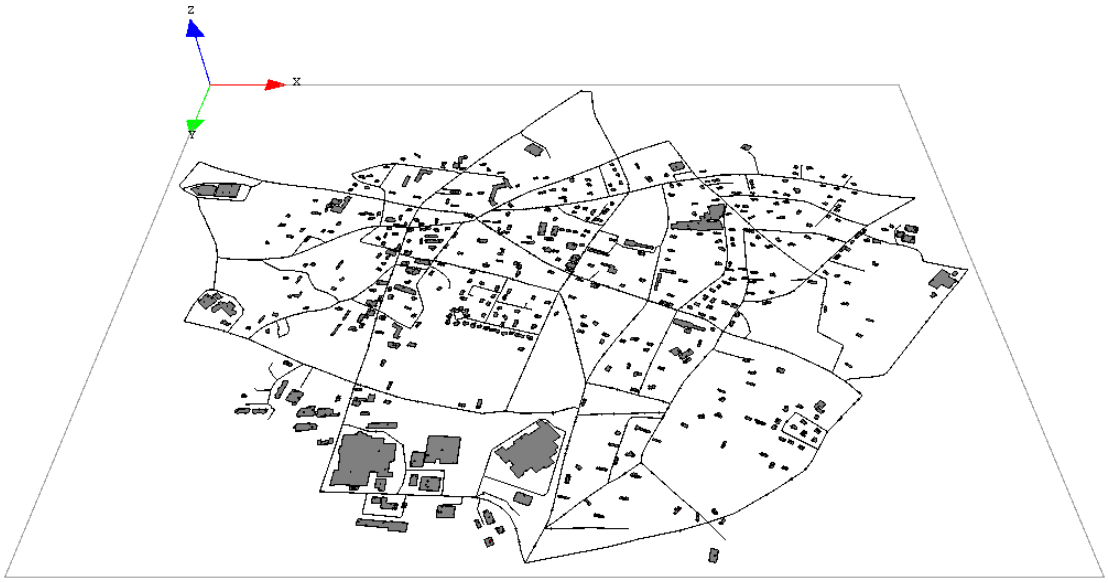


Figure 1.10 Capture d'écran de l'environnement continu d'un modèle épidémiologique (SI) (Hamer, 1906) réalisé avec GAMA (Grignard et al., 2013).

ter un environnement cellulaire superposé à un environnement continu. Chaque cellule correspond à une zone de l'environnement euclidien.

Le découpage en zone offre également la possibilité de distribuer l'environnement avec une plus grande facilité. En effet, Soulié (2001) identifie plusieurs catégories d'environnements. Parmi elles, il décrit les environnements *centralisés* formés par une structure unique et les environnements *distribués* formés par une structure discrétisée dont chaque élément peut être considéré comme un environnement centralisé.

Ressources

Au delà de son rôle de structuration physique des agents, l'environnement peut comprendre des ressources exploitables par les agents. Ferber (1995) précise qu'il existe un ensemble d'objets situés dans l'environnement qui « peuvent être perçus, créés, détruits et modifiés par les agents. » Une ressource peut par exemple représenter un obstacle, de la

nourriture, une clé servant à ouvrir une porte, *etc*. Le contrôle d'accès à ces ressources fait partie des responsabilités de l'environnement et dépend généralement de plusieurs facteurs : le voisinage de l'agent à la ressource ou les capacités de l'agent par exemple. Notons que les travaux de Soulié (2001) proposent de séparer de manière explicite le « corps » et « l'esprit » de l'agent pour des raisons conceptuelles et pratiques. La partie physique de l'agent peut alors être considérée comme n'importe quelle ressource de l'environnement. Cette idée de séparer l'agent de sa représentation physique se retrouve dans un certain nombre de modèles conceptuels tels que ceux proposés par Ferber et al. (2005) ou encore Dinu et al. (2012).

En dehors des objets situés, les ressources peuvent prendre d'autres formes. Les champs par exemple, permettent de porter une information ayant différentes valeurs en chaque « point » de l'espace. Ils peuvent par exemple être utilisés pour représenter un courant maritime sous forme de vecteurs ou encore pour représenter la nature d'un sol. Les champs sont également très adaptés pour représenter des zones attractives ou répulsives pour un agent. Un champ de phéromones peut par exemple être utilisé pour attirer des agents vers une zone comportant de la nourriture.

3.2.2 *Aspects dynamiques*

Les aspects dynamiques de l'environnement sont à la fois liés aux agents, plus particulièrement à leur activité de perception et à leurs actions, et à l'activité endogène de l'environnement, générée par la relation entre les lois de l'environnement et les ressources qui le composent.

Dynamique endogène

La dynamique endogène est liée à la propriété environnementale *statique / dynamique* identifiée par Russel et Norvig (1995). L'environnement est dit dynamique lorsque l'état de l'environnement évolue indépendamment des actions réalisées par les agents. Si les ressources de l'environnement sont différentes des agents dans la mesure où ce ne sont pas des entités autonomes et proactives, elles peuvent néanmoins évoluer en fonction des lois de l'environnement. Ainsi, un environnement qui tient compte du processus d'érosion venant dégrader le relief au fil du temps est un environnement disposant d'une dynamique endogène, au même titre qu'un environnement tenant compte du processus d'évaporation et de diffusion de phéromones.

Perception

La phase de perception d'un agent implique que l'environnement puisse être observé (Weyns et al., 2005b). Si Russel et Norvig (1995) indiquent qu'un environnement peut être totalement observable ou partiellement observable, la majorité des définitions s'accordent pour restreindre la perception de l'agent à une partie de l'environnement.

Le voisinage de l'agent est utilisé pour restreindre ce qu'il peut percevoir. Celui-ci est calculé en fonction de la position qu'occupe l'agent au sein de son environnement. Michel

(2007a) souligne l'importance de cette restriction, qu'il appelle la *contrainte de localité* de l'environnement. Le voisinage d'un agent peut varier en fonction de la topologie de l'environnement mais peut également varier en fonction de la portée de perception de l'agent. Cette portée peut être matérialisée par une forme dans un environnement continu (cercle, triangle, ...) ou par le nombre d'unités dans un environnement discret (voisinage de Moore d'ordre 2 par exemple). En plus des propriétés des agents, la perception peut également dépendre des propriétés de l'environnement. Par exemple, Weyns et Holvoet (2004) proposent d'introduire des lois de perception. Ces dernières permettent de refléter certaines règles environnementales, comme l'impossibilité pour un agent de voir les ressources situées derrière un mur.

Action

La gestion des actions réalisées par les agents, particulièrement dans un environnement simulé, est un aspect de conception à la fois important et difficile à traiter. Etant donné qu'une action peut conduire à un changement d'état de l'environnement, son traitement est une responsabilité qui incombe à l'environnement (Weyns et al., 2005b). Cependant, la plupart des plateformes de simulation orientées agents, comme NetLogo par exemple (Wilensky, 1999), octroient cette responsabilité à l'agent en lui permettant de modifier l'état de l'environnement. Cette manière de concevoir le traitement de l'action est problématique dès lors que plusieurs agents peuvent agir en parallèle au sein du même environnement. En simulation, l'activation des agents étant soumise à une horloge virtuelle, il est courant que plusieurs agents puissent être activés au même temps virtuel. Nous reviendrons de manière plus détaillée sur cette problématique dans la suite de ce chapitre afin d'étudier les conséquences d'une telle conception et les différents modèles permettant d'y apporter une solution.

Au-delà de la problématiques des actions concurrentes, se pose celle de la validité d'une action. Lorsqu'un agent souhaite réaliser une action, il s'agit de déterminer si celle-ci est valide pour le contexte courant, ou plus généralement pour les lois de l'environnement. Prenons pour exemple un SMA dont l'environnement doit répondre aux lois de la physique. Un agent souhaite déplacer une ressource dont la masse est bien plus importante que celle de l'agent. Si celui-ci peut modifier l'état de l'environnement, il lui incombe d'effectuer le calcul permettant de savoir si il est capable de déplacer l'objet. Pire, il peut éventuellement « tricher » et ne pas tenir compte de la contrainte physique. Afin d'éviter cette problématique, l'apparition de métamodèles permettant de matérialiser les lois de l'environnement ont fait leur apparition. D'autres métamodèles empêchent les agents de modifier directement l'état de l'environnement, afin de respecter ce que Michel (2007a) appelle la contrainte d'intégrité de l'environnement.

3.3 *Interaction*

L'interaction, qui désigne une action réciproque, permet à deux systèmes ou deux agents d'échanger des informations. Dans le cadre des systèmes multi-agents, l'interaction est la base sur laquelle repose les communications entre agents. Elle est primordiale pour

qu'ils puissent coopérer, réaliser des tâches en commun ou encore négocier pour régler une situation de conflit. C'est précisément grâce aux interactions qu'un système (multi-agents ou pas) est capable de produire un effet synergique.

Les différences de conception des approches réactives et cognitives n'entraînent pas uniquement des architectures différentes d'agents. Comme nous avons pu le voir dans la section 3.1, elles ont également un impact sur la manière dont les agents peuvent s'échanger des informations. Deux types d'interaction sont généralement distingués : l'interaction *indirecte*, défendue par l'approche réactive et l'interaction *directe*, plutôt utilisée dans le cadre d'agents cognitifs. D'après la définition de l'interaction qui désigne l'action réciproque, le mode direct suppose qu'un agent échange des informations directement avec un autre agent. Le mode indirect, lui, suppose que l'échange s'effectue à travers un intermédiaire qui est l'environnement. Ferber (1995) emploie les termes d'agents *purement réactifs* pour désigner les agents qui communiquent uniquement de manière indirecte. À l'inverse, il appelle agent *purement communiquant* un agent cognitif qui communique uniquement de manière directe. Cependant, il est tout à fait possible qu'un agent puisse communiquer à l'aide des deux types d'interaction.

Dans cette sous-section, nous commençons par présenter les modes d'interaction indirecte aussi connus sous le nom de stigmergie. Nous faisons ensuite un état de l'art des différentes architectures qui permettent l'interaction directe entre agents.

3.3.1 *Interaction indirecte*

Nous nous intéressons ici à la manière dont plusieurs agents peuvent communiquer de manière indirecte, à travers un intermédiaire. Pour communiquer, les agents en interaction n'ont pas besoin d'avoir conscience d'autrui ou encore d'être à proximité d'un autre agent. Dans un premier temps, nous parlons des systèmes à tableau noir qui ont été les premiers modèles d'interaction indirecte avant de parler de l'utilisation de tuples comme intermédiaire d'interaction. Nous étudions ensuite la *stigmergie* comme modèle d'interaction.

Les tableaux noirs et les tuples

- À l'origine, les *systèmes à tableaux noirs* tels que ceux initiés par Erman et al. (1980) avec le projet HEARSAY II[8] sont nés du besoin de créer des systèmes faisant appel à différents domaines d'expertise pour résoudre des problèmes complexes. L'architecture permet d'apporter des éléments de solution progressivement par les différents composants « experts ». La métaphore du tableau noir est fondée sur la manière dont un groupe peut résoudre un problème à l'aide un tableau sur lequel chacun des pairs peut lire, juger son contenu et y apporter un nouvel élément de solution (Newell, 1969). Comme Weyns et al. (2005b) le font remarquer, ces systèmes restent éloignés des SMA puisque le tableau noir est une entité centralisée. Les sources de connaissances n'agissent pas de manière concurrente mais de manière séquentielle et contrôlée par l'ordonnanceur, contrairement aux agents qui sont des entités autonomes.

8. HEARSAY II est un système d'IA développé à la fin des années 70 dont le but était d'analyser une voix humaine afin de l'interpréter comme une requête de base de donnée exploitable.

- Un autre modèle d'interaction indirecte beaucoup plus adapté aux systèmes multi-agents est celui des *tuples* comme intermédiaire, qui a été introduit par Gelernter et Carriero (1992) avec Linda, un framework doté d'un modèle de coordination des calculs pouvant être généralisé et utilisé par n'importe quel langage de programmation. Un agent communique en insérant et en supprimant des tuples dans un espace partagé par l'ensemble des agents. Contrairement aux systèmes à tableau noir, les agents sont autonomes et évoluent de manière asynchrone.

La stigmergie

Le dernier modèle d'interaction indirecte que nous présentons est basé sur la *stigmergie*, un terme introduit par le zoologiste français Grassé (1959), du grec *stigma* (« marque ») et *ergon* (« action », « ouvrage »), pour désigner la façon dont les colonies de termites construisent leur nid. De manière généralisée et dans le cadre des SMA, le principe de la stigmergie peut se résumer à un ensemble d'entités individuelles interagissant de manière indirecte par l'intermédiaire d'un environnement partagé. Une interaction est initiée par une entité effectuant une première action qui consiste à déposer une marque au sein de son environnement. Cette marque va stimuler une deuxième action, et ainsi de suite, jusqu'à ce que de l'ensemble des actions individuelles émerge un ouvrage spontané. Ce type de mécanique a été observé avec les colonies de fourmis (Corbara et al., 1993 ; Drogoul, 1993). La façon dont les fourmis se déplacent dans leur environnement, guidées par les traces de phéromones qu'elles laissent derrière elles est devenu un exemple très classique de stigmergie.

3.3.2 *Interaction directe*

À l'inverse de la communication indirecte, l'interaction directe s'effectue d'agent à agent. Elle ne nécessite donc pas d'intermédiaire. Les communications s'effectuent par transfert de messages d'un agent émetteur vers un agent récepteur via un canal de communication, conformément à la théorie de la communication développée par Shannon et Weaver (1949). La communication directe permet de concevoir des SMA pour la résolution collective de problèmes, dans lesquels les agents sont capables de coopérer entre eux. Ainsi, si un agent n'est pas capable ou ne possède pas les compétences pour exécuter une tâche spécifique, il peut décider de se tourner vers d'autres agents en utilisant un langage de communication.

Les SMA ont été influencés par la théorie des actes du langage (Austin, 1975 ; Searle, 1969), laquelle distingue les énoncés constatifs, qui décrivent simplement l'état du monde, des énoncés performatifs, qui visent à accomplir une action permettant de modifier l'état du monde. Ces derniers forment les actes du langage.

Les actes de langage permettent de faciliter la modélisation des communications entre agents puisqu'ils permettent d'exprimer en dehors de son contenu, l'intention d'un message. Pour permettre aux agents de communiquer, un langage commun, appelé langage de communication agent ou *Agent Communication Language* (ACL), doit être connu des agents afin qu'ils puissent en analyser la syntaxe. Au delà de la forme du langage, la sémantique du contenu du message ainsi que la sémantique du contexte de communication (Poslad,

2007) doit être compris et analysable. La FIPA (*Foundation for Intelligent Physical Agents* [9]), un organisme de normalisation membre de l'IEEE Computer Society depuis 2005, propose un ensemble de spécifications pour le développement de systèmes multi-agents, notamment pour la communication entre agents. Un aperçu de l'approche de la FIPA est disponible dans Poslad (2007). Parmi les standards finalisés, le langage FIPA-ACL (Foundation for Intelligent Physical Agents, 2002a) met en œuvre les concepts de la théorie des actes du langage et permet de définir la sémantique du message.

Si un ACL permet aux agents de communiquer, il ne permet pas de structurer les interactions pour former une conversation. Pour pallier à cela, les protocoles d'interactions tels que FIPA Contract Net (Foundation for Intelligent Physical Agents, 2002b) ont fait leur apparition afin d'établir les règles à respecter pour que les agents puissent interagir sous la forme de conversations. Un certain nombre de plateformes sont compatibles avec les standards de la FIPA, c'est le cas de JADE (Bellifemine et al., 1999).

3.4 *Organisation*

La manière dont les agents sont organisés au sein d'un système multi-agents pour former une société virtuelle a intéressé une partie de la communauté, laquelle a donné naissance aux concepts d'*organisations*, de *communautés*, de *groupes*, d'*institutions* ou de *normes*. Bon nombre d'applications des systèmes multi-agents étant liées aux sciences humaines et sociales, une partie des recherches s'est concentrée sur la dimension sociale des SMA (Demazeau et Rocha Costa, 1996 ; Ferber et Gutknecht, 1998 ; Gasser, 2001). Ces travaux peuvent être classés en trois catégories :

1. L'organisation structurelle, qui s'attache à fournir les abstractions permettant d'organiser les agents en terme de groupes.
2. L'organisation du comportement des agents, à travers des concepts permettant au modélisateur de guider ou de réguler le comportement des agents. Pour cela, certains concepts ont été empruntés aux sciences humaines et sociales, comme les normes (Boella et al., 2006), l'engagement social (Pasquier et al., 2005) ou encore les sanctions (Kaminka et al., 2002).
3. La dernière catégorie de travaux définit les concepts permettant d'organiser la connaissance collective de groupes d'agents. Les concepts d'institution (Searle, 2005) ou encore de culture (Dinu et al., 2012) sont utilisés pour regrouper des notions propres à une collectivité d'agents.

Etant donné que nous n'utilisons pas les concepts relatifs aux deux dernières catégories dans nos travaux, seule la première catégorie est développée.

La structuration sociale permet d'établir les liens qui unissent les agents, ou au contraire, qui les opposent. Pour distinguer cette orientation des SMA, Ferber et al. (2004) parlent de SMA *centrés organisations* par opposition aux SMA *centrés agents*. Dans les SMA centrés agents, l'accent est mis sur le processus décisionnel de l'agent, qui se retrouve responsable de la gestion d'aspects sociaux. C'est lui, à travers son état, qui est responsable de déterminer

9. www.fipa.org

avec qui il peut entretenir des relations, ou encore par qui il peut être contacté. Avec les SMA centrés organisations, l'agent n'a plus à gérer ces aspects car une structure globale permet de maintenir une hiérarchie entre les agents, de les organiser en groupes et de rendre explicite leurs rôles. Cela permet également de réduire la complexité de l'agent : quand l'agent doit interagir, l'organisation le guide dans sa recherche d'un interlocuteur adéquat. Le modélisateur conçoit en amont la manière dont les agents sont organisés ainsi que les relations qu'ils entretiennent pour ensuite se concentrer sur l'architecture interne des agents. Ainsi, si la position physique d'un agent conditionne ses percepts et donc l'accès aux ressources qui l'entourent, la position sociale au sein d'une organisation peut restreindre l'interaction avec d'autres agents. Cette approche, qui permet de faciliter la coopération et la coordination des agents a donné lieu à différents métamodèles permettant de plonger les agents au sein de ces concepts. Nous présentons brièvement l'un d'eux, le modèle Agent-Groupe-Rôle (AGR) (Ferber et al., 2003).

Originellement appelé AALAADIN (Ferber et Gutknecht, 1997), le modèle deviendra avec Ferber et al. (2003) le modèle Agent-Groupe-Rôle (AGR). C'est un modèle organisationnel qui repose sur trois concepts primitifs : l'*agent*, le *groupe* et le *rôle*, structurellement connectés. Ces concepts sont définis de la manière suivante :

- Un *agent* est une entité autonome et communicante qui joue des *rôles* au sein de *groupes*. Un agent peut jouer plusieurs rôles et peut faire partie de plusieurs groupes. Aucune contrainte n'est posée sur l'architecture de l'agent afin de laisser plus de libertés au concepteur.
- Un *groupe* est simplement formé par un ensemble d'agents et un agent peut appartenir à plusieurs groupes. Le groupe permet de structurer une organisation d'agents, étant donné que deux agents peuvent communiquer si et seulement s'ils appartiennent au même groupe. Cette restriction permet notamment de construire des structures organisationnelles hiérarchiques.
- Le *rôle* est la représentation abstraite d'une fonction particulière qu'un agent a la possibilité d'exercer au sein d'un groupe. Un agent doit avoir un rôle au sein d'un groupe, mais il peut en jouer plus d'un. Les rôles sont locaux aux groupes, et peuvent être joués par plus d'un agent.

Le méta-modèle associé aux concepts que nous venons d'énumérer est représenté par le diagramme de classe UML en figure 1.11. Le lecteur intéressé peut se référer à Ferber et al. (2003) pour avoir la définition formelle de ces concepts sous forme d'axiomes. Comme Weyns et al. (2005b) le souligne, en restreignant la possibilité de communiquer avec des agents qui ne font pas partie d'un même groupe, le modèle permet de créer des structures hiérarchiques bien définies. Les groupes font office d'environnement pour les agents et peuvent fournir des services à leurs membres. La plate-forme de simulation multi-agents MadKit (Gutknecht et al., 2001) est basée sur ce modèle.

Depuis qu'il a été proposé, le modèle AGR a été étendu à plusieurs reprises. Parmi ces extensions, le modèle Agent-Groupe-Rôle-Environnement (AGRE), proposé par Ferber et al. (2005), permet de faire co-exister la notion d'organisation avec celle d'environnement physique.

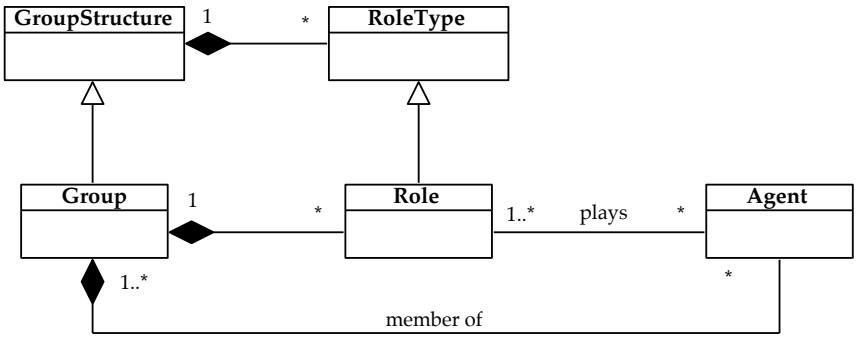


Figure 1.11 Méta-modèle UML du modèle organisationnel AGR.

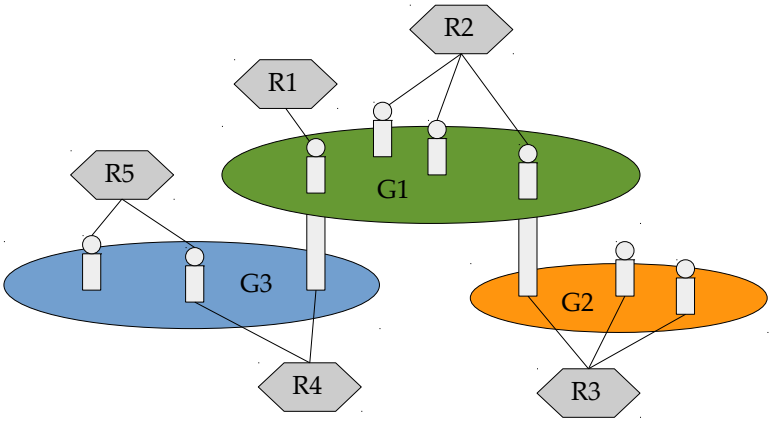


Figure 1.12 Exemple d'illustration concrète d'une organisation définie avec le modèle Agent-Groupe-Rôle à travers le diagramme du « plateau à fromages », repris de Ferber et al. (2003). Les rôles sont représentés par les hexagones et les groupes par les ellipses.

En étudiant les différents aspects de conception, nous avons pu entrevoir la richesse des concepts mis en application dans les SMA. Tous ces concepts ne sont généralement pas mis en œuvre au sein d'un même SMA, mais sont souvent dictés par le domaine d'application. Du point de vue de l'ingénierie logicielle, un nombre important de modèles piochant parmi les différentes notions que nous avons abordées ont été proposés pour la conception de SMA.

3.5 *Méthodologies de conception*

Nous nous plaçons maintenant dans un cadre d'ingénierie logicielle pour étudier les différents modèles et méthodologies qui ont été proposés pour guider la communauté dans la conception d'un SMA. Nous n'étudions pas de manière exhaustive l'ensemble des modèles existants, mais proposons plutôt une vue d'ensemble afin de voir les différents concepts mis en œuvre dans chacun d'eux. Ceci nous permet également d'identifier quels modèles sont adaptés à la simulation orientée agent, et lesquels sont les plus adaptés pour représenter des systèmes environnementaux, qui nous intéressent plus particulièrement

dans le cadre de cette thèse.

Avant de parler de manière plus détaillée des apports de ces différents travaux, quelques définitions s'imposent, afin de clarifier les concepts de *métamodèle*, de *méthode* et de *méthodologie*, issus de l'ingénierie logicielle :

Métamodèle Un métamodèle est un modèle qui définit le langage d'expression d'un modèle, c'est-à-dire le langage de modélisation (Combemale, 2008). Selon Cadavid et al. (2012), il est définit par la composition de :

- Concepts. Les concepts fondamentaux et les attributs qui définissent le domaine.
- Relations. Les relations qui spécifient la manière dont les concepts peuvent être liés ensemble dans un modèle.
- Règles. Des propriétés supplémentaires qui restreignent la façon dont les concepts peuvent être assemblés pour former un modèle valide.

Méthodologie (Ghezzi et al., 2002) Une méthodologie est une collection de méthodes couvrant et reliant les différentes étapes d'un processus. L'objectif d'une méthodologie est de prescrire une approche cohérente à adopter pour résoudre un problème dans le contexte d'un processus de développement.

Méthode (Cernuzzi et al., 2005) Une méthode prescrit une manière d'exécuter un type d'activité dans un processus donné afin de produire correctement un résultat spécifique à partir d'une entrée spécifique. Toutes les phases d'un processus, pour être applicables avec succès, doivent être complétées par toutes directives méthodologiques (identification des techniques et outils à employer, et définition de la façon dont les résultats doivent être produits) susceptibles d'aider les parties prenantes du projet à accomplir leur travail selon les meilleures pratiques définies.

La plupart des méthodologies orientées agent sont issues de l'ingénierie logicielle et s'inspirent des méthodes orientées objet tout en intégrant les spécificités liées au paradigme agent, comme la notion d'autonomie, de protocoles d'interaction entre agents ou d'organisation sociale. Si certaines méthodologies sont plus ou moins complètes, la plupart couvrent l'intégralité du processus de développement, c'est-à-dire l'analyse des besoins, la conception de l'architecture, le développement et le déploiement (Gleizes et al., 2009).

Comme le font remarquer Kubera et al. (2011), la plupart des méthodologies orientées agent sont destinées à la conception d'applications livrables. De fait, elles ne sont pas forcément adaptées au processus de développement de modèles de simulation pour plusieurs raisons. En effet, les cycles et processus de développement logiciels classiques (en cascade, en V, en spirale, itératif) diffèrent du processus de simulation (Campos et al., 2004). Ce dernier peut être considéré comme un processus itératif dont l'objectif est de faire évoluer un modèle initial jusqu'à obtenir un modèle en adéquation avec les objectifs de simulation, représentatif d'un système réel, tandis que les cycles de conception des processus d'ingénierie logicielle visent à atteindre de nouvelles fonctionnalités à travers une conception modulaire et incrémentale.

Au delà du processus de développement, il est possible de déterminer si une méthodologie est compatible avec la M&S en fonction des concepts du paradigme agent qu'elle

manipule. En effet, les méthodologies « sont principalement destinées à la spécification de composants logiciels concurrents (agents) » (Campos et al., 2004), et leur conception se concentre sur la façon dont ils sont organisés et sur la façon dont ils communiquent (Kubera et al., 2011). Or, comme le rappellent Kubera et al. (2011), les interactions dans une simulation orientée agent (particulièrement dans des domaines comme l'écologie ou l'éthologie) ne reposent pas uniquement sur une communication par échanges de messages, mais impliquent également des interactions physiques au sein d'un environnement.

Quelques travaux ont appliqué une méthodologie pour la simulation d'agents : INGENIAS (Pavón et al., 2008) et IODA (Kubera et al., 2011). Campos et al. (2004), eux, proposent MASIM, une méthodologie dédiée au développement de simulations orientées agent. À notre connaissance, le reste des méthodologies proposées sont destinées à la conception d'applications plutôt qu'au développement de modèles de simulation. Gleizes et al. (2009) dressent un état de l'art offrant une vue d'ensemble des différentes méthodologies qui ont été proposées pour le développement de SMA.

Si les méthodologies ne sont pas forcément adaptées au processus de modélisation et de simulation, elles établissent en revanche des relations entre les concepts issus du paradigme agent. À ce titre, elles définissent un métamodèle qu'il est possible d'étudier pour voir quels concepts ont été appliqués. Le métamodèle est parfois clairement identifié au sein d'une méthodologie, comme CRIO (Rodriguez et al., 2007) pour ASPECS (Cossentino et al., 2009) ou AMAS-ML pour ADELFE (Picard et Gleizes, 2004). Dans le cas contraire, il peut tout de même être extrait. D'autres chercheurs ont également proposé des métamodèles agent sans y associer une méthodologie de conception particulière. C'est le cas d'AGR (Ferber et Gutknecht, 1998), de RIO (Mathieu et al., 2003), d'AGRE (Ferber et al., 2005), de MASQ (Stratulat et al., 2009) ou encore des travaux d'Odell et al. (2003b). Face à la diversité des métamodèles existants, des travaux d'unification ont également été proposés. Par exemple, Bernon et al. (2009) proposent un métamodèle issu des méthodologies GAIA (Wooldridge et al., 2000), ADELFE et PASSI (Cossentino et Potts, 2001). Elammari et Elsaeti (2011), eux, proposent un métamodèle issu des méthodologies ROADMAP, HILM (Elammari et Lalonde, 1999), SONIA (Alonso et al., 2005) et STYX (Bush et al., 2001).

Dans le tableau 1.1, nous dressons un comparatif de plusieurs métamodèles selon les concepts qu'ils mettent en œuvre. L'aspect environnemental est étudié afin de savoir si l'approche offre une structuration physique aux agents (topologie, ressources, ...), une structuration sociale (concepts organisationnels), si plusieurs environnements peuvent coexister ou encore si l'agent dispose d'une représentation dans l'environnement (corps, rôle, ...). L'aspect interactionnel est également étudié, afin de déterminer si les agents peuvent communiquer de façon directe ou indirecte.

Comme on peut l'observer sur le tableau 1.1, les aspects sociaux associés à la communication directe sont les plus développés par les différents métamodèles. Ce constat n'est pas surprenant, puisque comme nous l'avons déjà mentionné, ces métamodèles sont pour la plupart associés à des méthodologies pour la conception de SMA applicatifs. Pour la simulation de systèmes environnementaux, qui nous intéresse plus particulièrement,

Métamodèle	Environnement				Interaction	
	physique	social	multiple	repr. agent	directe	indirecte
AMAS-ML	✓	✗	✓	✓	✓	✓
GAIA	✓	✓	✓	✗	✓	✓
PASSI	✗	✓	✗	✓	✓	✗
ROADMAP	✓	✓	✓	✓	✓	✓
HILM	✗	✓	✗	✗	✓	✗
STYX	✗	✓	✗	✓	✓	✗
SONIA	✓	✓	✗	✗	✓	✓
CRIO	✗	✓	✗	✓	✓	✗
AGRE	✓	✓	✓	✓	✗	✗
IODA	✓	✓	✗	✗	✓	✓
MASQ	✓	✓	✓	✓	✓	✓

Tableau 1.1 Comparaison (non-exhaustive) des différents métamodèles SMA selon les aspects environnementaux et interactionnels.

un environnement capable de structuration physique, associé à la possibilité d'interaction indirecte, est indispensable. L'aspect multi-environnemental n'est pas une caractéristique très répandue, mais offre des possibilités également intéressantes en vue de la simulation de systèmes environnementaux (*e.g.* représentation d'un environnement aquatique et de surface pour la modélisation d'un système de pêcherie). Parmi les différents métamodèles étudiés, bien que minimaliste, seul le métamodèle MASQ (Stratulat et al., 2009), semble développer tous les critères qui nous semblent important pour la conception d'un modèle SMA. À ce titre, il représente une source d'inspiration intéressante pour la suite de nos travaux. Soulignons néanmoins que ce constat ne constitue pas selon nous un critère suffisant pour arrêter définitivement notre choix sur l'utilisation d'un métamodèle en particulier.

4 *Simulation orientée agent et reproductibilité*

Jusqu'ici, nous avons donné une vision d'ensemble des différents points de vue à propos des concepts du paradigme agent, ainsi que des travaux visant à guider la conception d'un SMA. Nous proposons dans cette section de présenter les différentes problématiques qui ont pour dénominateur commun l'amélioration de la reproductibilité des expériences numériques, influence principale de nos travaux de recherche. Les difficultés liées à la reproductibilité des modèles et des résultats publiés, a été soulevée à plusieurs reprises au sein de la communauté SMA (Duboz et al., 2012 ; Michel, 2004 ; Rouchier, 2003 ; Wilensky et Rand, 2007). Bien que cette problématique concerne la simulation de manière générale (Stodden et al., 2013), la complexité des SMA et la jeunesse du paradigme agent semblent l'exacerber.

4.1 *Qu'est-ce que la reproductibilité ?*

L'idée première derrière la notion de reproductibilité est qu'un scientifique puisse reproduire une expérience qui a été partagée au sein de la communauté scientifique dans

le but d'obtenir les mêmes résultats. Dans le cadre d'expériences numériques, Dao et al. (2016) proposent un état de l'art des définitions issues de la littérature. Deux dimensions de la reproductibilité y sont dégagées : la reproduction de l'expérience numérique et la reproduction exacte du résultat calculé (*i.e.* reproductibilité numérique). La nuance entre *reproductibilité* et *répétabilité* est également mise en avant : la répétabilité « consiste à retrouver les mêmes résultats lorsque deux expériences sont menées avec les mêmes paramètres d'entrées, avec des matériels, des méthodes et des contextes identiques ». La reproductibilité se veut donc plus générale que la répétabilité. Parmi les apports de la reproductibilité, Dao et al. (2016) nous rappelle qu'elle « constitue une méthode et un standard pour juger de la pertinence d'une expérience numérique publiée et donc des conclusions qui en découlent. »

Entre la notion de répétabilité et de reproductibilité, différents degrés sont suggérés dans la littérature. Dalle (2012) propose par exemple quatre niveaux de reproductibilité, selon la disponibilité du code source, les aspects non-déterministes du calcul et la similitude du plan d'expérience et des sorties du modèle. Dans leur rapport, Stodden et al. (2013) définissent également plusieurs niveaux de reproductibilité :

- l'examen par les pairs, qui est la méthode traditionnelle de publication (les travaux et résultats décrits sont jugés crédibles par la communauté scientifique) ;
- la recherche réplicable, où les outils permettant de reproduire les même résultats sont fournis ;
- la recherche vérifiable, où les même conclusions peuvent être atteintes indépendamment du code source fournit par l'auteur ;
- la recherche validable, où suffisamment de ressources (sources et données) sont archivées afin de permettre de défendre les résultats fournis ;
- la recherche ouverte, où tous les éléments utilisés pour arriver aux résultats présentés sont fournis en accès libre et documentés.

Les conclusions établies par Stodden et al. (2013), Dalle (2012) et Dao et al. (2016) se rejoignent. Nous les synthétisons de la manière suivante : pour permettre un degré de reproductibilité se rapprochant de la « recherche ouverte », un changement dans la culture de publication est nécessaire. Ce changement doit être impulsé non seulement par les auteurs, mais également par les éditeurs. Il est recommandé aux premiers de fournir les éléments permettant de reproduire l'expérience de simulation (données, code source, ...), et aux seconds d'encourager cet effort durant le processus de publication en offrant les supports numériques d'archivage et de diffusion de ce contenu.

Dans le cadre de cette thèse, nous nous intéressons particulièrement à la reproductibilité de l'expérience numérique, pour permettre à une simulation orientée agent d'être reproduite une fois publiée avec des matériels et des outils différents. Nous négligeons cependant les aspects de reproductibilité numérique, qui prennent une dimension « bas niveau » avec des problématiques liées par exemple à la comparaison de nombres à virgule flottante, au soupassement arithmétique ou aux spécificités des architectures matérielles.

La notion de reproductibilité est en lien avec les problématiques soulevées par le domaine de la vérification et de la validation (V&V). La V&V s'attache à décomposer l'ensemble des

étapes de conception d'un modèle de simulation afin d'identifier les sources d'introduction d'erreurs pendant le processus de modélisation. La figure 1.13 illustre l'ensemble des étapes de la conception d'un modèle. Entre la spécification du modèle conceptuel et son implémentation, deux vérifications s'imposent (Sargent, 2005) :

- la vérification des spécifications, qui permet de vérifier si les spécifications sont satisfaisantes et qu'elles permettent une implémentation non ambigüe sur le système informatique cible ;
- et la vérification de l'implémentation, qui permet de s'assurer que l'implémentation du modèle est conforme aux spécifications et que le simulateur n'introduit pas d'autres biais.

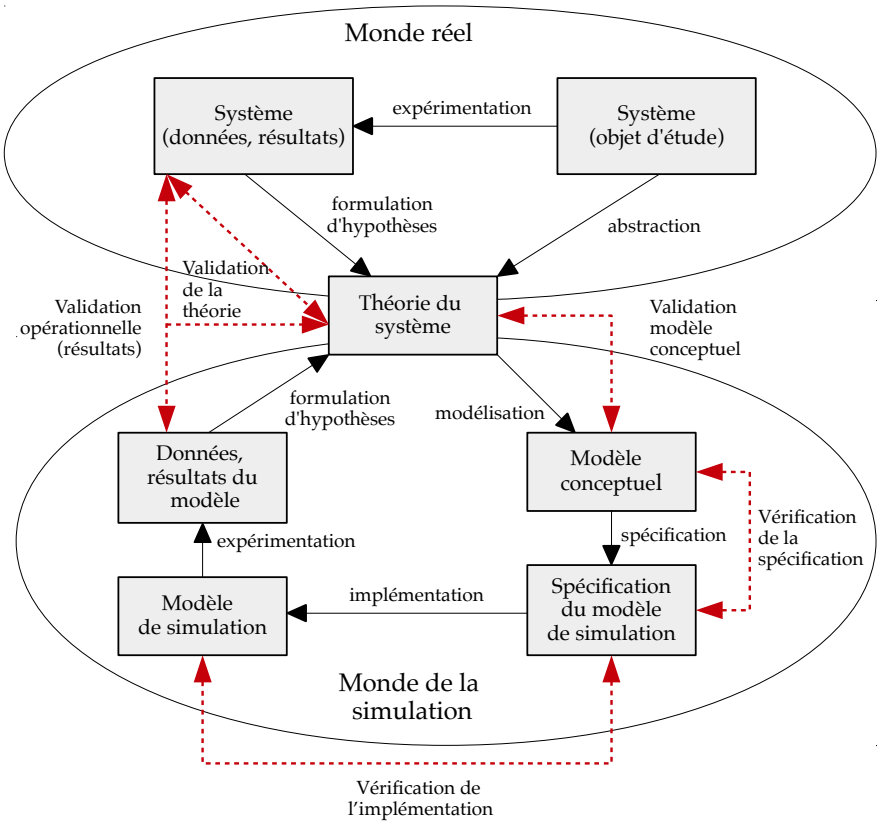


Figure 1.13 Relation entre le monde réel et le monde de la simulation, repris de Sargent (2005).

Michel (2004) s'est particulièrement intéressé à la relation de simulation. Selon lui, la relation de simulation « soulève la question de la neutralité du simulateur vis-à-vis du modèle. Il s'agit de garantir autant que possible que le simulateur exécute le modèle tel qu'il a été formulé et qu'il n'introduit pas de biais dans le processus de simulation dus à sa propre implémentation ». Lorsque cette relation n'est pas vérifiée, il parle de phénomène de *divergence implémentatoire*. Sa conclusion est la suivante : le manque de spécifications des modèles multi-agents associé à l'absence d'un formalisme adéquat favorise l'introduction de biais dans le processus de modélisation et de simulation.

Nous pensons qu'un changement de la culture de la publication s'accompagne inévita-

blement d'une longue période de transition pour atteindre le degré de recherche ouverte décrit par Stodden et al. (2013). Selon nous, cette transition doit s'accompagner des moyens permettant aux chercheurs de spécifier et publier leurs modèles avec les outils existants, indépendamment d'une plateforme, afin d'atteindre au moins le niveau de « recherche vérifiable » à l'aide des méthodes traditionnelles de publication.

Puisque nos travaux concernent les modèles de simulation basés sur le paradigme agent, nous proposons dans la suite de cette section d'énumérer les freins à la reproductibilité d'un modèle SMA. Nous donnons ensuite quelques éléments de solution sur lesquels nous basons nos contributions.

4.2 *Entraves à la reproductibilité*

Dans cette section, nous donnons les différentes problématiques qui peuvent empêcher la reproductibilité d'une expérience numérique. Ces problématiques peuvent être organisées selon deux catégories : les facteurs humains et les facteurs techniques.

Parmi les facteurs humains, Dalle (2012) met en cause l'ignorance sincère à propos de certains paramètres (*e.g.* le générateur de nombre aléatoire utilisé), les erreurs de manipulation, l'impossibilité de partager le code source pour des raisons de licence, ou encore l'omission de détails importants vis-à-vis du modèle dans les publications. Michel (2004) le souligne également, une spécification insuffisante du modèle dans une publication ne permet ni la réplication d'un modèle ni la reproductibilité d'une expérience numérique. Ce constat a également été souligné dans des travaux consistant à reproduire une expérience numérique, comme ceux de Edmonds et Hales (2003).

À ce sujet, Duboz et al. (2012) nous alertent sur la nécessité d'utiliser une spécification rigoureuse afin de permettre une implémentation non ambigüe du modèle. Cette question est d'autant plus difficile que les systèmes multi-agents sont issus et utilisés dans des domaines de recherches différents aux enjeux variés, ce qui a entraîné des définitions et visions différentes à propos de leur nature, même si aujourd'hui la communauté semble aboutir à un consensus.

Ces problématiques sont également liées aux facteurs techniques. En effet, la définition d'un modèle SMA et sa simulation sont souvent liées aux spécificités de la plateforme utilisée. Les plateformes de modélisation et de simulation dédiées aux SMA telles que NetLogo (Wilensky, 1999) ou GAMA (Grignard et al., 2013) proposent des langages informatiques spécifiques qui offrent des primitives permettant de faciliter la modélisation des systèmes multi-agents. Un tel langage constitue une abstraction sur les concepts implémentés par la plateforme. Il dépend donc des choix conceptuels qui ont été opérés à propos de la nature d'un SMA et des entités qui le compose.

L'étude menée par Bajracharya et Duboz (2013) qui consiste à implémenter un modèle épidémiologique simple sur trois plateformes SMA différentes en témoigne. Après la simulation des différents modèles, chaque plateforme produit des résultats différents. La

diversité des plateformes de simulation SMA ne permettent pas d'obtenir des résultats identiques d'une plateforme à l'autre. Parmi les facteurs qui peuvent influencer les résultats, Lawson et Park (2000) identifient l'ordonnancement des agents. En effet, le fait qu'une multitude d'agents effectuent leurs actions potentiellement de manière simultanée dans un environnement commun peut entraîner des trajectoires d'états différentes et donc influencer les résultats en fonction de leur ordre d'activation. Michel (2004) identifie également la manière dont le simulateur traite les actions simultanées des agents comme une source qui peut mener à des résultats différents.

L'activité de l'environnement, formée par l'ensemble des perceptions/actions des agents et des phénomènes endogènes est un processus complexe qui n'est pas toujours implémenté de la même manière dans les plateformes de simulation multi-agents. La difficulté principale est que les agents puissent réaliser leurs actions de manière simultanée. Nous voyons ici différentes approches qui permettent de prendre en compte la simultanéité des actions.

Pour traiter les actions que les agents effectuent de manière simultanée, la méthode majoritairement utilisée dans les plateformes de simulation SMA est basée sur la transformation directe de l'état de l'environnement. Plusieurs approches ont été mises en œuvre dans les différentes plateformes de simulation orientées agent :

Activation séquentielle Pour l'ensemble des agents effectuant une action à la même date, les agents sont activés de manière séquentielle. Cette solution est facile à mettre en œuvre mais naïve, car si deux actions influent sur la même variable d'état de l'environnement, l'ordre d'activation aura un impact sur la valeur finale de la variable, mais également sur l'état de l'agent. Ainsi, avec le même ordre d'activation, certains agents sont constamment privilégiés.

Activation aléatoire Pour l'ensemble des agents simultanés, les agents sont activés de manière aléatoire. Une activation aléatoire permet d'éviter de privilégier systématiquement le même agent. Bien évidemment, comme toute simulation stochastique, il convient de réaliser un certain nombre de répétitions de la même simulation afin d'obtenir une approximation du phénomène.

Activation ordonnée Pour l'ensemble des agents simultanés, les agents sont activés selon un ordre précis. Cette solution permet de privilégier volontairement un agent et peut être utile lorsqu'une règle de priorité doit être mise en place. Cependant, un problème subsiste : celui de la perception du monde par les agents. Pendant la prise en compte des actions de tous les agents simultanés pour calculer le nouvel état, tous les agents doivent avoir accès au même état de l'environnement, ce qui n'est pas le cas avec une activation séquentielle, aléatoire ou ordonnée. En effet, à partir du moment où un agent effectue une action qui entraîne la modification directe de l'état de l'environnement, les agents suivants dans la liste d'activation pour le même temps de simulation n'accèderont pas au même état que les agents précédents.

Etat temporaire Une solution au problème de perception posé par l'activation séquentielle, aléatoire ou ordonnée réside dans l'utilisation de variables temporaires permettant de stocker l'état de l'environnement. L'algorithme générique d'évolution de l'environnement proposé par Russel et Norvig (1995) par exemple, réalise dans un premier temps le

calcul des perceptions pour chaque agent, puis active chacun d'eux dans un second temps. Cette solution est à combiner avec l'activation aléatoire, pour éviter de privilégier certains agents, ou à une méthode de résolution de conflit.

Résolution de conflit Dans le cas où les actions entraînent la modification de la même variable d'état de l'environnement, un mécanisme de gestion de conflit associé à la mise en place d'un état temporaire peut être mis en place afin d'éviter d'écraser les nouvelles valeurs. Dans ce cas de figure, l'ensemble des nouvelles valeurs produites par les actions des agents sont mises en tampon, puis une seconde phase vient valider une valeur pour chaque variable, en fonction d'une règle précise. Cette technique est celle de l'approche à trois phases, décrite par Balci (1988).

Le problème avec l'ensemble de ces solutions est que le simulateur applique un comportement prédéterminé pour gérer les agents simultanés. Un certain nombre de plateformes de simulation multi-agents utilisent l'une de ces solutions. C'est le cas par exemple de NetLogo (Wilensky, 1999) qui réalise des simulations à temps discret et dont le modèle d'action est basé sur l'activation aléatoire (avec possibilité de tri). La plateforme GAMA (Grignard et al., 2013), elle, utilise également le temps discret et propose l'activation séquentielle par défaut mais laisse la possibilité de mélanger ou de trier la liste d'exécution. Malheureusement, les choix qu'opèrent ces plateformes sont des facteurs pouvant rendre difficile la reproduction d'un modèle.

Dans la prochaine section, nous donnons les éléments de solution qui permettent de faciliter la spécification d'un modèle, tout en rendant explicite les détails liées aux problématiques d'ordonnancement et de perception de l'environnement.

4.3 *Éléments de réponse*

Afin d'apporter une solution aux problématiques que nous avons listées dans la section précédente, nous donnons ici les éléments qui, selon nous, peuvent permettre de faciliter la reproduction d'une expérience. Les principales problématiques que nous avons soulignées sont :

1. en terme de facteur humain : les spécifications des modèles, qui ne sont pas suffisamment détaillées dans les publications ;
2. en terme de facteur technique : les problématiques relatives à l'ordonnancement des agents, qui diffère selon les plateformes de simulation utilisées.

Pour répondre à ces problématiques, nous développons deux axes, pour lesquels nous donnons différents éléments de réponse, et sur lesquels nous basons nos travaux.

4.3.1 *Méthodes de spécification des modèles*

Afin de permettre d'atteindre un niveau de recherche vérifiable (Stodden et al., 2013) en utilisant les supports traditionnels de publication, nous pensons que l'utilisation de méthodes permettant de guider le modélisateur dans la spécification de ses modèles peut permettre d'atteindre un niveau de détail suffisant pour faciliter sa reproductibilité. Dalle

(2012) suggère que pour atteindre un niveau de reproductibilité pour lequel le partage du code source n'est pas requis, toute spécification suffisamment détaillée de l'expérience est acceptable. Pour atteindre un degré plus élevé, l'auteur suggère également le partage du modèle dans une forme indépendante d'une plateforme. Celle-ci peut par exemple être atteinte à travers l'utilisation d'un langage mathématique, tel qu'un formalisme adapté à la M&S.

Duboz et al. (2012) proposent une méthode de spécification des modèles de simulation en cinq parties distinctes, lesquelles peuvent être appliquées de manière itérative. Les auteurs laissent la possibilité d'omettre une partie si celle-ci n'est pas considérée nécessaire suivant le contexte. Les différentes parties sont décrites dans l'ordre suivant :

1. la *communication des objectifs de modélisation et du modèle* est une description textuelle permettant de comprendre dans quel objectif le modèle est conçu, son contexte d'utilisation ainsi que les hypothèses qui ont été formulées à propos du système source ;
2. la *spécification formelle du cadre expérimental du modèle*, où l'approche préconisée est d'utiliser le formalisme proposé par Traoré et Muzy (2006) ;
3. la *spécification formelle du modèle* ;
4. la *spécification formelle du simulateur et du coordinateur associé* permet de définir une sémantique d'exécution du modèle ;
5. enfin, la *communication des expériences de simulation* permet de décrire les différents plan d'expériences (paramètres initiaux, durée de simulation, ...) utilisés dans la simulation.

Au sein de la communauté SMA, la problématique a également été soulignée (Grimm, 1999). Afin de faciliter le partage des modèles, le protocole ODD, pour *Overview, Design concepts, and Details* (Grimm et al., 2006, 2010) a été proposé afin de guider les auteurs dans la spécification de leurs modèles orientés agent.

4.3.2 *Intégrer la simultanéité des actions au modèle*

Parmi les obstacles identifiés comme des facteurs de non-reproductibilité, nous avons souligné les aspects liés à l'ordonnancement des agents, ainsi que les différences de perception de l'état de l'environnement par les agents. Nous avons vu que selon les plateformes SMA utilisées, l'activation des agents peut être gérée de manière très différente et ne laisse pas forcément la possibilité à l'utilisateur de maîtriser l'ordonnancement. Pour répondre à cette problématique, nous présentons une théorie de l'action spécifiquement étudiée pour rendre explicite la façon dont les actions simultanées des agents doit être prise en compte dans un modèle. Appelée Influence/Réaction, cette vision différente de l'action permet de tirer parti de la simultanéité ainsi que du parallélisme intrinsèque du paradigme agent. Nous pensons, au même titre que Michel (2004), que cette façon de prendre en compte l'action des agents permet d'apporter une solution au phénomène de divergence implémentatoire.

Contrairement à l'approche classique, les actions des agents ne modifient pas directement l'état de l'environnement ; elles *influent* sur l'environnement et ce dernier y *réagit* par la suite. Les agents font part au système de leurs intentions, ils tentent de modifier l'environnement

qui peut y réagir favorablement ou non. Ce modèle d'action, proposé par Ferber et Müller (1996) repose sur trois concepts :

1. la distinction entre les *influences* et les *réactions* afin de gérer les actions simultanées. Les influences représentent les tentatives des agents de modifier le cours d'évènements qui auraient eu lieu avec une approche classique. Les réactions se traduisent par des changements d'états et sont produites en combinant les influences de tous les agents, à partir de l'état courant de l'environnement et de ses lois.
2. La décomposition de la dynamique globale du SMA en deux parties : la dynamique de l'environnement (l'état de l'environnement) et la dynamique des agents situés dans l'environnement (les influences qu'ils produisent).
3. La description des différentes dynamiques par des machines à états abstraits.

Avec cette approche, le nouvel état de l'environnement est calculé en réaction à l'ensemble des influences qui sont produites de manière simultanée. Ce modèle d'action fait la distinction entre l'action produite par l'agent et son effet sur l'environnement. Cet effet ne peut être calculé qu'à partir de l'ensemble des tentatives d'action effectuées sur l'environnement au même instant, en fonction des règles environnementales. Imaginons deux agents, qui agissent de concert pour déplacer un objet lourd. Grâce à la phase de réaction de l'environnement, le modélisateur peut prendre en compte la force produite par les deux agents et l'additionner pour déterminer à quelle vitesse l'objet en question va bouger.

À partir de ces travaux, Michel (2007a) définit le modèle IRM4S (*Influence Reaction Model for Simulation*), que nous décrivons partiellement. Le comportement d'un agent est représenté par une fonction permettant de décrire le processus de délibération :

$$Behaviour_a : \Sigma \times \Gamma \rightarrow \Gamma$$

où Σ est l'ensemble des états de l'environnement et Γ l'ensemble des influences produites par les agents. Contrairement à Ferber et Müller (1996), qui permettent aux agents de percevoir uniquement les influences sur l'environnement, le modèle IRM4S offre la possibilité aux agents de percevoir à la fois l'état et les influences. La possibilité de décrire une dynamique environnementale est également proposée via la fonction suivante :

$$Natural_{env} : \Sigma \times \Gamma \rightarrow \Gamma$$

Les fonctions *Behaviour* et *Natural* forment l'activité des agents et de l'environnement. Elles permettent de produire des influences en fonction de l'état courant du système.

L'évolution de l'état du SMA est décomposée en deux fonctions distinctes, appliquées de manière séquentielle :

$$Influence : \Sigma \times \Gamma \rightarrow \Gamma'$$
$$Reaction : \Sigma \times \Gamma' \rightarrow \Sigma \times \Gamma$$

La fonction $Influence$ représente la première phase de l'évolution du système. Elle permet de calculer le nouvel ensemble d'influences produites par les agents et l'environnement à partir de l'état courant. Ces influences sont ensuite utilisées en argument de la fonction

Reaction. Cette dernière représente la deuxième phase de l'évolution du SMA et décrit la transition d'état de l'environnement.

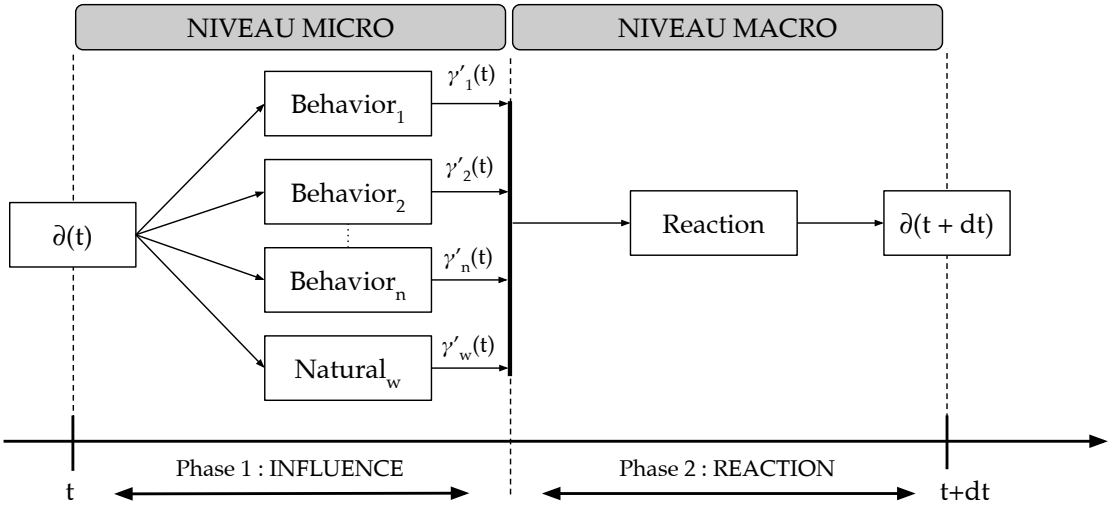


Figure 1.14 Schéma de l'évolution de l'état dynamique d'un système multi-agents modélisé avec le modèle d'action IRM4S, repris de Michel (2007a).

La figure 1.14, reprise de Michel (2007a), permet d'avoir une vision temporelle des phases influence et réaction. Ainsi, contrairement au modèle d'origine, c'est l'application séquentielle de ces deux phases qui conduit à un état cohérent. D'un point de vue temporel, l'application des deux phases est instantanée. Durant le calcul de réaction, l'ensemble des tentatives d'action simultanées peuvent être prises en compte pour calculer le nouvel état du système. Ceci permet de balayer les problématiques liées à l'ordonnancement des agents et d'intégrer dans le modèle du SMA la gestion simultanée des actions.

Cette section a été l'occasion de mettre en avant les problématiques rencontrées par le domaine de la simulation de systèmes multi-agents. Nous pensons qu'un langage formel permettant de décrire les modèles associé à une méthode de spécification peut contribuer à faciliter la réplicabilité des modèles SMA et la reproductibilité des expériences numériques.

5 *Positionnement et conclusion*

Ce chapitre nous a permis de poser les bases de la modélisation et de la simulation en étudiant les concepts fondamentaux qu'abritent les notions de système, de modèle, de simulation ou encore de paradigme de modélisation. Nous nous sommes ensuite attardés sur le paradigme agent, qui est au centre de notre problématique, en introduisant les concepts du paradigme et les enjeux pour leur conception. Nous avons souligné l'existence de différents points de vue quant à la définition de ces concepts et aux responsabilités accordées à chaque entité d'un SMA. Nous nous sommes alors employés à utiliser des définitions consensuelles et à présenter les différentes visions qui existent au sein de la communauté SMA.

Etant donné que nous manipulons les concepts associés aux SMA tout au long de ce

manuscrit, nous donnons notre positionnement vis-à-vis de ces concepts et des responsabilités que nous accordons à chaque entité d'un SMA de façon explicite. Pour chaque entité d'un SMA, nous dédions un paragraphe dans lequel nous synthétisons les aspects qui ont le plus d'importance selon notre point de vue, et que nous utilisons comme support pour nos contributions.

Agent

L'agent doit vérifier les propriétés qui le définissent selon Wooldridge et Jennings (1995) : *proactivité*, *réactivité*, *sociabilité* et *autonomie*. Pour établir la propriété d'autonomie, l'agent doit être le seul à disposer du contrôle de son propre état (*contrainte d'intégrité interne*). Pour cela, nous utilisons la séparation explicite du corps et de l'esprit de l'agent. C'est à travers l'esprit (système cognitif) que l'agent perçoit l'environnement, qu'il délibère (de manière proactive ou réactive) et qu'il agit en produisant des influences, conformément à l'approche Influence/Réaction. Si l'esprit est responsable des décisions, le corps peut représenter une limite à leurs réalisations, d'abord par ses propriétés, qui caractérisent l'agent, mais également parce qu'il est soumis aux contraintes imposées par l'environnement. Il en va de même pour la perception : les propriétés du corps déterminent ce que l'agent est capable de percevoir. Le corps a donc un rôle de représentation au sein de l'environnement : c'est l'intermédiaire entre l'esprit de l'agent et l'environnement. Nous considérons cette entité comme une abstraction de première classe, au même titre que l'agent ou l'environnement.

Environnement

Selon nous, l'environnement est une entité qui joue un rôle central au sein d'un SMA. Au même titre que Weyns et al. (2006), nous lui prêtons les responsabilités suivantes :

1. L'environnement structure le système multi-agents en formant un espace partagé pour les agents et d'éventuelles ressources sous trois formes :
   - structuration physique, en fournissant une structure spatiale aux agents, en établissant par exemple des relations topologiques et/ou en permettant de distribuer l'environnement ;
   - structuration sociale, qui permet d'organiser les agents pour former une société virtuelle en terme de groupes et de rôles ;
   - structuration des communications, en fournissant l'infrastructure permettant aux agents de communiquer de manière directe ou indirecte.
2. L'environnement peut maintenir une dynamique propre afin de représenter des processus environnementaux (*e.g.* évaporation et diffusion de phéromones).
3. L'état de l'environnement est accessible selon la position physique ou sociale de l'agent qui l'observe, afin de respecter la contrainte de localité.
4. L'environnement est responsable de son propre état. Afin de respecter la contrainte d'intégrité de l'environnement, les agents n'effectuent pas de modification directe de l'état de l'environnement. L'utilisation du principe Influence/Réaction, permet de s'en assurer.
5. Des règles peuvent contraindre les actions des agents afin de préserver l'environnement dans un état cohérent pour les objectifs du modèle SMA. Là aussi, le principe Influence/-

Réaction apporte une solution en permettant à l'environnement de réagir favorablement ou non aux influences en fonction du contexte.

Système multi-agents

Les propriétés que nous attendons du système multi-agents sont les suivantes :

- possibilité de faire coexister plusieurs environnements au sein d'un même SMA (Soulié, 2001), afin qu'un agent puisse être plongé dans de multiples environnements, physiques et/ou sociaux ;
- la dynamique structurelle doit être possible, afin de matérialiser les aspects dynamiques d'un SMA, tel que la création ou la destruction d'entités ou l'évolution des interactions ;
- permettre la gestion d'actions simultanées afin d'éviter les problématiques liées à l'ordonnancement des agents (Lawson et Park, 2000) ;
- permettre de représenter des SMA multi-niveaux, en donnant la possibilité de considérer le SMA comme un agent dans un SMA de niveau supérieur.

La dernière partie de ce chapitre nous a permis de mettre en avant l'intérêt de l'utilisation des méthodes formelles pour la définition d'un modèle de simulation basé sur le paradigme agent. Afin d'orienter notre choix d'un formalisme adapté à la modélisation de SMA, nous effectuons dans le prochain chapitre un état de l'art des différents formalismes de modélisation et de simulation, ainsi que des différents travaux formels concernant les systèmes multi-agents. C'est sur la base de ces définitions que notre approche sera construire. Dans le chapitre suivant, nous présentons de manière plus concrète la notion de formalisme, identifié comme essentiel pour favoriser une description univoque des SMA.

Chapitre II

FORMALISMES POUR LA SPÉCIFICATION DE MODÈLES

Sommaire

1 *Introduction*

La modélisation et la simulation (M&S) sont utilisées par de nombreuses disciplines. Elle permet de nous aider à comprendre le monde qui nous entoure, d'anticiper des phénomènes naturels, d'optimiser la conception d'infrastructures et de produits, ou encore d'étudier l'impact des activités humaines sur l'environnement. L'étude et la compréhension du monde réel sont guidées par un ensemble d'étapes (observation, hypothèse, abstraction, prédiction, expérience, *etc.*) qui constituent la démarche scientifique. Au cœur de cette démarche, la modélisation est l'étape qui permet de formaliser une hypothèse ou une prédiction. Le processus de modélisation peut être vu comme un processus itératif dont l'objectif

est de faire évoluer un modèle initial jusqu'à obtenir un modèle en adéquation avec les objectifs de simulation, représentatif d'un système réel (Campos et al., 2004). Ce processus itératif passe par un certain nombre d'étapes, que Sargent (2005) décompose ainsi (*cf.* figure 1.13) : l'expérimentation, qui permet de rassembler des données à propos du système étudié (l'objet d'étude) ; la formulation d'hypothèses à partir des données récoltées et la théorisation du système observé ; la réalisation du modèle, qui passe lui-même par plusieurs étapes (modèle conceptuel, sa spécification et son implémentation) ; enfin, la simulation est une expérimentation qui génère des données comparables à celles du système source. Ces données peuvent aider à mieux comprendre le fonctionnement du système réel, de confronter les résultats aux observations du système réel et ainsi, servir à émettre d'autres hypothèses à intégrer dans le processus de modélisation.

Si le modèle possède un niveau de justesse en accord avec les objectifs fixés, les résultats peuvent être exploités dans une démarche descriptive ou prédictive. Dans une démarche descriptive, la simulation du modèle peut permettre de se passer de la prise de mesure, celle-ci étant parfois coûteuse ou complexe à réaliser. Dans une démarche prédictive, la simulation du modèle peut permettre de prévoir le comportement du système réel sous certaines conditions, de pousser le modèle dans ses retranchements afin d'analyser son comportement dans des conditions extrêmes, peut-être trop coûteuses ou trop dangereuses à mettre en place dans un processus expérimental.

Au vu de l'importance accordée à la simulation, il est essentiel de spécifier les différentes étapes du processus de modélisation et de simulation. L'activité de M&S peut être considérée comme une expérience numérique, au même titre qu'une expérience scientifique menée au sein d'une discipline expérimentale. À ce titre, elle doit être traitée avec la même rigueur et donc être assujettie à un protocole expérimental afin de pouvoir permettre sa reproduction et sa vérification (Duboz et al., 2012). Si pour traiter ce sujet une méthodologie complète est nécessaire, nous traitons dans ce chapitre uniquement des méthodes formelles permettant de spécifier les modèles. L'utilisation d'un formalisme pour la description d'un modèle constitue une première étape indispensable en vue de son partage et par extension de sa reproductibilité. En effet, une spécification formelle fournit une notation non-ambiguë qui permet de décrire précisément la structure et le comportement d'un modèle.

Ce chapitre est organisé comme suit. Après avoir défini la nature et le rôle d'un formalisme, nous donnons une vue d'ensemble des formalismes existants adaptés à la modélisation et à la simulation. Notre but étant la spécification d'un système multi-agent, nous présentons le formalisme qui selon nous rassemble les qualités nécessaires, c'est-à-dire le formalisme *Parallel Discrete Event system Specification* (PDEVS). Nous dressons enfin un état de l'art sur les différents travaux formels autour de la simulation de systèmes multi-agents.

2 *Formalismes de modélisation*

Dans un contexte de modélisation, la spécification d'un modèle concerne la description de ses états et de son comportement. Celle-ci peut être effectuée à l'aide d'une approche

formelle ou d'une approche semi-formelle. Une spécification formelle est généralement basée sur un formalisme mathématique qui permet de définir une sémantique précise tandis qu'une spécification semi-formelle utilise plutôt un ensemble de notations généralement graphiques (telles que des diagrammes UML) accompagnées de précisions exprimées en langage naturel. Le langage naturel étant sujet à interprétation, la sémantique d'un modèle exprimé sous cette forme peut être ambiguë. Ainsi, certaines méthodologies de conception des SMAs que nous avons évoquées dans le chapitre I (section 3.5) telles que GAIA (Wooldridge et al., 2000), PASSI (Cossentino et Potts, 2001) ou ADELFE (Picard et Gleizes, 2004) sont considérées comme semi-formelles puisqu'elles se basent sur le langage UML.

Une approche formelle permet de décrire précisément le comportement attendu à travers une notation mathématique. Notre objectif étant de décrire de façon univoque le comportement des modèles, nous nous intéressons dans cette section aux outils mathématiques adaptés à leurs spécifications. Nous donnons une définition de ce qu'est un formalisme avant de réaliser un état de l'art de ceux existants. Ceci nous permet d'acquérir une vue d'ensemble et d'orienter notre choix vers le formalisme qui nous semble le plus à même d'exprimer le paradigme agent.

2.1 *Qu'est-ce qu'un formalisme ?*

Si un paradigme de modélisation permet de représenter un système grâce à un ensemble de concepts, un formalisme permet d'établir ces concepts à travers une syntaxe et une sémantique précise. Plusieurs paradigmes peuvent être exprimés par un même formalisme de modélisation. Pour Hardebolle (2008), le formalisme désigne une convention de notation qui permet de s'entendre sur la forme et sur le sens. Un formalisme définit donc un langage, qui est lui-même défini par une syntaxe abstraite, une syntaxe concrète et une sémantique.

La syntaxe *abstraite* définit les concepts du langage et leurs relations. Pour un langage de programmation par exemple, la syntaxe abstraite est généralement exprimée durant la phase de compilation à l'aide d'un arbre syntaxique. Plus généralement, la syntaxe abstraite d'un langage peut s'exprimer à l'aide d'un langage de méta-modélisation, comme EBNF (ISO et IEC, 1996) pour un langage de programmation, ou ECORE (Steinberg et al., 2008) pour un langage de modélisation. La syntaxe *concrète*, elle, définit une notation (textuelle et/ou graphique) qui associe à chaque concept abstrait une représentation concrète. Afin de mieux comprendre la différence entre les deux types de syntaxe, considérons l'exemple suivant. Une opération arithmétique dans un langage de programmation est composée d'un opérateur et d'opérandes, qui forment une expression renvoyant une valeur. Les concepts que nous venons de décrire forment la syntaxe abstraite. Si nous décidons de représenter l'opérateur d'addition par le symbole '+', les opérandes par des nombres entiers relatifs et la fin d'une expression par le symbole ';', nous définissons la syntaxe concrète suivante pour l'opération d'addition : '`40+2;`'. Pour un formalisme, le même principe s'applique : les concepts sont associés à des définitions mathématiques.

La sémantique, elle, permet d'exprimer la façon dont la composition des éléments

syntaxiques doit être comprise – ou interprétée. Pour un langage, la définition de la sémantique peut prendre différentes formes, plus ou moins précises : langage naturel, définitions mathématiques ou langage informatique. Comme nous l'avons évoqué, c'est en fonction de ce niveau de précision que l'approche peut être qualifiée de formelle ou non. Pour un formalisme, la sémantique est toujours formelle. Nous nous intéressons donc uniquement aux sémantiques formelles, qui peuvent appartenir à trois grandes familles :

- la *sémantique opérationnelle* définit un modèle mathématique permettant de décrire le comportement du modèle au cours de son exécution en terme de suite d'états. Un système de transition d'états est généralement utilisé.
- la *sémantique axiomatique* ne considère pas une succession d'états, mais plutôt des propriétés logiques qui doivent être vérifiées sur l'état de la mémoire avant et après exécution. Il s'agit de spécifier les propriétés attendues, qu'il est par exemple possible d'exprimer en logique de Hoare ou à l'aide du langage de contraintes OCL (Object Management Group, 2014).
- la *sémantique dénotationnelle* associe une fonction mathématique à chaque élément syntaxique, et a pour but de définir l'effet de son exécution. La sémantique de la composition de plusieurs éléments syntaxiques est la composition de leurs fonctions respectives.

Le fait d'utiliser une sémantique formelle offre de nombreuses perspectives. Celle-ci permet en effet la vérification formelle d'un modèle, qui offre la possibilité de démontrer mathématiquement que le modèle vérifie un certain nombre de propriétés. Chaque style de sémantique formelle est plus ou moins adapté à la vérification (Blazy, 2008). Ainsi, une sémantique axiomatique est particulièrement adaptée à la vérification d'un modèle. Celle-ci consiste en effet à vérifier la validité des propriétés logiques définies par la sémantique. La sémantique opérationnelle permet également de vérifier certaines propriétés sémantiques tandis que les concepts mathématiques utilisés dans une sémantique dénotationnelle sont plus difficilement manipulables dans un assistant à la preuve. Cette discipline, aussi appelée *model-checking*, est une activité du domaine de la V&V (Validation et Vérification). La vérification est un principe essentiel pour rendre crédible un résultat de simulation, d'autant plus que Stodden et al. (2013) établissent un lien entre la reproductibilité et la V&V, les activités de cette dernière étant considérées comme des étapes pour aboutir à la reproductibilité. Malgré ces conclusions, les aspects de vérification ne sont pas abordés dans le cadre de cette thèse. Nous soulignons toutefois le fait que la vérification n'est pas incompatible avec l'approche que nous proposons dans la suite de ce mémoire puisque nous utiliserons un formalisme de modélisation doté d'une sémantique opérationnelle.

Maintenant que nous avons clarifié ce qu'est un formalisme et ce qu'il peut apporter, nous étudions dans la section suivante les différents formalismes de M&S afin de voir lesquels sont les plus à même de représenter notre vision du paradigme agent, que nous avons détaillée dans le chapitre I.

2.2 *Classification des formalismes*

Il existe un certain nombre de formalismes qu'il est possible d'utiliser pour la spécification d'un système complexe. Parmi ces formalismes, tous ne partagent pas les mêmes

objectifs ni les mêmes propriétés. Certains sont plus ou moins adaptés à la description d'un phénomène en fonction des paradigmes de modélisation qu'ils permettent de représenter, comme le suggèrent Uhrmacher et al. (2006a) :

> « *Mostly, it is less the question whether a formalism is able to express certain phenomena, but how easily this can be done.* »[1]

Afin de nous guider dans notre choix pour le paradigme agent, nous reprenons la classification proposée par Ramat (2003) qui tient compte de plusieurs aspects pour synthétiser différents formalismes permettant de spécifier des systèmes dynamiques (tableau 2.1). Les critères de distinction concernent essentiellement la dichotomie entre discret et continu selon plusieurs aspects : l'espace, le temps et les changements d'états. Grâce à cette classification, il suffit de répondre aux questions suivantes à propos du système que l'on souhaite modéliser pour être aiguillé vers le formalisme adéquat :

- les variables d'états du système évoluent-elles de manière continue ou discrète ?
- le temps est-il représenté de manière continue ou discrète ?
- l'espace a-t-il une importance ? Si oui, est-il discret ou continu ?

Pour le paradigme agent, l'espace joue un grand rôle et les changements d'états s'opèrent de manière discrète. En ce qui concerne le temps, le paradigme dominant au sein de la simulation de systèmes multi-agents est celui du temps discret. Néanmoins, le paradigme à évènements discrets est également répandu (Maione et Naso, 2003 ; Su, 2013). Dans le cadre de cette thèse, c'est au paradigme à évènement discret que nous nous intéressons particulièrement, puisqu'il permet une représentation continue du temps, mais où les changements d'états sont discrets. Aussi, ce choix n'empêche pas de représenter des modèles SMA à temps discret, ce dernier étant un cas particulier de l'évènement discret.

Changement d'états	**Temps**	**Espace**	**Formalisme**
Continu	Continu	Absent	Equations différentielles ordinaires
		Continu Discret	Equations aux dérivés partielles
	Discret	Absent	Equations aux différences
		Continu Discret	Equations aux différences et spatialisées
Discret	Continu	Absent Continu Discret	Modèles à évènements discrets
	Discret	Absent	Automates à états finis
		Continu	Modèles à temps discret
		Discret	Automates cellulaires

Tableau 2.1 Classification de formalismes, proposée par Ramat (2003), en fonction de la représentation de l'espace, du temps et des changements d'états.

Les systèmes à évènements discrets, ou *discrete event systems* (DES) en anglais, permettent de faire abstraction de l'écoulement continu du temps afin de réagir uniquement à un

1. « La question n'est pas de savoir si un formalisme est capable d'exprimer certains phénomènes, mais à quel point il est facile de le faire. » (traduit de Uhrmacher et al. (2006a)).

ensemble d'évènements particuliers dans l'évolution d'un système. Un évènement se produit de manière instantanée et entraîne une transition d'état ; il peut survenir de manière spontanée (*e.g.* une alarme qui retentit) ou lorsqu'un certain nombre de conditions sont réunies (*e.g.* un élément passe de l'état liquide à l'état solide) (Cassandras et Lafortune, 2008). Contrairement à une modélisation à temps discret où le système est observé à pas de temps constant, le paradigme de modélisation à évènements discrets permet d'éviter d'observer le système si son état n'en présente pas l'intérêt. Cet aspect est particulièrement intéressant puisqu'il permet également d'éviter d'effectuer des calculs superflus lors de la simulation du système.

Les systèmes multi-agents, qui peuvent faire interagir un nombre important d'entités au sein d'un même modèle de simulation ne dérogent pas à cette règle. En dehors des aspects de performance d'exécution, l'évènement discret présente l'avantage de pouvoir faire interagir des agents dont les échelles de temps diffèrent. En contrepartie, l'évènement discret demande un effort de modélisation plus important et pose certaines problématiques que nous aborderons dans la suite de ce mémoire.

La classification proposée par Ramat (2003) n'indique pas de formalisme précis pour exprimer des modèles à évènements discrets (*cf.* Tableau 2.1). Il en existe un certain nombre, nous pouvons citer par exemple les automates à états finis, les réseaux de Petri, leurs variantes respectives temporisées ou encore les chaînes de Markov. Notre choix s'est porté sur la théorie développée par Zeigler et al. (2000) sur la M&S de systèmes dynamiques et couplés à évènements discrets à travers le formalisme *Discret EVent system Specification* (DEVS) pour les raisons suivantes :

- Au-delà de permettre la spécification de la dynamique d'un système, il permet une modélisation structurelle à travers la description modulaire et hiérarchique des systèmes.
- La sémantique opérationnelle de DEVS a été définie à travers les simulateurs abstraits. La simulation des modèles est totalement décorrélée de leur spécification ; il y a une séparation explicite entre la modélisation et la simulation.
- Il est considéré comme un formalisme pivot. Son caractère universel (Vangheluwe, 2000) lui permet de représenter une large classe d'autres systèmes dynamiques. Il peut par exemple représenter des systèmes à temps discret comme les automates cellulaires ou fournir une approximation (aussi proche que voulue) de systèmes continus tels que des équations différentielles (Kofman, 2002).
- Proposé dès 1976 (Zeigler, 1976), une large communauté s'est formée et a contribué à faire évoluer et à enrichir le formalisme, ainsi qu'à concevoir et à optimiser de nombreux outils informatiques permettant de simuler des modèles spécifiés avec DEVS.

L'avant dernier point soulève la question de la multi-modélisation. La multi-modélisation suggère qu'un système puisse être modélisé à l'aide de différents paradigmes ou de différents formalismes. La multi-modélisation ouvre de nombreuses perspectives. Elle permet de faire coexister des modèles différents au sein d'une même simulation. Ceci peut être motivé par la volonté de réutiliser des modèles existants exprimés à l'aide d'outils différents. Ou, par exemple, lorsqu'un seul formalisme ne permet pas d'exprimer toutes les propriétés nécessaires à la modélisation du système étudié. Si ce dernier est décomposable en sous-

systèmes, il est possible d'exprimer chacun d'eux à l'aide d'outils différents. Enfin, la motivation peut également naître du paradigme plutôt que du système lui-même. Si la richesse d'un paradigme de modélisation impose qu'un nombre important de propriétés soient satisfaites, il peut être difficile de trouver – ou de définir – un seul formalisme qui puisse réunir toutes les qualités recherchées. La solution peut alors venir de la multi-modélisation.

Néanmoins, la multi-modélisation est un exercice compliqué. Cela présuppose que les différents formalismes utilisés soient compatibles à la fois sur les éléments manipulés mais également d'un point de vue sémantique. Si c'est le cas, plusieurs méthodes sont envisageables :

- utiliser un formalisme capable de représenter un autre formalisme afin d'« encapsuler » le second dans le premier ;
- définir un nouveau formalisme réunissant les propriétés des formalismes avec lesquels le modélisateur souhaite travailler.

La propriété d'universalité du formalisme DEVS permet d'envisager la première solution. Comme nous allons le voir dans la suite de ce mémoire, lorsque certaines propriétés font défaut, des extensions du formalisme peuvent être proposées afin d'enrichir ses propriétés.

Afin de mieux comprendre les raisons qui nous poussent vers la théorie de la M&S développée par B. P. Zeigler, nous dédions la section suivante à la présentation du formalisme DEVS, ou plus particulièrement de l'une de ses variantes : le formalisme PDEVS. Au même titre que Uhrmacher et Schattenberg (1998) ou encore, Duboz et al. (2012), nous pensons que celui-ci forme un excellent candidat pour représenter le paradigme agent.

3 *Le formalisme PDEVS*

Le Pr. B. Zeigler (1976) a développé dans un premier ouvrage une théorie de la modélisation et de la simulation (TMS), aujourd'hui étendue (Zeigler et al., 2000), permettant de spécifier des systèmes (continus, discrets ou hybrides) grâce au formalisme DEVS (*Discret EVent system Specification*). Le formalisme DEVS permet de décrire les systèmes de manière modulaire et hiérarchique à travers la notion de modèles couplés. Un modèle *atomique* décrit le comportement autonome d'un système ; un modèle *couplé* décrit le système en tant que réseau de composants. Sa propriété de fermeture sous composition montre que n'importe quel modèle couplé peut lui-même être représenté par son résultant atomique, et donc que le couplage n'influence en aucune manière le comportement des modèles. Il bénéficie également d'une sémantique formelle, ce qui permet d'exécuter les modèles sur une implémentation du simulateur DEVS de manière non-ambigüe. Par ailleurs, l'un des atouts du formalisme réside dans la séparation explicite de la modélisation et de la simulation.

Une vingtaine d'années après l'introduction du formalisme DEVS, Chow et Zeigler (1994) proposent l'extension *Parallel DEVS* (PDEVS) pour répondre à la difficulté de conceptualiser certains phénomènes à travers des simulations DEVS classiques. La problématique

est essentiellement liée à la gestion des évènements simultanés. La prise en charge des évènements simultanés avec le formalisme DEVS classique est difficile, c'est pourquoi nous présentons et utilisons PDEVS dans la suite de ce mémoire, puisqu'il permet au modélisateur de gérer la simultanéité des évènements au niveau du modèle.

Le formalisme PDEVS est suffisamment abstrait pour pouvoir représenter d'autres types de formalismes (Vangheluwe, 2000) et permet ainsi de faire interagir des modèles exprimés dans des formalismes différents au sein d'une même simulation. Un nombre important d'extensions du formalisme PDEVS ont vu le jour depuis son introduction afin d'enrichir ses propriétés. La suite de cette section présente les définitions formelles des modèles atomiques et couplés, avant de s'intéresser à la sémantique du formalisme et aux extensions indispensables pour représenter des systèmes multi-agents.

3.1 *Modèle atomique*

Le formalisme PDEVS permet de spécifier un système dynamique à travers la définition d'un modèle atomique. Sa structure est décrite par l'octuplet suivant :

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

où X est l'ensemble des valeurs et des ports d'entrée et Y est l'ensemble des valeurs et des ports de sortie tels que :

$$X = \{(p, v) \mid p \in IPorts, v \in X_p\} \qquad Y = \{(p, v) \mid p \in OPorts, v \in Y_p\}$$

avec $IPorts$ l'ensemble des ports d'entrée ; X_p l'ensemble des valeurs possibles sur le port d'entrée p ; $OPorts$ l'ensemble des ports de sortie ; et Y_p l'ensemble des valeurs possibles sur le port de sortie p.

S est l'ensemble des états séquentiels du modèle. Le couple (s, e) forme un état total où e représente le temps écoulé dans l'état courant depuis la dernière transition. L'ensemble des états totaux est noté Q :

$$Q = \{(s, e) \mid s \in S, e \in \mathbb{R}^+, 0 \leq e \leq ta(s)\}$$

La fonction $ta(s)$ détermine le temps pendant lequel le modèle reste dans l'état courant si aucun évènement externe ne vient perturber l'évolution autonome du modèle :

$$ta : S \to \mathbb{R}^+_{0,\infty}$$

Dans le cas où $ta(s) = 0$, l'état courant est dit *transitoire*. En revanche si $ta(s) = \infty$, l'état est *passif*. Le modèle reste alors dans le même état indéfiniment, à moins qu'un évènement externe ne vienne perturber cet état.

Lorsqu'un ou plusieurs évènements externes se présentent, un nouvel état est engendré instantanément. Ce nouvel état est calculé par la fonction de transition externe, qui reçoit un

sac[2] d'évènements :

$$\delta_{ext} : Q \times X^b \to S$$
$$s' = \delta_{ext}(s, e, x^b)$$

Ainsi, le modèle peut changer d'état en ayant connaissance de toutes les entrées simultanées en une seule transition.

Lorsque le temps écoulé $e = ta(s)$, l'état courant expire. Avant de calculer son futur état, le modèle produit un sac de sorties pour l'état courant via la fonction de sortie $\lambda(s)$:

$$\lambda : S \to Y^b$$

Le futur état du modèle est ensuite calculé par la fonction de transition interne :

$$\delta_{int} : S \to S$$
$$s' = \delta_{int}(s)$$

Les trois fonctions λ, δ_{int} et ta forment le comportement *autonome* du modèle. Cependant, l'arrivée d'un évènement externe sur l'une des entrées du modèle peut venir perturber ce cycle autonome.

Une troisième fonction de transition, δ_{con}, appelée *transition de confluence*, permet de gérer de manière explicite les collisions d'évènements. Une collision se produit lorsque pour une date donnée t, un modèle atomique planifie son activation de manière autonome et reçoit au même moment un ou plusieurs évènements externes. Dans ce cas, la fonction de transition de confluence est utilisée pour calculer le nouvel état :

$$\delta_{con} : S \times X^b \to S$$

Si le modélisateur ne définit pas δ_{con}, un comportement par défaut est appliqué : l'activation séquentielle de δ_{int} et de δ_{ext}. La solution apportée par le formalisme PDEVS pour gérer les interférences donne la possibilité de modéliser un comportement propre à chaque modèle face à la simultanéité. C'est une différence majeure avec le formalisme DEVS classique puisque ce dernier définit un comportement par défaut en choisissant une approche séquentielle s'il y a des interférences.

Nous illustrons avec la figure 2.1 un modèle PDEVS atomique, composé de ports d'entrées et de sorties. Cette représentation rappelle sans aucun doute celle que nous avons donnée pour illustrer la notion de système dynamique (*cf.* figure 1.1). Ici, l'ensemble des entrées du système X_M est formé par l'union des couples ports/valeurs d'entrées ; l'ensemble des sorties Y_M est formé par l'union des couples ports/valeurs de sorties. La structure interne du système dynamique, composé de l'état et de la dynamique est également formalisé. L'état est formé par le couple $(s, e) \in Q$ et la dynamique est formée par les fonctions de transitions,

2. Un sac (en anglais *bag*), aussi appelé *multiensemble* est une généralisation du concept d'ensemble. À l'inverse d'un ensemble, un multiensemble autorise plusieurs occurrences d'un même élément.

la fonction de sortie et la fonction d'avancement du temps.

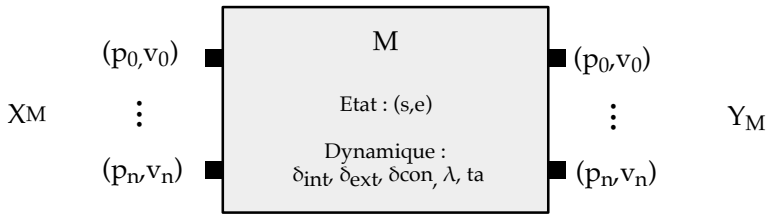


Figure 2.1 Représentation graphique d'un modèle atomique PDEVS.

Exemple : un modèle de sablier

À des fins pédagogiques, nous donnons ci-dessous un exemple de spécification d'un modèle atomique représentant un sablier, pour lequel le temps d'écoulement du sable est de 180 secondes. Le modèle se compose d'un port d'entrée (?*renverse*) qui permet de simuler le renversement du sablier. Nous considérons que chaque message envoyé sur le port d'entrée ?*renverse* correspond à une rotation de 180° et que plusieurs personnes peuvent réaliser une rotation au même instant. À l'état initial s_0, la totalité du sable est dans le bulbe inférieur. Lorsque le sablier est renversé (un message d'entrée arrive), le sable s'écoule du bulbe supérieur au bulbe inférieur. Lorsqu'il n'y a plus de sable dans le bulbe supérieur, le modèle envoie une sortie sur le port !*ecoule*. La variable e (qui représente le temps écoulé depuis la dernière transition) permet de connaître la quantité de sable qui s'est déjà écoulée avant que le sablier ne soit renversé. Cette variable nous permet de prendre en compte le fait que le sablier peut être renversé à tout moment et d'ajuster le temps d'écoulement en conséquence. L'utilisation de PDEVS (qui active une seule fois δ_{ext} avec tous les messages simultanés en paramètre) nous permet de calculer facilement la nouvelle orientation du sablier. Si un nombre pair de messages arrive sur le port d'entrée, le sablier reste dans la même position. Pour savoir si un changement d'état est nécessaire, il suffit de vérifier si la norme du sac d'entrées est paire ou impaire.

$$S = \{(phase, \sigma) \mid phase \in \{ecoulement, attente\}, \sigma \in \mathbb{R}^{+}_{0,\infty}\}$$
$$X = \{(?renverse, v)\} \quad Y = \{(!ecoule, v)\} \quad s_0 = (attente, \infty)$$

$$ta(s) = \sigma$$
$$\delta_{int}(phase, \sigma) = s_0$$
$$\lambda(phase, \sigma) = \begin{cases} (!ecoule, \emptyset) & \text{si } phase = ecoulement \\ \emptyset & \text{si } phase = attente \end{cases}$$

$$\delta_{ext}(attente, \sigma, e, x^b) = (ecoulement, 180)$$

$$\delta_{ext}(ecoulement, \sigma, e, x^b) = \begin{cases} (ecoulement, e) & \text{si } ||x^b|| \text{ est pair} \\ (ecoulement, \sigma - e) & \text{si } ||x^b|| \text{ est impair} \end{cases}$$

$$\delta_{con}(ecoulement, \sigma, e, x^b) = \begin{cases} (attente, \infty) & \text{si } ||x^b|| \text{ est pair} \\ (ecoulement, e) & \text{si } ||x^b|| \text{ est impair} \end{cases}$$

Dans la section suivante, nous étudions à travers la définition du modèle couplé la manière dont le formalisme PDEVS permet de former un réseau de composants.

3.2 *Réseaux de composants*

Le formalisme PDEVS permet de décrire les systèmes de manière modulaire et hiérarchique à travers les modèles couplés, qui permettent de décrire un système comme un ensemble de composants interconnectés à travers leurs ports. Le modèle couplé est définit par le septuplet suivant :

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,j}\} \rangle$$

où les entrées/sorties sont semblables à celles d'un modèle atomique avec X, l'ensemble des valeurs et des ports d'entrée et Y, l'ensemble des valeurs et des ports de sortie tels que :

$$X = \{(p, v) \mid p \in IPorts, v \in X_p\} \qquad Y = \{(p, v) \mid p \in OPorts, v \in Y_p\}$$

D est l'ensemble des identifiants des composants M_d, soit $\{M_d \mid d \in D\}$. Un composant M_d est un modèle PDEVS atomique ou couplé.

Le rôle d'un modèle couplé est de mettre en relation les entrées et les sorties de ses composants à travers des couplages. Trois types de couplages existent. Le premier, appelé *couplage interne*, met en relation deux composants du modèle couplé. L'un des ports de sortie du premier composant est lié à un port d'entrée du second composant. Les deux autres types de couplage lient les composants du modèle couplé à ses propres ports. Le *couplage d'entrée externe* connecte un port d'entrée du modèle à un port d'entrée d'un composant. Enfin, le *couplage de sortie externe* lie un port de sortie d'un composant à un port de sortie du modèle.

Les couplages sont représentés à l'aide de l'ensemble I_d où pour chaque $d \in D \cup \{N\}$, I_d représente les composants qui *influencent* d et où I_d est définit par :

$$I_d \subseteq D \cup \{N\} \qquad \text{avec } d \notin I_d$$

À l'aide de cet ensemble, les couplages sont donnés par la fonction de mise en correspon-

dance $Z_{i,d}$. Ainsi, pour chaque $d \in D \cup \{N\}$ et pour chaque $i \in I_d$, on a :

$$\begin{aligned} Z_{i,d} &: X_N \to X_d && \text{si } i = N \\ Z_{i,d} &: Y_i \to Y_N && \text{si } d = N \\ Z_{i,d} &: Y_i \to X_d && \text{si } i \neq N \text{ et } d \neq N \end{aligned}$$

Z_{N,d_1} correspond donc aux couplages d'entrée externe entre N et d_1, $Z_{d_2,N}$ aux couplages de sortie externe entre d_2 et N, et enfin, Z_{d_1,d_2} donne les couplages internes entre les composants d_1 et d_2.

Exemple de spécification d'un modèle couplé

Afin d'éclaircir le concept de réseaux de modèles, nous donnons un exemple illustré par la figure 2.2 d'un PDEVS couplé N composé de deux PDEVS atomiques M_1 et M_2, formellement défini de la manière suivante :

$$\begin{aligned} D &= \{M_1, M_2\} & IPorts_N &= \{in_1\} \\ OPorts_N &= \{out_1, out_2\} & Z_{N,M_1}(in_1) &= in_1 \\ I_N &= \{M_1, M_2\} & Z_{M_1,M_2}(out_1) &= in_1 \\ I_{M_1} &= \{N\} & Z_{M_1,N}(out_1) &= out_1 \\ I_{M_2} &= \{M_1\} & Z_{M_2,N}(out_1) &= out_2 \end{aligned}$$

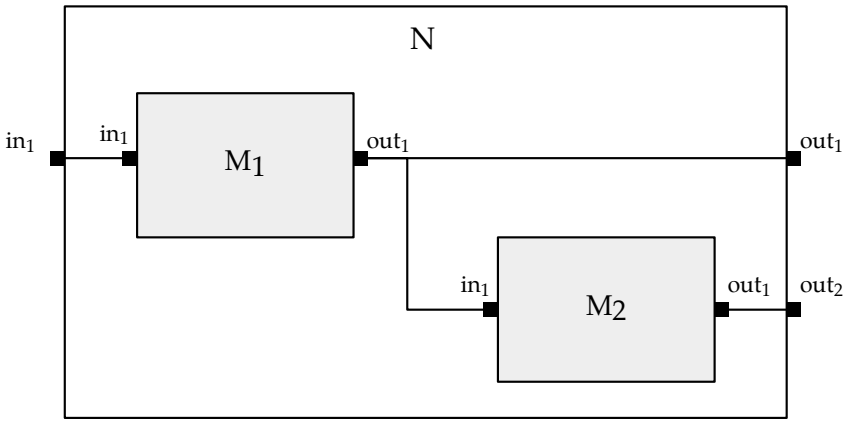


Figure 2.2 Représentation graphique d'un modèle couplé DEVS N composé de deux modèles atomiques M_1 et M_2 avec leurs couplages.

Le formalisme PDEVS garantit que le couplage de modèles atomiques est équivalent à un modèle PDEVS grâce à la propriété de *fermeture sous composition*, qui autorise la composition hiérarchique de modèles.

Fermeture sous composition

Pour garantir une conception de modèles de manière hiérarchique et modulaire, Zeigler et al. (2000) propose les règles suivantes :

- un modèle de base est un composant hiérarchique ;
- un modèle couplé formé de composants hiérarchiques est lui-même un composant hiérarchique ;
- aucun autre composant ne peut être considéré comme composant hiérarchique.

Lorsque ces règles sont respectées, comme c'est le cas pour le formalisme PDEVS, il est possible de démontrer la propriété de fermeture sous composition en prouvant qu'un modèle couplé est équivalent à un modèle atomique.

Cette preuve est construite en représentant la structure du modèle couplé au sein d'un modèle atomique. La structure du réseau de modèles est alors associée à l'état d'un modèle atomique et la sémantique du réseau est définie à l'aide des fonctions de transitions du modèle atomique. Ainsi, n'importe quel modèle couplé spécifié à l'aide du formalisme PDEVS peut lui-même être représenté par son *résultant* atomique. Cette propriété permet notamment de garantir que le couplage de modèles n'a aucune influence sur les trajectoires d'états des modèles. Par ailleurs, elle permet la conception de systèmes hiérarchiques : chaque composant d'un modèle couplé peut être un modèle atomique ou un modèle couplé. Le lecteur intéressé par la preuve de fermeture sous composition peut se référer à l'ouvrage de Zeigler et al. (2000).

3.3 *Séparation de la modélisation et la simulation*

L'un des atouts majeur du formalisme PDEVS réside dans la séparation explicite de la modélisation et de la simulation. En effet, le formalisme bénéficie d'une sémantique formelle grâce à la définition des algorithmes de simulation basés sur les concepts suivants :

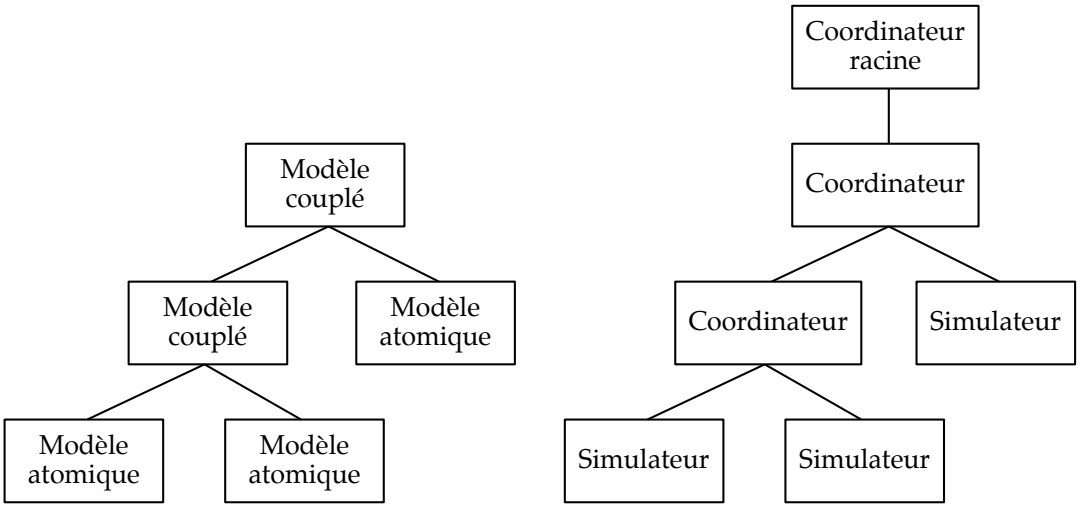
1. il existe un processeur de type simulateur pour chaque modèle atomique ;
2. chaque modèle couplé est associé à un processeur de type coordinateur ;
3. l'allocation de chaque processeur (simulateur ou coordinateur) suit la structure hiérarchique des modèles (les simulateurs gèrent les modèles atomiques et les coordinateurs gèrent les modèles couplés et cela pour chaque niveau de hiérarchie) ;
4. les processeurs adhèrent à un protocole de communication permettant de coordonner l'exécution de la simulation.

Ces concepts offrent des algorithmes de simulation génériques pouvant simuler n'importe quel modèle PDEVS à partir d'une plateforme de modélisation et de simulation.

3.3.1 *Structure hiérarchique*

Les trois premiers axiomes que nous venons d'énumérer impliquent qu'il existe un composant chargé de faire évoluer le modèle dans le temps. Puisqu'ils suivent la même hiérarchie que leurs modèles, les composants sont organisés sous forme d'arbre. La figure

2.3 établit la relation entre l'arbre de modélisation et l'arbre de simulation. Les feuilles de l'arbre représentent les simulateurs ou les modèles atomiques ; les nœuds représentent les coordinateurs ou les modèles couplés. Le *coordinateur racine*, en haut de la hiérarchie,



(a) Arbre de modélisation

(b) Arbre de simulation correspondant à l'arbre de modélisation

Figure 2.3 Relation entre arbre de modélisation (a) et arbre de simulation (b).

est en charge de l'horloge de simulation. Il est à l'origine du déclenchement de chaque étape de simulation. Comme son nom le suggère, le *coordinateur* est là pour coordonner l'exécution de ses enfants (simulateurs ou coordinateurs). Il gère également la répartition des messages d'entrée et de sortie en fonction des associations spécifiées au sein du modèle couplé associé. Enfin, le *simulateur* gère l'activation des fonctions formant la dynamique de son modèle atomique associé (δ_{int}, δ_{ext}, δ_{con}, λ, ta). Chaque simulateur ou coordinateur définit une variable tn indiquant sa prochaine date d'activation. Les simulateurs définissent cette variable d'après la fonction d'avancement du temps du modèle atomique associé. Les coordinateurs, eux, la définissent en calculant le tn minimum de leurs enfants.

La dynamique de chaque modèle étant confiée à un composant spécifique, il est nécessaire de coordonner l'ensemble de ces composants à travers un protocole de communication, comme le dernier axiome l'affirme.

3.3.2 *Protocole de communication*

Parmi les concepts énumérés par Zeigler et al. (2000) pour la conception des simulateurs, il est fait mention d'un protocole de communication (*cf.* figure 2.4) permettant aux processeurs de réaliser l'ensemble des calculs de manière coordonnée à travers l'utilisation de différents types de messages :

- Un message d'initialisation (i, t) est propagé depuis le coordinateur racine jusqu'à l'ensemble des simulateurs. Il est utilisé au début d'une simulation afin de permettre

aux processeurs d'initialiser leurs horloges de simulation.

- Le message de transition $(*, t)$ est également propagé depuis le coordinateur racine. Il permet de déclencher les différentes transitions des modèles qui ont planifié un évènement ou qui reçoivent des entrées.
- Le message de collecte $(@, t)$ est systématiquement propagé depuis le coordinateur racine avant le message de transition $(*, t)$. Il permet à l'ensemble des processeurs dont les modèles ont planifié un évènement interne de récupérer les sorties des modèles de manière simultanée.
- Le message de sortie (y, t) est propagé depuis un simulateur pour informer son coordinateur parent que son modèle associé a récupéré une sortie. En fonction des couplages définis par le modèle couplé parent, le coordinateur associé fait remonter l'information lorsque le destinataire dépend d'un autre coordinateur, ou propage vers le bas un message d'entrée aux processeurs concernés.
- Le message d'entrée (x, t) est propagé par un coordinateur lorsqu'il reçoit un message de sortie et que le destinataire du message fait partie de sa filiation. C'est-à-dire si le simulateur destinataire est accessible depuis les processeurs fils du coordinateur. Lorsque le message parvient au simulateur destinataire, l'évènement est ajouté au sac d'entrées.
- Le message (d, t) est propagé par les processeurs pour informer leurs parents qu'ils ont terminé leur traitement.

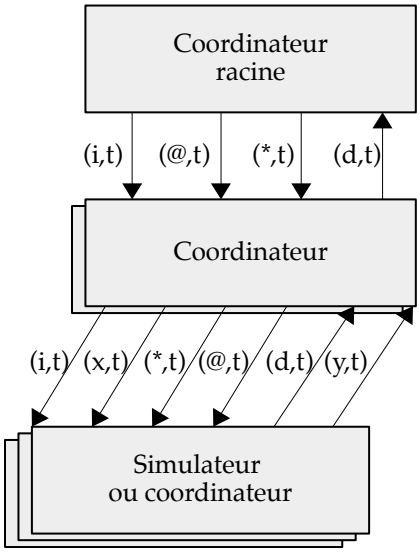


Figure 2.4 Représentation graphique des différents échanges de messages entre les processeurs dans une simulation Parallel DEVS.

Bien que les processeurs soient organisés de manière hiérarchique, les messages ne sont pas forcément propagés dans toute la hiérarchie. Les messages d'entrées et de sorties sont propagés en fonction des couplages spécifiés par le modélisateur et les messages de transition internes sont propagés uniquement vers les processeurs dont l'activation est imminente, c'est-à-dire vers les processeurs dont les modèles associés ont planifié un évènement pour le temps de simulation courant.

Chaque étape de simulation est réalisée en deux phases distinctes. La première, la phase de collecte, permet de récupérer l'ensemble des sorties des modèles pour l'état courant. Cette phase est cruciale car elle permet de rassembler plusieurs entrées destinées à un même modèle mais également d'identifier les conflits entre évènements internes et évènements externes pour un modèle. Durant cette phase, aucune transition n'est activée. C'est uniquement lors de la seconde phase, initiée par le message de transition $(*, t)$ propagé aux processeurs dont les modèles ont planifié un évènement interne et aux processeurs qui reçoivent un ou plusieurs évènements externes, que les fonctions de transitions sont activées. La transition de confluence est activée si le modèle a un évènement interne planifié et qu'il reçoit au moins un évènement externe. La transition externe est activée si aucun évènement interne n'est planifié mais que le modèle reçoit au moins un évènement externe. Enfin, la transition interne est activée si le modèle a un évènement interne planifié et qu'il ne reçoit aucun évènement externe. Ainsi, en plus d'apporter une solution aux interférences que provoquent les évènements simultanés, PDEVS offre l'avantage de pouvoir tirer profit du parallélisme intrinsèque aux modèles couplés sans pour autant s'orienter vers des algorithmes optimistes. À chaque étape de la simulation, tous les composants imminents, c'est-à-dire tous les composants qui reçoivent un évènement (interne ou externe) prévu pour le temps courant, peuvent être exécutés en parallèle. Cela reste vrai à la fois pour la phase de collecte des sorties et pour la phase de transition d'états.

Cette architecture de simulation a l'avantage de permettre l'interopérabilité des formalismes. Il est possible de définir de nouveaux modèles et leurs simulateurs puis de les intégrer au sein d'une simulation PDEVS à condition que les simulateurs adhèrent au protocole de communication. La généralisation de ce concept est appelé *DEVS Bus*.

3.3.3 *DEVS Bus*

Il est possible d'intégrer d'autres approches de modélisation dans un environnement de simulation PDEVS à travers un concept permettant l'interopérabilité entre formalismes. L'idée est d'encapsuler le formalisme à intégrer en l'exprimant sous une forme PDEVS et de définir l'algorithme de son simulateur associé. Le modèle exprimé à l'aide du nouveau formalisme peut alors être couplé conjointement avec d'autres modèles PDEVS. L'ensemble des modèles est donc simulé normalement à travers un coordinateur. Ce concept, illustré par la Figure 2.5, est appelé un *DEVS Bus*. Il présente l'intérêt de préserver de manière explicite

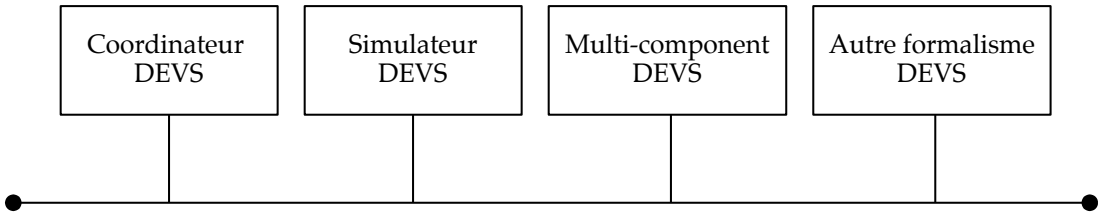


Figure 2.5 Représentation du concept DEVS Bus.

la séparation de la modélisation et de la simulation, et de faire d'un environnement PDEVS

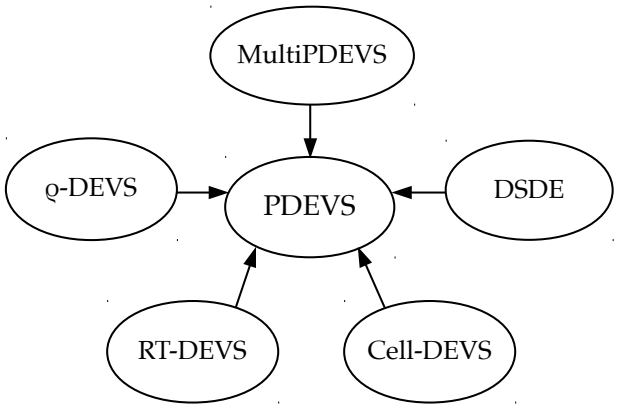


Figure 2.6 DEVS peut être étendu ou restreint pour former d'autres classes de systèmes. En cela il est considéré comme un environnement multi-formalisme.

un environnement multi-formalisme. L'intégration d'un nouveau formalisme élargit la classe de systèmes que l'on peut représenter en PDEVS et forme une *extension* ou une *restriction* du formalisme PDEVS (figure 2.6). Plusieurs extensions et restrictions ont été développées depuis l'introduction du formalisme DEVS classique, notamment : *Multi-component DEVS* (Zeigler et al., 2000) qui permet de définir des systèmes couplés non modulaires ; *Dynamic Structure DEVS* (Barros, 1995, 1996a,b) pour représenter des changements structurels au cours de la simulation ; ou encore *Cell-DEVS* (Wainer et Giambiasi, 2001) pour définir des automates cellulaires.

Le formalisme PDEVS est lui-même une extension du formalisme DEVS. Nous le plaçons néanmoins au centre étant donné qu'il modifie le protocole de communication défini par le formalisme DEVS. Les extensions compatibles sont donc celles basées sur le formalisme PDEVS et non sur le formalisme DEVS classique. Plusieurs extensions proposées à l'origine comme extension de DEVS classique ont d'ailleurs évoluées pour étendre ensuite le formalisme PDEVS. C'est le cas de *Dynamic Structure DEVS* (DSDEVS) (Barros, 1996b), proposé pour permettre d'effectuer des changements structurels de manière dynamique. Originellement proposée pour le formalisme DEVS classique, une variante de l'extension, appelée DSDE (Barros, 1998) a été proposée dans un souci de compatibilité avec PDEVS.

3.4 *Réseaux dynamiques*

Si PDEVS permet de concevoir un système à l'aide des modèles couplés, la spécification de ces derniers est statique et ne permet pas de représenter des changements structurels au cours de la simulation. Or, la possibilité de modifier l'interconnexion entre les composants d'un réseau de modèles, d'ajouter de nouveaux modèles ou d'en supprimer est essentielle pour refléter la dynamique de certains systèmes. En ce qui concerne les systèmes multi-agents, la naissance ou la mort d'individus et l'évolution des interactions entre individus sont des fonctionnalités primordiales. Un nombre important de travaux ont été proposés

pour que les membres d'un modèle couplé puissent modifier la structure du réseau, leur propre structure ou celle d'autres composants.

Muzy et Zeigler (2014) introduisent la notion de « point de contrôle unique », qui permet de caractériser les différentes contributions concernant la dynamique structurelle. Ainsi, dans un réseau dynamique à point de contrôle unique, un seul composant est responsable d'appliquer des modifications structurelles à chaque instant de simulation. Si ce dernier est identique tout au long de la simulation, il est qualifié de statique. S'il peut changer, le point de contrôle est dynamique.

Les travaux de Barros (1995, 1996a,b), à l'origine de l'extension DSDE, sont à placer dans le premier cas. Les modifications structurelles sont effectuées par un composant spécifique en charge d'appliquer ces changements. Cette centralisation peut être vue comme une façon trop restrictive de concevoir la dynamique structurelle (Barros, 2005). D'autres travaux sont basés sur une vision décentralisée des changements structuraux. C'est le cas de l'extension dynDEVS (Uhrmacher, 2001) et de ρ-DEVS (Uhrmacher et al., 2006b), qui plutôt que de proposer un mode centralisé des modifications structurelles, offre une approche réflexive en donnant la possibilité à chaque modèle d'engendrer une modification structurelle. Muzy et Zeigler (2014) placent cette contribution dans la catégorie de point de contrôle dynamique. Cependant, ce type de solution est plus complexe à mettre en œuvre compte tenu de la possibilité de non-déterminisme si plusieurs composants décentralisés souhaitent effectuer une modification structurelle simultanément (Barros, 2005).

L'approche décentralisée peut dans un premier temps sembler plus adaptée pour représenter un SMA, étant donné qu'une modification structurelle est la conséquence de l'action d'un agent. À ce sujet, Barros (2005) nous rappelle que le changement structurel ne dépend pas seulement de la volonté de l'agent, mais également d'autres entités. En effet, l'action prend place au sein de l'environnement et celle-ci peut être considérée valide ou non en fonction des lois de l'environnement (*cf.* sections I.3.2, I.4.3).

Nous présentons donc plus en détails l'extension DSDE, qui étend le formalisme PDEVS afin d'ajouter au modèle couplé le composant chargé de centraliser et d'appliquer les modifications structurelles du réseau. Ce composant, appelé modèle *exécutif*, est un modèle atomique dont l'état représente la structure du réseau de composants auquel il est rattaché. Il dispose donc d'une dynamique propre, lui permettent de décider de modifier la structure de manière autonome ou suite à une demande externe issue d'autres modèles.

L'extension DSDE apporte donc un certain nombre de modifications au modèle couplé, dont la structure est représentée par le quadruplet suivant :

$$DSDE_N = \langle X_N, Y_N, \chi, M_\chi \rangle$$

où N est le nom du réseau de composants ; X_N et Y_N sont les ensembles des valeurs et des ports d'entrées/sorties du réseau ; χ est le nom de l'exécutif du réseau dynamique et M_χ est le modèle exécutif. Ce dernier a vocation à détenir toutes les informations concernant les modèles et leurs couplages. Il est lui-même défini par un modèle PDEVS atomique auquel

la fonction ζ a été ajoutée :

$$M_\chi = (X_\chi, Y_\chi, S_\chi, \delta_{int_\chi}, \delta_{ext_\chi}, \delta_{con_\chi}, \lambda_\chi, \mathrm{ta}_\chi, \zeta).$$

ζ est la fonction de structure :

$$\zeta : S_\chi \to \Sigma^*$$

avec Σ^* l'ensemble des structures du réseau. Une structure $\Sigma \in \Sigma^*$, associée à chaque état de l'exécutif $s_\chi \in S_\chi$, est définie par le 4-uplet suivant :

$$\Sigma = \zeta(s_\chi) = (D, \{M_d\}, \{I_d\}, \{Z_{i,j}\})$$

où D est l'ensemble des identifiants des modèles du réseau pour l'état s_χ. Pour chaque $d \in D$, M_d est le modèle associé au composant d. Les couplages entre composants sont représentés à l'aide de l'ensemble I_d, où pour chaque $d \in D \cup \{\chi, N\}$, I_d est l'ensemble des composants capable d'influencer d, formellement défini par :

$$I_d \subset D \cup \{\chi, N\} \qquad \text{avec } d \notin I_d$$

Accompagnée de l'ensemble I_d, la fonction de mise en correspondance $Z_{i,d}$ permet de donner les couplages. Ainsi, pour chaque $d \in D \cup \{\chi, N\}$ et pour chaque $i \in I_d$:

$$\begin{aligned} Z_{i,d} &: X_N \to X_d && \text{si } i = N \\ Z_{i,d} &: Y_i \to Y_N && \text{si } d = N \\ Z_{i,d} &: Y_i \to X_d && \text{si } i \neq N \text{ et } d \neq N \end{aligned}$$

Dans un DSDE, toutes les informations sur la structure du réseau de modèles font parties de l'état de l'exécutif, ce qui implique que tout changement d'état via une des fonctions de transition δ_χ entraîne une modification de la structure du réseau, avec les contraintes suivantes :

1. Afin que l'exécutif ne puisse pas être supprimé du réseau, $\chi \notin D$.
2. Pour rendre déterministe les modifications structurelles, si χ reçoit un évènement externe, χ sera le seul modèle à recevoir une entrée à ce moment. Formellement :

$$Z_{k,\chi}(y) \neq \emptyset \implies Z_{k,i}(y) = \emptyset \qquad \begin{array}{r} \text{pour } k \in D \cup \{N\} \\ \text{et } \forall i \in I_k - \{\chi\} \\ \text{avec } \emptyset \equiv \text{ évènement nul} \end{array}$$

Etant donné que DSDE modifie la structure du réseau couplé, la propriété de fermeture sous composition a été prouvée par l'auteur afin qu'elle puisse toujours être vérifiée. Par ailleurs, l'auteur propose également la définition des simulateurs abstraits afin de préserver la séparation explicite de la modélisation et de la simulation. Le lecteur intéressé peut se référer à Barros (1995, 1996a,b, 1997, 1998) pour davantage de détails concernant ces aspects.

Les alternatives à DSDE permettant de définir des réseaux de composants dynamiques sont nombreuses :

- Comme évoquée en introduction de cette section, l'extension dynDEVS, qui a ensuite

évoluée vers ρ-DEVS (Uhrmacher, 2001 ; Uhrmacher et al., 2006b) adoptent une approche décentralisée. Les membres du réseau peuvent réaliser des modifications locales limitées à leur structure interne (espace d'état et fonctions). Les modifications concernant l'interface des membres du réseau, des couplages et des modèles sont réalisées au niveau du modèle couplé.

- Hu (2005) propose de manière non-formelle une approche décentralisée où les membres du réseau peuvent modifier la structure du réseau de manière séquentielle.
- Shang et Wainer (2006) proposent une alternative qui se base sur la spécification formelle de DSDE et qui reprend une partie des algorithmes de simulation de dynDEVS.
- Baati et al. (2007) définissent également un modèle couplé où la dynamique structurelle est gérée par un modèle atomique spécifique.

Parmi ces différentes solutions, DSDE et ρ-DEVS sont les solutions qui ont été les plus appliquées dans les différentes plateformes de M&S basées sur le formalisme DEVS ou PDEVS. D'autres travaux (Barros, 2005 ; Kim et Kim, 2000) font mention explicite des systèmes multi-agents comme une application possible des réseaux dynamiques. Avant d'étudier plus en détails les différents travaux concernant la spécification de SMAs basée sur le formalisme PDEVS et plus généralement de la formalisation des SMAs, nous soulignons dans une dernière sous-section un aspect intéressant du formalisme PDEVS : le fait que sa sémantique soit compatible avec le modèle d'action IRM4S (*cf.* section I.4.3).

3.5 *Une sémantique compatible avec IRM4S*

Comme nous l'avons déjà abordé, PDEVS a été proposé afin de permettre au modélisateur de gérer la simultanéité des évènements au niveau du modèle atomique Chow (1996). Un autre aspect intéressant de PDEVS réside dans le fait que PDEVS soit compatible avec la sémantique du formalisme IRM4S (Michel, 2007a) que nous avons présentée dans la section I.4.3 et qui a été définie pour apporter une solution aux phénomènes de divergence implémentatoire. Comme Michel (2007a) le fait très justement remarquer, la façon dont le modèle d'action IRM4S gère la simultanéité rappelle la manière dont le formalisme PDEVS gère les évènements simultanés à travers la fonction δ_{con}. Cette similitude entre IRM4S et PDEVS est particulièrement intéressante car elle fait de PDEVS un excellent candidat pour représenter le paradigme agent. Au même titre qu'IRM4S, PDEVS permet de rendre explicite le comportement du système face à des actions simultanées. Il permet ainsi de limiter les biais potentiels lors de l'implémentation d'un modèle, en renvoyant la gestion des évènements simultanés à la phase de modélisation. La sémantique de PDEVS étant bien définie à travers la notion de processeur, les biais liés à l'implémentation de différents simulateurs sont également réduits, généralisant ainsi la reproductibilité.

En superposant pour PDEVS et pour IRM4S les différentes étapes qui ont lieu lors du calcul du nouvel état du système, la similitude entre les deux formalismes devient évidente. La figure 2.7, qui met en parallèle la manière dont l'état du système évolue pour les deux formalismes permet de s'en rendre compte. Rappelons la sémantique de PDEVS : pour calculer les nouveaux états, les sorties de l'ensemble des modèles dont l'activation est imminente sont collectées dans un premier temps puis distribuées aux destinataires dans un

second temps. L'ensemble des modèles dont l'activation est imminente et l'ensemble des modèles qui ont reçu une entrée peuvent alors effectuer leur transition d'état. La sémantique d'IRM4S est similaire à celle de PDEVS. Pour calculer le nouvel état du SMA, les influences sont produites par les agents et/ou l'environnement dans un premier temps, puis celles-ci sont traitées pendant le calcul de réaction dans un second temps. La sémantique de PDEVS est donc compatible avec celle d'IRM4S. Cette particularité implique qu'il est possible d'effectuer de la multi-modélisation et de faire interagir des systèmes exprimés dans les deux formalismes, ou encore de représenter IR4MS à l'aide du formalisme PDEVS.

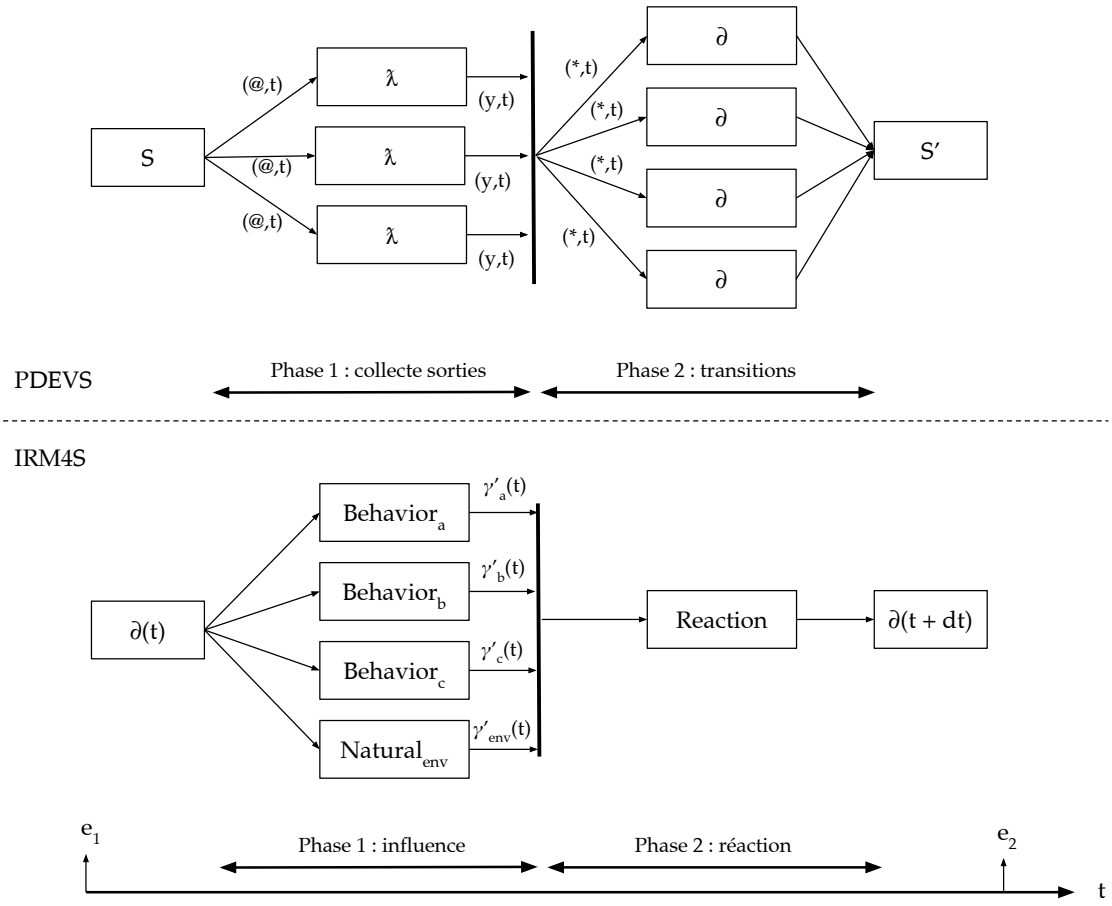


Figure 2.7 Parallèle entre le schéma d'évolution de l'état d'un système modélisé avec PDEVS et l'évolution de l'état d'un SMA modélisé avec IRM4S.

Que ce soit pour PDEVS ou IRM4S, la figure 2.7 permet également de visualiser le fait qu'un certain nombre d'étapes peuvent être réalisées de manière parallèle. Pour PDEVS, l'activation des fonctions de sorties λ ainsi que l'activation des fonctions de transitions peuvent être réalisées en parallèle, avec une barrière de synchronisation entre les deux phases. Pour IRM4S, le calcul de l'ensemble des influences peut être effectué de manière parallèle. En revanche, la phase de calcul de réaction est centralisée. Ce calcul peut cependant être réparti en fonction de la localité des influences (Michel, 2007a) ou en établissant une classification permettant d'identifier des groupes d'influences qui ne peuvent pas interférer entre elles (Weyns et Holvoet, 2003).

Le fait que PDEVS traite la simultanéité des évènements de manière similaire au modèle d'action IRM4S, qui a été pensé pour éliminer un critère de non-reproductibilité, constitue un critère important dans le choix d'un formalisme adapté à la spécification de SMAs. Cette section nous a également permis de souligner les autres qualités qui en font un choix d'autant plus pertinent :

- il permet de concevoir les systèmes dynamiques de manière modulaire et hiérarchique à travers la définition de réseaux de composants, qui bénéficient de la propriété de fermeture sous composition ;
- la communauté a fait évoluer le formalisme à travers la proposition d'extensions lui conférant de nouvelles propriétés, telle que la dynamique structurelle ;
- le caractère universel du formalisme permet de représenter d'autres formalismes ;
- il est muni d'une sémantique opérationnelle ;
- il existe une séparation explicite entre la modélisation et la simulation ;
- le protocole de simulation et le concept de « DEVS bus » permettent de faire interagir des modèles exprimés dans des formalismes différents, à condition que leur sémantique soit compatible.

Nous nous intéressons dans la prochaine section aux différents travaux formels qui ont été proposés pour la spécification de SMAs, y compris ceux basés sur le formalisme DEVS.

4 *Formalismes pour les systèmes multi-agents*

Nous avons mis en évidence au début de ce chapitre l'intérêt d'utiliser un formalisme pour spécifier un modèle. Muni d'une sémantique connue, le formalisme permet de spécifier des modèles univoques et par extension de favoriser la reproductibilité des modèles. En décrivant la structure du modèle et son comportement, il est possible de balayer toute ambiguïté au moment de l'implémentation. Un formalisme adapté au paradigme agent serait donc profitable à la communauté SMA. Comme le soulignent Bae et al. (2012), alors que les premiers modèles orientés agent étaient publiés sous forme de pseudo-code ou de codes sources, leur description est aujourd'hui en majorité basée sur des diagrammes UML (Object Management Group, 2015), voire des descriptions en langage naturel.

Il existe pourtant un nombre important d'études sur la formalisation des SMAs. Cependant, une partie de ces travaux de formalisation se basent sur une approche semi-formelle, notamment à travers l'utilisation du langage UML qui ne permet pas de décrire certains aspects de la sémantique. C'est le cas par exemple d'un certain nombre de travaux réalisés par la communauté des SMA centrés organisation, qui a donné naissance a plusieurs méta-modèles tels que AGRE Ferber et al. (2005), RIO Hilaire et al. (2000) ou encore MASQ Dinu et al. (2012) permettant de décrire la structure d'un SMA et que nous avons abordés dans la section I.3.5.

Des approches entièrement formelles existent, mais leur utilisation semble anecdotique. Aucun formalisme dominant au sein de la communauté n'est identifié (Bae et al., 2012). Nous tâchons de comprendre pourquoi dans la suite de cette section, en étudiant les différents

formalismes qui ont été spécifiquement proposés pour les systèmes multi-agents. Nous traitons dans un premier temps les formalismes influencés par le domaine de l'intelligence artificielle avant de nous intéresser aux travaux influencés par la théorie de la M&S. Nous nous appuyons pour cela sur l'état de l'art proposé par Bae et Moon (2016), auquel nous ajoutons d'autres travaux.

4.1 *Orientés comportement*

Bien que la communauté SMA ait donné naissance à de nombreux formalismes, il n'existe pas à notre connaissance de formalisme permettant de décrire un système multi-agent qui fasse consensus. La plupart des formalismes qui ont été proposés s'attachent à formaliser un aspect spécifique des SMA, tel que l'agent, l'environnement, les interactions ou l'organisation. Il est donc difficile d'envisager une spécification formelle complète d'un modèle de système multi-agent, étant donné que certains points sont laissés de côté.

La plupart des formalismes utilisés sont essentiellement destinés à décrire le comportement de l'agent. Ces formalismes sont généralement issus du domaine de l'intelligence artificielle. C'est le cas du formalisme *Beliefs-Desire-Intentions* (Rao et Georgeff, 1992) spécifié en logique temporelle ou encore des processus de décision Markoviens partiellement observables (Kaelbling et al., 1998), qui permettent tous deux de formaliser la façon dont l'agent choisit l'action à réaliser. Quelques travaux plus récents considèrent toutefois d'autres aspects de conception des SMAs. Le tableau 2.2, repris et complété de Bae et Moon (2016), dresse une liste de quelques travaux formels concernant les SMA. Pour chacun d'eux, sont indiqués la notation formelle utilisée ou définie ainsi que les différents aspects de conception des SMAs qui sont traités.

Si l'agent et l'interaction sont des aspects quasiment toujours abordés par ces travaux, l'environnement est plus souvent laissé de côté. La dynamique structurelle est également un aspect qui est rarement pris en compte par les formalismes listés dans le tableau 2.2. Notons que ce tableau ne prend pas en compte les aspects organisationnels tels que présentés section I.3.1. Les travaux issus de l'ingénierie système, eux, visent à guider la conception du modèle SMA d'un point de vue structurel et dynamique, et proposent donc généralement une vision plus globale.

4.2 *Orientés structure*

Les travaux que nous présentons dans cette section sont plutôt issus de l'ingénierie système. Ils visent à proposer une méthode globale de modélisation des SMAs, et sont généralement basés sur une vision systémique, avec des concepts d'organisation structurelle comme la composition/décomposition de systèmes. Dans ce cadre, un nombre important de travaux a vu le jour. La plupart sont basés sur la théorie de la M&S de Zeigler et al. (2000) et sur le formalisme DEVS. Nous donnons une place particulière à ces travaux que nous traitons en détails étant donné que nous avons souligné l'intérêt de ce formalisme pour représenter le paradigme agent. Nous évoquons cependant d'abord les travaux qui ne sont

Littérature	Notation	Aspect de conception SMA			
		Agent	Env.	Int.	Dyn.
Luck et DInverno (1995)		✓	✗	✓	✗
Silva (2005)	Spécification Z	✓	✗	✓	✗
Niazi et Hussain (2011)		✓	✗	✗	✗
Fisher et Wooldridge (1997)	Logique de croyance temporelle	✓	✗	✓	✗
Conrad et al. (2002)	Logique temporelle évolutive	✓	✗	✓	~
Zhu (2001)	Logique temporelle	✓	✓	✓	✗
Ricci et al. (2010)	Algèbre de processus	✓	✓	✓	✓
Brazier et al. (2000)	DESIRE	✓	✓	✓	✗
Lomazova (2000)	Réseaux de Petri imbriqués	✓	✗	✓	✗
Bauer et al. (2001)	AgentUML	✓	✗	✓	✗
Bentahar et al. (2004)	Network structure	✗	✗	✓	~
Chatfield et al. (2007)	Multi-formalisme	✓	✗	✓	✗
Marzougui et al. (2010)	Réseau de Petri agent	✓	✗	✓	✗
Wu et al. (2012)	ADAM	✓	✗	✓	✗
Weyns et Holvoet (2004)	Multi-formalisme	✓	✓	✓	✗
Michel (2007a)	IRM4S	✓	✓	✓	✗

Tableau 2.2 Aspects de conception (agent, environnement, interaction et dynamique structurelle) abordés par les travaux de formalisation des SMAs, repris et complété de Bae et Moon (2016). Les aspects traités sont mentionnés par le symbole '✓', partiellement par '~' et non traités par '✗'.

pas liés au formalisme DEVS.

Une des premières proposition est celle de (Ferber, 1995) avec le formalisme BRIC (*Basic Representation of Interacting Components*). Celui-ci permettait déjà la spécification formelle d'un SMA à travers l'utilisation des réseaux de Petri colorés pour la définition du comportement des agents, auxquels était associé une approche modulaire pour la définition de la structure du système. Une structure générique est proposée où le principe Influence/Réaction (Ferber et Müller, 1996) est utilisé. Néanmoins, BRIC ne permet pas de représenter une dynamique structurelle associée à la naissance d'un agent, à la mort d'un agent, ou à l'évolution des interactions.

Gruer et al. (2004) ont proposé le langage de spécification OZS, qui est muni d'une sémantique formelle. Cette approche se base sur Object-Z pour décrire les différentes entités, pour lesquelles le comportement est défini à l'aide de diagrammes d'états (*statecharts*). Les auteurs ont également développé des outils de validation et de vérification associés à la notation, ce qui est un avantage important en vue de la reproductibilité. Cependant, à notre connaissance aucune plateforme ne permet de faciliter l'exécution de modèles spécifiés à l'aide d'OZS.

L'intérêt du formalisme DEVS pour la spécification de systèmes multi-agents n'est pas nouveau et a fait l'objet de plusieurs propositions. Nous présentons dans la suite de cette section ces différents travaux.

4.2.1 *Une première analogie*

Les premiers travaux de formalisation des SMAs basés sur le formalisme DEVS semblent avoir été proposés par Uhrmacher et Arnold (1994) ainsi que Uhrmacher et Schattenberg (1998). Ces travaux mettent en évidence les similitudes entre les propriétés d'un agent et la structure d'un modèle DEVS. Ainsi, une première analogie est faite entre la perception d'un agent et la transition externe d'un modèle atomique. La fonction de sortie est considérée comme analogue à l'action et enfin, comme l'agent, le modèle atomique est capable de proactivité à travers sa transition interne.

Travaillant sur des problématiques de modélisation d'écosystèmes, leur besoin de représenter des structures dynamiques les poussent à proposer l'extension dynDEVS (Uhrmacher, 2001). dynDEVS est une alternative à DSDE, qui plutôt que de proposer un mode centralisé, offre une approche réflexive où chaque modèle peut engendrer une modification structurelle. Elle impose cependant que les ports soient définis de manière statique. ρ-DEVS (Uhrmacher et al., 2006a), une nouvelle extension est finalement proposée pour permettre de modifier les ports dynamiquement. Elle est basée cette fois sur le formalisme PDEVS.

L'utilisation d'une extension DEVS pour représenter une dynamique structurelle constitue un apport original vis-à-vis du formalisme BRIC (*cf.* section 4), qui permet de décrire des SMA dont la structure est statique. En revanche, bien que la simulation orientée agent (ABS) soit à la base de ces travaux, rien n'est mentionné à propos de la modélisation de l'environnement. Cette lacune a été récemment comblée à travers la définition de ML-DEVS (Steiniger et al., 2012), une extension de ρ-DEVS qui permet d'associer un comportement à un modèle couplé. Le modèle couplé joue alors le rôle d'environnement pour ses différents composants.

4.2.2 *GALATEA*

Dávila et Tucci (2000) proposent un langage de programmation spécifiquement conçu pour la modélisation d'agents. Le modèle qui en résulte est destiné à être simulé au sein d'un environnement DEVS, la plate-forme GALATEA (Dávila et Uzcátegui, 2000). Le langage est inspiré du langage GLIDER (Domingo, 1988 ; Domingo et Hernández, 1985) et par le principe Influence/Réaction (Ferber et Müller, 1996) en y apportant certaines modifications qui partagent certaines similitudes avec le modèle IRM4S (*cf.* I.4.3) :

- une variable temporelle, $\mathfrak{T}$, est également ajoutée ;
- les auteurs critiquent la fonction de perception telle qu'elle est formalisée par Ferber et Müller (1996) ($Perception_a : \Gamma \rightarrow P_a$; l'agent perçoit seulement les influences). Dávila et Tucci (2000) donnent la possibilité aux agents d'accéder aux variables de l'environnement.

Face à l'incohérence au niveau de l'évolution du système (états dynamiques incohérents), les auteurs s'orientent vers une autre solution en choisissant de combiner les deux phases influence/réaction en une seule. D'un point de vue temporel, l'application de cette phase est bien instantanée et règle donc le problème du modèle d'origine. Cependant, le modèle IRM4S qui règle cette problématique tout en préservant les deux phases est selon nous plus

clair, et plus en adéquation avec le principe même d'influence/réaction.

4.2.3 *Dynamic Structure Discrete Event Multiagent Systems*

L'idée de définir un SMA basé sur le formalisme DEVS a très tôt suscité l'intérêt d'un autre groupe de chercheurs, qui a poursuivi la réflexion de Uhrmacher (2001) afin de proposer une vision plus complète ainsi qu'une spécification DEVS détaillée d'un système multi-agent. La spécification DSDEMAS, pour *Dynamic Structure Discrete event Multiagent Systems* (Duboz et al., 2005) est une formalisation du paradigme agent basée sur le formalisme DEVS et l'extension DSDEVS (Barros, 1995). Quesnel (2006) complète ensuite ces travaux de spécification, en se basant sur la variante parallèle des formalismes (*i.e.* PDEVS et DSDE).

Dans cette formalisation, un SMA est représenté par un modèle couplé DSDEVS, dont les composants forment l'ensemble des agents et des environnements. Les changements de structures sont demandés par les agents du SMA au modèle exécutif à travers son interface. Le SMA a la possibilité de changer sa propre structure de manière autonome à travers la définition de la fonction de transition interne du modèle exécutif. Dans ce cas, les auteurs considèrent le SMA comme un agent.

Un agent est également représenté par un modèle DSDEVS. Il est formé par l'ensemble des modèles qui le compose. Une analogie est faite entre la *perception* des agents et l'arrivée d'évènements externes sur les ports d'entrées des modèles, impliquant un changement d'état via les fonctions de transition externe des modèles. En ce qui concerne l'*action*, une analogie est faite avec les fonctions de sorties. L'ensemble des modèles dénués de ports d'entrée et initiant des sorties forment la *proactivité* de l'agent. Le comportement *réflexe* est formé par l'ensemble des modèles initiant une sortie juste après la perception, à travers un état *transitoire*. L'*autonomie* de l'agent est formée par l'ensemble des modèles non couplés aux ports d'entrée ou de sortie du modèle agent. L'union de la perception, proaction, action et autonomie forme le comportement de l'agent. Le lecteur intéressé peut se référer à Duboz et al. (2005) pour la formalisation du comportement de l'agent.

En ce qui concerne l'environnement, les auteurs laissent au modélisateur la possibilité d'utiliser un environnement *centralisé*, *distribué* ou vu comme un *agent*, conformément aux travaux de Soulié (2001). L'environnement centralisé est défini par un modèle atomique DEVS. Ce dernier est formé par l'ensemble des informations concernant la nature de l'environnement et des ressources du système. Le modèle est passif et attend les demandes de perceptions effectuées par les agents. Dès qu'une demande se présente, l'environnement répond de manière instantanée ($ta(s) = 0$) à l'agent. Si il y a nécessité de distribuer l'environnement, les auteurs proposent d'utiliser un environnement cellulaire à travers Cell-DEVS (Wainer et Giambiasi, 2001), une extension DEVS permettant de simplifier la définition d'automates cellulaires. Chaque cellule est alors définie comme un environnement centralisé. Enfin, le modélisateur peut également utiliser la définition de l'agent pour définir un environnement.

Cette spécification présente de nombreux avantages, notamment la possibilité de définir

des agents de manière modulaire, ce qui permet d'envisager des architectures d'agents complexes. Néanmoins, nous pensons que formaliser systématiquement l'agent en tant que réseau couplé dynamique peut être disproportionné pour certains SMAs, par exemple pour la modélisation d'un agent réactif dont l'architecture la plus adaptée serait celle de subsomption (Brooks, 1983) (*cf.* section I.3.1). Par rapport à l'approche de Uhrmacher (2001), DSDEMAS dispose d'un autre avantage : celui de supporter une approche multi-environnement. En revanche, seuls les modèles génériques sont formalisés et doivent être adaptés à chaque application. Ainsi, la modélisation n'est pas facilitée et reste dépendante de chaque application.

4.2.4 *M-DEVS*

Plus récemment, J.P. Müller, l'un des auteurs de l'approche Influence/Réaction (Ferber et Müller, 1996), pose les bases d'une approche dérivée du formalisme PDEVS, nommée M-DEVS (Müller, 2009), qui a été mise en œuvre dans la plateforme de simulation MIMOSA (Müller, 2004). Le but de ces travaux est de définir une sémantique non-ambigüe pour la simulation de SMAs à évènements discrets satisfaisant un certain nombre de propriétés :

1. le comportement réactif et proactif de l'agent ;
2. la dynamique structurelle, afin de pouvoir modifier la structure des interactions et de créer/supprimer des entités au cours de la simulation ;
3. la concurrence, afin qu'une entité puisse traiter l'ensemble des évènements simultanés et à laquelle PDEVS apporte une solution ;
4. l'instantanéité, qui permet de traiter séparément les influences physiques (tentatives d'action des agents) des influences logiques (perception, message informationnel), qui sont instantanées.

Le comportement réactif/proactif ainsi que la concurrence sont deux propriétés déjà supportées par le formalisme PDEVS. En revanche, PDEVS ne permet pas d'obtenir une dynamique structurelle, à moins de définir une extension ou d'en utiliser une existante, telle que DSDE ou ρ-DEVS. Enfin, l'instantanéité des évènements, bien que déjà supportée par le formalisme PDEVS, est gérée par un autre mécanisme dans M-DEVS. L'auteur propose donc d'altérer la structure d'un modèle atomique afin de permettre une dynamique structurelle tout en modifiant le principe d'instantanéité. Nous présentons cette nouvelle structure proposée par Müller (2009) :

$$M-DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \delta_{log}, \lambda_{ext}, \lambda_{log}, \lambda_{str}, ta)$$

où les ensembles X, Y, S et les fonctions δ_{int}, δ_{ext}, δ_{con} sont les mêmes que pour un modèle atomique PDEVS. La fonction λ est rebaptisée λ_{ext}. Afin de traiter la diffusion d'information instantanée au sein du système, généralement modélisée à l'aide d'un $ta(s) = 0$, l'auteur propose de restreindre la définition de la fonction ta à l'ensemble des réels positifs ($ta : S \rightarrow \mathbb{R}^+ - \{0\}$) et d'introduire à la place deux nouvelles fonctions (λ_{log} et δ_{log}) qui permettent respectivement de diffuser et de traiter de manière spécifique cette diffusion d'information

instantanée, que l'auteur appelle une *influence logique* :

$$\delta_{log} : Q \to S \qquad\qquad \lambda_{log} : Q \to Y^b$$

Pour répondre à la problématique de la dynamique structurelle, l'auteur s'inspire à la fois de DSDE et de ρ-DEVS et ajoute la notion d'*influences structurelles* permettant de modifier la structure du système de manière réflexive, propagées via la fonction de sortie λ_{str} :

$$\lambda_{str} : S \to Y^b$$

La fonction ta, dont le codomaine est déjà restreint à $\mathbb{R}^+ - \{0\}$ est encore modifiée pour ajouter la notion d'influence interne. La fonction est non seulement responsable d'indiquer la date du prochain évènement mais aussi la nature de cet évènement interne à travers une influence :

$$ta : S \to (\mathbb{R}^+ - \{0\}) \times \Gamma$$

En conséquence, la fonction δ_{int} est également modifiée pour recevoir en paramètre une influence en plus de l'état du modèle :

$$\delta_{int} : S \times \Gamma \to S$$

Ainsi, ta et δ_{int} forment la proactivité de l'agent, λ_{ext} et δ_{ext} forment la réactivité de l'agent à travers la propagation et le traitement d'influences physiques, λ_{log} et δ_{log} sont responsables de la diffusion d'information instantanée et λ_{str} permet d'initier les changements structurels du système.

Si la définition d'une unique extension permettant d'englober l'ensemble des propriétés nécessaires à la spécification d'un SMA semble attrayante, M-DEVS ne rassemble pas toutes les qualités nécessaires. L'une des principales raison est l'absence de preuve de fermeture sous composition, pourtant essentielle afin de montrer l'équivalence entre un modèle atomique et un modèle couplé. Cette propriété permet de garantir que le couplage n'a aucune influence sur les trajectoires d'états d'un modèle. Elle peut être démontrée en associant la structure du réseau de modèles à l'état d'un modèle atomique et en définissant la sémantique du réseau à l'aide des fonctions du modèle atomique. Au sein de la communauté de la TMS, l'absence de cette propriété ne permet pas de considérer un formalisme comme une extension de DEVS.

Nous pensons également que le rôle d'une telle approche est de simplifier la définition de modèles pour un paradigme donné, comme c'est le cas de Cell-DEVS (Wainer et Giambiasi, 2001) pour les automates cellulaires. Or, les modifications apportées par M-DEVS à la structure d'un modèle atomique complexifient la définition d'un agent (ajout de trois nouvelles fonctions et modification de la sémantique de la fonction ta). De plus, une partie de ces additions aurait pu être évitée. En effet, le choix de traiter séparément les évènements instantanés des évènements différés nous semble discutable ; associer systématiquement le premier à une diffusion d'information entre deux modèles est beaucoup trop restrictif.

En effet, Mosterman et Biswas (2000) identifient deux types d'abstraction de phénomènes continus qui peuvent motiver l'utilisation de l'instantanéité lors de la modélisation d'un système à évènements discrets : l'abstraction *de paramètres* et l'abstraction d'*échelle de temps*. La première abstraction permet de simplifier le système modélisé en se passant des variables pouvant affecter les comportements rapides d'un système, et dont les valeurs d'intérêt (seuil critique, saturation) peuvent être remplacées par un changement d'état. La seconde permet de condenser un comportement qui se produit à une plus petite échelle de temps par rapport à l'échelle d'intérêt en un seul point dans le temps. L'utilisation de ces abstractions sont le signe que le modèle n'est pas suffisamment détaillé pour refléter le véritable phénomène physique. Cependant, elles peuvent être parfaitement acceptables si elles sont en accord avec les objectifs de modélisation, même si elles peuvent conduire à l'occurrence d'évènements simultanés, d'autant plus que le formalisme PDEVS permet de gérer ces collisions d'évènements.

Le mécanisme mis en place par M-DEVS qui permet de traiter de manière spécifique les influences logiques (diffusion d'information instantanée) reflète plutôt une abstraction de paramètres. Restreindre la fonction d'avancement du temps pour empêcher la propagation d'un évènement instantané ($ta(s) = 0$) force l'utilisation du mécanisme d'influence logique pour des abstractions d'échelle de temps. Ceci rend beaucoup moins explicite la notion de contraction du temps, puisque l'évènement n'est pas déclenché par la fonction d'avancement ta. Considérons par exemple un système dont la dynamique est saisonnière. Si de manière ponctuelle, il peut être nécessaire de représenter une dynamique plus fine (*e.g.* journalière), il est tout à fait acceptable de rester sur la première échelle de temps [3], et d'utiliser un état transitoire ($ta(s) = 0$) pour représenter une dynamique journalière. L'évènement n'est pas intemporel, mais est bien un évènement physique dont la durée a été abstraite.

4.2.5 *Large-scale, dynamic, extensible and flexible (LDEF) formalism*

À notre connaissance, les derniers travaux de formalisation des SMAs qui s'appuient sur le formalisme DEVS sont proposés par Bae et Moon (2016). Les auteurs proposent le formalisme *Large-scale, dynamic, extensible and flexible* (LDEF) dans le but de favoriser la réutilisabilité de modèles à travers une définition modulaire et hiérarchique. Les objectifs donnés par les auteurs sont les suivants :

- *large-scale* : permettre le développement de modèles intégrant un grand nombre de composants ;
- *dynamic* : permettre les changements de structure au cours de la simulation ;
- *extensible* : permettre la construction d'un modèle en ajoutant un autre modèle à un modèle existant (modélisation incrémentale) ;
- *flexible* : permettre la construction d'un modèle en remplaçant un composant d'un modèle existant.

Contrairement aux autres travaux que nous avons présentés dans cette section, les auteurs de LDEF ont choisi une approche différente. Plutôt que de spécifier l'ensemble

3. À condition que les objectifs de simulation permettent une telle abstraction.

des éléments du SMA à l'aide du formalisme DEVS, ils proposent leurs propres structures formelles (dont certaines sont directement reprises du formalisme DEVS classique). Ils montrent dans un second temps que LDEF est homomorphe[4] à DEVS.

Fondamentalement, les auteurs proposent des entités permettant de spécifier les aspects comportementaux et structurels d'un SMA. Les éléments structurels peuvent être des agents ou des environnements. Pour chacun d'eux, un élément comportemental est défini. L'élément comportemental dédié aux agents est appelé modèle d'action, et est basé sur une version modifiée d'un modèle atomique : S est formé par des sous-ensembles, lesquels sont utilisés pour restreindre les fonctions δ et λ (qui sont par ailleurs renommées pour utiliser le vocabulaire propre aux SMAs). Le modèle d'action définit comment l'agent perçoit, décide et agit. L'élément comportemental associé à l'environnement est un atomique, il est destiné à traiter les sollicitations externes. Pour les éléments structuraux, le modèle structurel dynamique est défini. Il joue le rôle de modèle couplé, où les changements structuraux sont centralisés à la manière de DSDE.

4.3 *Discussion*

Cette section a été l'occasion d'étudier les différents formalismes utilisés par la communauté SMA et ceux qui ont été conçus pour la spécification de modèles SMAs. Si un nombre important de formalismes sont utilisés, peu d'entre eux permettent d'envisager une spécification complète des comportements de chaque entité du SMA et de décrire la façon dont ils sont structurés. Les travaux basés sur le formalisme DEVS permettent en revanche d'envisager une spécification complète. Bien que leur adoption au sein de la communauté SMA reste marginale, « *il s'avère que la communauté TMS propose un cadre général qui peut bénéficier à la communauté ABS*[5] » (Duboz et al., 2012).

Les travaux de Uhrmacher et Schattenberg (1998) ont permis de faire le pont entre le vocabulaire propre aux SMAs et celui de la TMS, les concepts entre l'agent et un modèle étant analogues. Ces derniers ont également permis de gérer une dynamique structurelle au cours de la simulation, ce qui constitue un avantage certain par rapport aux formalismes existants à l'époque, tel que le formalisme BRIC. En revanche, ils ont négligé les aspects d'environnements et d'organisation, bien que des travaux ultérieurs ont partiellement traité ces aspects (Uhrmacher et Kuttler, 2006).

La spécification DSDEMAS (Duboz et al., 2005) est plus complète. Ces travaux, que l'on peut considérer comme la suite logique de Uhrmacher et Schattenberg (1998) permettent de représenter des systèmes multi-agents réactifs et cognitifs. L'agent, l'environnement et le système multi-agent sont formalisés sous forme de modèles DEVS. Les environnements peuvent être physiques ou sociaux, être distribués, et peuvent éventuellement avoir une dynamique endogène. En revanche, l'approche s'apparente plutôt à une méthode, guidant à

4. Un homomorphisme permet de mettre en évidence les similitudes entre deux systèmes. Si les entrées, sorties et les états d'un système peuvent être associés à ceux d'un autre système, alors il y a une relation homomorphique entre les deux systèmes.

5. *Agent-Based Simulation*.

travers une spécification générique le modélisateur dans sa conception d'un modèle SMA. Seuls les modèles génériques sont formalisés, les modèles ne sont pas réutilisables. Les auteurs de la spécification ne semblent pas avoir de position arrêtée quant à l'utilisation du formalisme DEVS ou PDEVS. En effet, si l'article base la spécification sur DEVS classique, les travaux de thèse de Quesnel (2006) sont basés sur PDEVS.

L'approche de GALATEA est intéressante puisqu'un langage dédié permet de faciliter la modélisation. La coordination des actions est réalisée à l'aide d'une variante du modèle Influence/Réaction. Néanmoins, les deux phases Influence/Réaction sont combinées en une seule, l'approche étant basée sur DEVS classique. La manière dont les modèles DEVS sont construits à travers le langage n'est pas communiquée. Ceci ne facilite pas une implémentation de l'approche sur d'autres plateformes. Les travaux de Müller (2009) se démarquent en proposant un formalisme basé sur PDEVS qui permet de décrire des SMA sous la forme d'une extension du formalisme, mais n'apporte en revanche pas la preuve de fermeture sous composition. Trois nouvelles fonctions sont ajoutées à la définition d'un modèle atomique et la sémantique de la fonction *ta* est modifiée. Enfin, le formalisme LDEF constitue une approche originale en définissant son propre formalisme, avant de montrer que celui-ci est homomorphe à DEVS.

Nous synthétisons dans le tableau 2.3 les aspects abordés par les différentes spécifications tout en rappelant les formalismes utilisés. Parmi les critères indiqués, nous avons partiellement fait apparaître les aspects auxquels nous attachons une importance (*cf.* section I.5).

Littérature	Formalismes	Corps	Env.	Org.	Int.	Dyn.
Uhrmacher et Schattenberg (1998)	DEVS, dynDEVS	✗	✗	✗	✓	✓
Dávila et Uzcátegui (2000)	DEVS	✗	✓	✗	✓	✗
Duboz et al. (2005)	(P)DEVS, DSDEVS/DSDE	✗	✓	✓	✓	✓
Müller (2009)	M-DEVS	✗	✓	✓	✓	✓
Bae et Moon (2016)	LDEF	✗	✓	✓	✓	✓

Tableau 2.3 Synthèse des aspects abordés (corps, environnement, organisation, interaction, dynamique structurelle) des différents travaux basés sur le formalisme DEVS pour représenter des SMAs.

Trois des cinq approches sont basées sur le formalisme DEVS classique, qui ne permet pas de gérer la simultanéité des actions de la même façon que le principe Influence/Réaction. La position des auteurs de Duboz et al. (2005) en ce qui concerne la question de la simultanéité reste ambiguë puisque, sur deux contributions différentes, les auteurs utilisent DEVS et PDEVS de façon interchangeable. Seul M-DEVS est explicitement basé sur PDEVS et aborde les problématiques de simultanéité des actions en utilisant le mécanisme Influence/Réaction. Cependant, comme nous l'avons vu précédemment nous sommes en désaccord avec la restriction appliquée à la fonction d'avancement du temps.

Au delà des problématiques d'ordonnancement, les différentes approches ne réunissent pas les propriétés qui nous semblent importantes (*cf.* section I.5). Par exemple, la distinction

entre le corps et le système cognitif de l'agent est un concept qui n'est appliqué dans aucune des approches. Le corps représente néanmoins un moyen de permettre la conception de SMAs multi-environnements, et de permettre à une dynamique environnementale d'agir sur des propriétés physiques liées à l'agent.

De notre point de vue, l'approche DSDEMAS est la plus complète et la plus fidèle aux concepts de la TMS. Les travaux que nous développons dans le chapitre suivant sont à placer dans cette continuité.

5 *Résumé du chapitre*

Nous avons présenté dans ce chapitre les outils liés au domaine de la théorie de la modélisation et de la simulation. Nous nous sommes particulièrement intéressés à l'étude des différents formalismes permettant de spécifier les modèles, afin de poursuivre notre objectif visant à proposer une spécification précise du paradigme agent. Dans la recherche d'un outil permettant de représenter ce paradigme, notre choix s'est porté sur le formalisme PDEVS. Celui-ci, en dehors de permettre la description de la structure et du comportement d'un modèle de façon non-ambiguë, dispose de propriétés adaptées à la description de systèmes multi-agents.

Dans la dernière partie du chapitre, nous avons réalisé un état de l'art des travaux formels concernant le paradigme agent. Ceci nous a permis de mettre en avant le fait que la plupart des formalismes proposés s'attachent à décrire un aspect particulier du système (*e.g.* comportement de l'agent, interactions). D'autres travaux plus génériques, dont une partie est basée sur le formalisme DEVS, permettent de décrire la structure d'un SMA. Bien qu'intéressants, ces derniers n'ont pas selon nous, toutes les caractéristiques pour permettre la spécification de SMAs basés sur notre positionnement vis-à-vis du paradigme agent (*cf.* section I.5).

Pour ces raisons, nous développons dans le prochain chapitre une spécification générique des SMAs basée sur le formalisme PDEVS, baptisée DPDEMAS (*Dynamic Parallel Discrete Event Multi-Agent Specification*). Nous associons à celle-ci une méthode permettant de guider le modélisateur dans l'utilisation de cette spécification afin de décrire un modèle, et ainsi faciliter sa traduction dans des outils de simulation.

Chapitre III

SPÉCIFICATION FORMELLE DES SMA

Sommaire

1 *Introduction*

Les deux précédents chapitres ont permis de présenter les enjeux liés à la conception d'un SMA pour la modélisation et la simulation. Si l'une de nos principales préoccupations concerne la reproductibilité des expériences numériques basées sur les SMA, nous ne prétendons pas dans ce chapitre y apporter une solution complète. Pour traiter pleinement cette problématique, une méthodologie associée à des outils de V&V est nécessaire. Le travail que nous proposons ici permet néanmoins de traiter une partie de la problématique, qui doit être considérée comme une étape pouvant être intégrée à une méthodologie complète. Cette étape est celle de la spécification formelle du système multi-agents, dont nous avons évoqué l'intérêt dans le chapitre II et qui constitue un pas vers la reproductibilité. Il est tout à fait envisageable d'utiliser cette spécification au sein d'une méthodologie future ou d'une méthodologie existante, telle que celle proposée par Duboz et al. (2012) (*cf.* section I.4.3.1).

Dans la continuité des travaux de Duboz et al. (2005) que nous avons présentés dans le

chapitre II, nous développons dans ce chapitre une spécification générique du paradigme agent à l'aide des outils formels proposés par la théorie de la M&S (Zeigler et al., 2000). La complexité des SMA, du fait du nombre important de sous-systèmes qui les composent, de la diversité des domaines d'applications et des points de vue conceptuels, font qu'il est contraignant de les exprimer à l'aide d'un seul formalisme. La position centrale du formalisme PDEVS a été montrée (Vangheluwe, 2000) lorsqu'il s'agit d'effectuer de la multi-modélisation. Cette approche permet de préserver l'hétérogénéité des différents composants qui entrent en jeu au sein d'un système. Pour les systèmes multi-agents, nous allons voir que cette approche permet de s'adapter au mieux aux spécificités de chaque modèle SMA.

Pour définir une spécification des SMA, nous suivons le positionnement que nous avons établi section I.5 vis-à-vis des concepts du paradigme agent, et des responsabilités que nous accordons à chaque entité d'un SMA. Ce chapitre est organisé comme suit. Une première section est consacrée à la définition de notre spécification formelle des SMA, que nous avons baptisée *Dynamic Parallel Discrete Event Multi Agent Specification* (DPDEMAS). Une seconde section est dédiée à la définition de différents environnements à l'aide d'extensions PDEVS adaptées à la topologie visée. Une troisième section permet de présenter une vue d'ensemble de l'approche permettant d'avoir une vision synthétique de nos propositions. Nous y présentons également des méthodes de spécification associées à notre spécification afin de guider le modélisateur dans la conception d'un modèle.

2 *Spécification DPDEMAS*

Dans cette section, nous présentons notre approche générique *Dynamic Parallel Discrete Event Multi-Agent System* (DPDEMAS), une spécification SMA basée sur le formalisme PDEVS que nous avons en partie présentée dans Franceschini et al. (2017). Nous y formalisons les différentes entités génériques qui composent un SMA afin d'en faciliter la modélisation. Celles-ci prennent soit la forme d'un modèle du formalisme PDEVS, soit de structures intégrées au sein de l'état des modèles.

Cette approche permet dans un premier temps de guider le modélisateur dans la conception et la spécification d'un modèle SMA. Puisqu'elle est basée sur PDEVS, il est possible de tirer partie de la modularité offerte par le formalisme, qui permet de coupler des modèles hétérogènes de manière modulaire. Cette approche nous permet de développer des modèles réutilisables, où chacun d'eux est plus ou moins adapté à une problématique. Dans cette optique, nous proposons des modèles d'environnements pouvant être réutilisés par le modélisateur en fonction de ses besoins. Par exemple, une base de modèles d'environnements adaptés à différentes topologies peut être proposée. Le modélisateur peut alors concevoir son SMA à partir de cet ensemble de modèles préétablis. Cette approche permet non seulement de réduire le temps de conception des modèles, mais aussi d'enrichir la base de modèles réutilisables.

Cette section est organisée comme suit. Nous présentons dans un premier temps notre spécification du système multi-agents et de ses sous-systèmes : l'agent, l'environnement.

Nous précisons ensuite les interactions entre les différents composants de notre spécification.

2.1 *Le système multi-agents*

Afin de permettre au système multi-agents d'adapter sa structure de manière dynamique, nous formalisons le SMA à l'aide d'un modèle couplé DSDE. Celui-ci permet de centraliser les modifications structurelles au sein d'un modèle atomique spécifique, appelé modèle exécutif, qui peut agir sur la topologie du réseau de modèles (*cf.* section II.3.4). Cette particularité nous permet de refléter l'évolution des interactions entre les différentes entités du SMA, et de permettre l'apparition ou la disparition des agents.

Conformément à la structure d'un modèle couplé DSDE, un système multi-agents DPDEMAS est décrit par :

$$DPDEMAS = (X_{MAS}, Y_{MAS}, \chi, M_\chi)$$

où le modèle exécutif est représenté par M_χ, un PDEVS atomique modifié défini par :

$$M_\chi = (X_\chi, Y_\chi, S_\chi, \delta_{int_\chi}, \delta_{ext_\chi}, \delta_{con_\chi}, \lambda_\chi, \mathrm{ta}_\chi, \zeta).$$

Le modèle exécutif a la particularité de pouvoir agir sur la topologie du réseau grâce à la fonction ζ qui permet d'accéder à la structure du réseau de composants associée à un état $s_\chi \in S_\chi$:

$$\Sigma = \zeta(s_\chi) = (D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$$

où D est l'ensemble des identifiants des modèles ; $\{M_d\}$ l'ensemble des modèles ; $\{I_d\}$ et $\{Z_{i,d}\}$ les informations de couplage.

Rappelons que notre positionnement vis-à-vis du paradigme agent est de considérer l'agent comme une entité dont le corps est distinct de l'esprit. Ainsi, dans notre spécification, le système multi-agents représenté par un modèle $DPDEMAS$ est composé de modèles d'environnements, eux-mêmes composés par le corps des agents, et des modèles correspondant aux esprits des agents. Afin de différencier les environnements des agents, nous notons $D_{Ag} \subset D$ l'ensemble des identifiants des modèles agents et $D_E \subset D$ l'ensemble des identifiants des modèles d'environnements. L'ensemble des identifiants est formé par $D = D_{Ag} \cup D_E$. De la même manière, nous scindons l'ensemble des modèles $\{M_d\}$ afin de distinguer les modèles d'environnements et les modèles correspondant aux esprits des agents :

$$\{M_d\} = \{E_d \mid d \in D_E\} \cup \{Ag_d \mid d \in D_{Ag}\}$$

La figure 3.1 permet de donner une vue d'ensemble d'un SMA spécifié à l'aide de DPDEMAS. Nous reviendrons dans les sections suivantes sur les couplages entre les sous-systèmes.

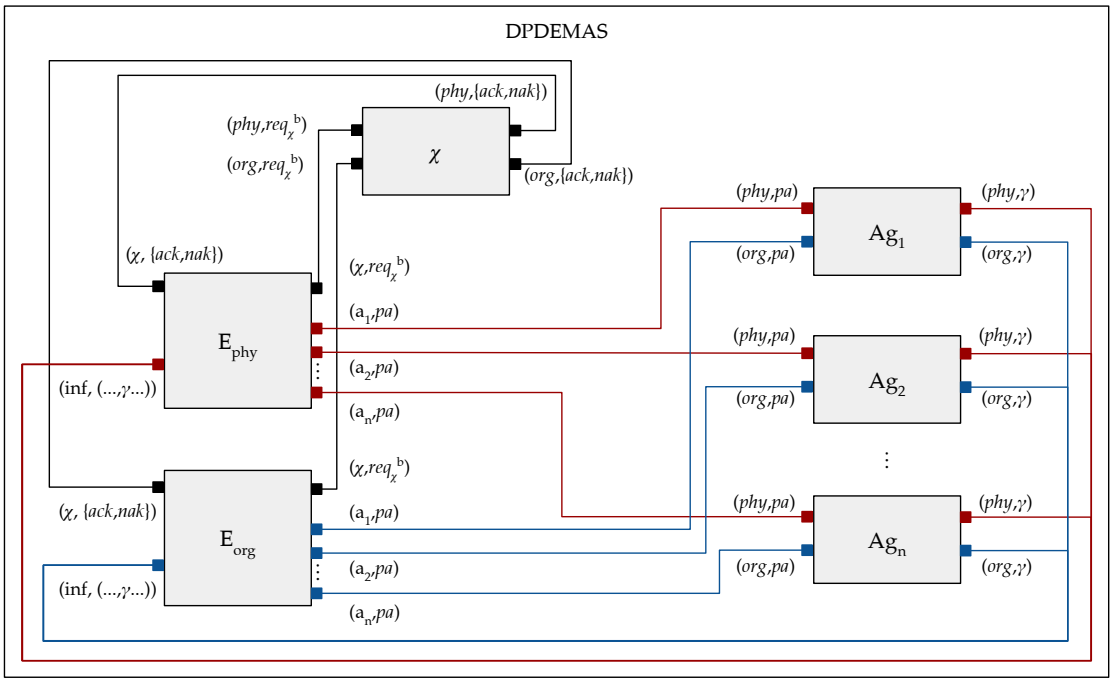


Figure 3.1 Vue d'ensemble d'un système multi-agents DPDEMAS.

2.1.1 *Interface*

Bien que le modèle couplé DPDEMAS permette à lui seul de représenter le paradigme agent, nous donnons la possibilité à un SMA d'être vu comme un agent au sein d'un SMA de niveau hiérarchique supérieur. Un DPDEMAS peut disposer d'une interface d'entrées/-sorties, qui peut être motivée par les raisons suivantes :

- la réception de paramètres d'entrée utiles à l'initialisation du système multi-agents ;
- la représentation des perturbations provenant de modèles exogènes au SMA et destinées aux agents ou aux environnements ;
- l'envoi en sortie de valeurs agrégées, comme des groupes de populations (*e.g.* population saine et population infectée dans le cadre d'un modèle épidémiologique).

Leur définition étant dépendante de l'application, nous laissons la spécification de ces entrées/sorties à la discrétion du modélisateur. Si le SMA est considéré comme un agent dans un SMA de niveau supérieur, le modèle DPDEMAS doit respecter l'interface que nous spécifions pour un agent dans la section suivante.

2.1.2 *Modèle exécutif*

Le modèle couplé SMA, à travers son modèle exécutif χ, est responsable de la structure du réseau. Celui-ci centralise les demandes de changements structurels. Le modèle exécutif du modèle DPDEMAS a un comportement passif : il opère des modifications uniquement lorsqu'il reçoit un évènement externe. Comme la figure 3.1 le montre, le modèle exécutif est destiné à être couplé avec les modèles d'environnements. C'est de ces derniers qu'émanent les

demandes de modifications structurelles lorsque un changement d'état de l'environnement implique également un changement de la topologie. Par exemple, la demande de suppression d'un agent implique la suppression de l'ensemble des couplages liés aux entrées/sorties de l'esprit de l'agent, des ports qui lui sont éventuellement dédiés dans d'autres modèles, et du modèle correspondant au système cognitif de l'agent. En ce qui concerne le corps de l'agent, comme nous le verrons dans la suite de ce chapitre, c'est l'environnement qui a en charge de supprimer le n-uplet correspondant.

Interface

Le modèle exécutif d'un modèle DPDEMAS dispose d'un port d'entrée pour chaque environnement du SMA. Le nom du port est l'identifiant de l'environnement associé, et la valeur du message est un sac de requêtes de changement structurels :

$$X_\chi = \{(p, v) \mid p \in D_E, v \in Req^b\}$$

où Req est l'ensemble des requêtes, dont chaque élément est formé par un couple composé du nom de l'opération à effectuer et de paramètres associés, qui est défini par :

$$Req = \{(req, params)\}$$

avec

$$req \in \{add_{ag}, att_{ag}, del_{ag}, add_{env}, del_{env}, add_{coupl}, del_{coupl}\}$$

et

$$params \in \begin{cases} \{(a, Ag_a) \mid a \in D_{Ag}, Ag_a \in \{M_d\}\} & \text{si } req = add_{ag} \\ \{(a, e) \mid a \in D_{Ag}, e \in D_E\} & \text{si } req = att_{ag} \\ \{(a) \mid a \in D_{Ag}\} & \text{si } req = del_{ag} \\ \{(e, E_e) \mid e \in D_E, E_e \in \{M_d\}\} & \text{si } req = add_{env} \\ \{(e) \mid e \in D_E\} & \text{si } req = del_{env} \\ \{(a_1, a_2) \mid a_1, a_2 \in D_{Ag}, a_2 \in OPorts_{a_1}, a_1 \in IPorts_{a_2}\} & \text{si } req \in \{add_{coupl}, del_{coupl}\} \end{cases}$$

Respectivement, les opérations qui peuvent être demandées à l'exécutif représentent :

- la création d'un nouvel agent au sein du SMA et son rattachement à l'environnement qui effectue la requête ;
- le rattachement d'un agent déjà existant à un environnement spécifique ;
- la suppression d'un agent de l'environnement qui effectue la requête ;
- la création d'un nouvel environnement ;
- la suppression d'un environnement ;
- la création d'un couplage entre deux agents pour permettre une communication directe ;
- la suppression d'un couplage de communication directe entre deux agents.

Le comportement associé à chacune de ces demandes est donné dans la suite de cette section. En ce qui concerne les sorties du modèle exécutif, celles-ci sont définies par :

$$Y_\chi = \{(p, v) \mid p \in D_E, v \in \{ack, nak\}\}.$$

Il existe un port de sortie pour chaque environnement, sur lequel une confirmation (*ack*) est envoyée lorsque les changements ont bien été effectués. Si l'état du réseau de composants du SMA ne permet pas de répondre aux demandes, une erreur (*nak*) est envoyée.

Comportement

Etant donné que l'exécutif est un modèle passif, le comportement dans son ensemble est relativement simple. Toute la complexité réside dans le traitement des demandes, qui est effectué par la transition externe. L'état du modèle exécutif est définit par :

$$S = \{(phase, \eta)\}$$

où $phase \in \{send, wait\}$ est le nom du macro-état de l'exécutif ; et $\eta = \{(env, rep) \mid env \in OPorts_\chi, rep \in \{ack, nak\}\}$.

L'état initial est $s_0 = (wait, \emptyset)$. Le modèle exécutif attend donc indéfiniment une requête de la part d'un environnement. Lorsque l'exécutif reçoit une requête, il la traite puis passe dans un état transitoire ($phase = send$) afin d'envoyer une réponse aux environnements qui ont effectué une demande de changement structurel. La fonction d'avancement du temps est définie par :

$$ta((phase, \eta)) = \begin{cases} 0 & \text{si } phase = send \\ \infty & \text{si } phase = wait \end{cases}$$

La fonction de sortie envoie la variable η qui contient l'ensemble des réponses aux environnements :

$$\lambda((phase, \eta)) = \eta$$

Le modèle exécutif passe ensuite dans la phase *wait* et attend les prochaines demandes de changements structurels.

$$\delta_{int}((send, \eta)) = (wait, \emptyset)$$

La définition de la fonction de transition externe étant moins concise, nous l'exprimons sous forme de pseudo-code à travers le listing 3.1. Le traitement des requêtes simultanées contenues dans le sac d'entrées est effectué séquentiellement par l'exécutif selon le type de la requête. Ainsi les lignes 5 à 16 du listing 3.1 correspondent à l'ajout des éventuels nouveaux environnements. L'identifiant et le modèle du nouvel environnement sont ajoutés au modèle couplé DPDEMAS, et les couplages entre l'environnement et le modèle exécutif sont réalisés. Les lignes 17 à 28 correspondent à l'ajout d'un nouvel agent dans le système. L'identifiant et le modèle de l'esprit du nouvel agent sont ajoutés au modèle couplé, et les couplages entre l'esprit de l'agent et l'environnement qui effectue la requête sont réalisés. Les lignes 29 à 41 correspondent au rattachement d'un agent déjà existant dans le système à un environnement spécifique, en créant les couplages nécessaires entre l'esprit de l'agent et cet environnement. Si ils ne font pas déjà parti du système, une réponse négative (*nak*) est préparée pour envoi à l'environnement qui effectue la requête. Les lignes 42 à 52 correspondent à l'ajout d'un couplage direct entre les esprits de deux agents, permettant aux deux agents de communiquer, et les lignes 53 à 63 correspondent à la suppression d'un couplage direct. Les lignes 64 à 81 correspondent à la suppression d'un agent de l'environnement qui effectue

la requête, qui se matérialise par la destruction des couplages qui lient les deux modèles. Si après suppression des couplages l'esprit de l'agent devient isolé, celui-ci est supprimé du SMA. Enfin, les lignes 82 à 98 correspondent à la gestion des demandes de suppression d'un environnement. Si l'environnement n'est lié à aucun agent, le modèle est supprimé une fois les couplages qui lient l'environnement et le modèle exécutif supprimés.

```
1  variables
2    Σ = (D, {M_d}, {I_d}, {Z_{i,d}}) // état du réseau de modèles
3  fonction δ_ext((wait, ∅), e, x^b) faire
4    η = ∅
5    // traitement des demandes d'ajout d'environnement
6    pour chaque (env, (add_env, (e, E_e))) ∈ x^b faire
7      // ajout de l'environnement
8      D = D ∪ {e}
9      {M_d} = {M_d} ∪ {E_e}
10     // création des couplages avec le modèle exécutif
11     I_χ = I_χ ∪ {e}
12     I_e = I_e ∪ {χ}
13     Z_{χ,e}(e) = χ
14     Z_{e,χ}(χ) = e
15     η = η ∪ (env, ack) si ∄(env, nak) ∈ η
16   fin pour
17   // traitement des demandes de création d'agents
18   pour chaque (env, (add_ag, (a, Ag_a))) ∈ x^b faire
19     // création de l'agent
20     D = D ∪ a
21     {M_d} = {M_d} ∪ {Ag_a}
22     // création des couplages entre le système cognitif et l'environnement
23     I_env = I_env ∪ {a}
24     I_a = I_a ∪ {env}
25     Z_{a,env}(env) = inf
26     Z_{env,a}(a) = env
27     η = η ∪ (env, ack) si ∄(env, nak) ∈ η
28   fin pour
29   // traitement des demandes de rattachement d'un agent à un environnement
30   pour chaque (env, (att_ag, (a, e))) ∈ x^b faire
31     si ∃a ∈ D_Ag ∧ ∃e ∈ D_E alors // vérification existence des modèles
32       // création des couplages entre le système cognitif et l'environnement
33       I_e = I_e ∪ {a}
34       I_a = I_a ∪ {e}
35       Z_{a,e}(env) = inf
36       Z_{e,a}(a) = e
37       η = η ∪ (env, ack) si ∄(env, nak) ∈ η
38     sinon // retour négatif à l'environnement
39       η = η \ (env, ack) ∪ (env, nak) si ∃(env, ack) ∈ η
40     fin si
```

41 fin pour
42 // traitement des demandes d'ajout d'une interaction directe entre deux agents
43 pour chaque $(env, (add_{coupl}, (a_1, a_2))) \in x^b$ faire
44 si $\exists a_1, a_2 \in D$ alors // vérification existence des modèles
45 // création du couplage direct entre les deux agents
46 $I_{a_2} = I_{a_2} \cup \{a_1\}$
47 $Z_{a_1,a_2}(p_{out}) = p_{in}$
48 $\eta = \eta \cup (env, ack)$ si $\nexists (env, nak) \in \eta$
49 sinon // retour négatif à l'environnement
50 $\eta = \eta \setminus (env, ack) \cup (env, nak)$ si $\exists (env, ack) \in \eta$
51 fin si
52 fin pour
53 // traitement des demandes de suppression d'une interaction directe entre deux agents
54 pour chaque $(env, (del_{coupl}, (a_1, a_2))) \in x^b$ faire
55 si $\exists a_1, a_2 \in D \wedge Z_{a_1}(a_2) = a_1$ alors // vérification existence du couplage
56 // suppression du couplage direct entre les deux agents
57 $Z_{a_1,a_2}(a_2) = \emptyset$
58 $I_{a_2} = I_{a_2} \setminus \{a_1\}$
59 $\eta = \eta \cup (env, ack)$ si $\nexists (env, nak) \in \eta$
60 sinon // retour négatif à l'environnement
61 $\eta = \eta \setminus (env, ack) \cup (env, nak)$ si $\exists (env, ack) \in \eta$
62 fin si
63 fin pour
64 // traitement des demandes de suppression d'un agent d'un environnement
65 pour chaque $(env, (del_{ag}, (a)) \in x^b$ faire
66 si $\exists a \in D$ alors
67 // suppression des couplages entre l'environnement et l'esprit cognitif
68 $I_a = I_a \setminus \{env\}$
69 $Z_{env,a}(a) = \emptyset$
70 $I_{env} = I_{env} \setminus \{a\}$
71 $Z_{a,env}(inf) = \emptyset$
72 // si l'agent n'est plus rattaché à aucun environnement, suppression de l'agent
73 si $I_a = \emptyset \wedge \forall d \in D \setminus \{env\}, a \notin I_d$ alors
74 $\{M_d\} = \{M_d\} \setminus \{Ag_a\}$
75 $D = D \setminus \{a\}$
76 fin si
77 $\eta = \eta \cup (env, ack)$ si $\nexists (env, nak) \in \eta$
78 sinon // retour négatif à l'environnement
79 $\eta = \eta \setminus (env, ack) \cup (env, nak)$ si $\exists (env, ack) \in \eta$
80 fin si
81 fin pour
82 // traitement des demandes de suppression d'un environnement

```
83    pour chaque (env, (del_env, (e)) ∈ x^b faire
84      // si l'environnement n'est plus rattaché à aucun agent
85      si ∃e ∈ D ∧ I_e \ {χ} = ∅ ∧ ∀d ∈ D \ {χ}, e ∉ I_d alors
86        // suppression des couplages entre l'exécutif et l'environnement
87        Z_{χ,e}(e) = ∅
88        Z_{e,χ}(χ) = ∅
89        I_χ = I_χ \ {e}
90        I_e = I_e \ {χ}
91        // suppression de l'environnement
92        {M_d} = {M_d} \ {E_e}
93        D = D \ {e}
94        η = η ∪ (env, ack) si ∄(env, nak) ∈ η
95      sinon // retour négatif à l'environnement
96        η = η \ (env, ack) ∪ (env, nak) si ∃(env, ack) ∈ η
97      fin si
98    fin pour
99    Σ' ← (D, {M_d}, {I_d}, {Z_{i,d}})
100   S' ← (send, η)
101 fin
```

Listing 3.1 Traitement des requêtes de changement dynamique à travers la fonction de transition externe du modèle exécutif.

2.2 *Les agents*

Afin de spécifier l'agent, nous nous appuyons sur l'analogie proposée par Uhrmacher et Arnold (1994), reprise par Duboz et al. (2002) entre un modèle PDEVS et un agent, laquelle permet de mettre en évidence la capacité d'un modèle PDEVS (couplé ou non) à satisfaire les propriétés définissant un agent selon Wooldridge et Jennings (1995) (*cf.* section I.5).

Plus précisément, nous reprenons cette analogie pour définir le *système cognitif* de l'agent, étant donné que nous utilisons la séparation explicite du « corps » et de « l'esprit » de l'agent. Le fait de décomposer architecturalement l'agent en deux parties pour distinguer l'esprit et le corps n'est pas une approche courante dans la communauté SMA (Saunier, 2015). Cette approche peut être associée à la théorie de l'approche incarnée de la cognition (Wilson, 2002). Cell-ci considère que l'esprit, le corps et l'environnement jouent un rôle dans le processus cognitif global, par opposition à l'approche cognitiviste classique (Haugeland, 1989) où le corps est entièrement déconnecté du siège de l'intelligence. Saunier (2015) considère le corps comme une abstraction de premier ordre : bien que l'esprit puisse prendre n'importe quelle décision, les limites à la réalisation de ces décisions sont imposées à la fois par les capacités du corps et par les règles de l'environnement.

Cette décomposition de l'agent présente plusieurs intérêts conceptuels. C'est d'abord pour donner la possibilité à un agent d'exister au sein de plusieurs environnements que le concept de corps trouve naturellement sa place, comme métaphore physique. Le corps est

alors la manifestation physique de l'agent au sein d'un environnement. C'est à travers lui que l'agent peut exposer certaines caractéristiques observables par les autres agents, qu'il peut percevoir des informations et agir. Les travaux de Soulié (2001) sur l'approche multi-environnement sont à notre connaissance les premiers à séparer de manière explicite le corps de l'esprit de l'agent. On retrouve ensuite cette idée dans les modèles conceptuels AGRE (Ferber et al., 2005) ou encore MASQ (Dinu et al., 2012). Plus qu'un simple intermédiaire entre l'environnement et le processus décisionnel de l'agent, le corps permet également de satisfaire la *contrainte d'intégrité interne de l'agent* (Michel, 2007a), qui consiste à s'assurer que les agents réalisent leur processus de délibération en disposant du contrôle de leur état. Bien qu'un modèle PDEVS soit capable d'évoluer de manière autonome, séparer le corps et l'esprit de l'agent permet de rendre explicite les propriétés de l'agent considérées comme perceptibles et éventuellement modifiables par d'autres agents, et de les décorréler du processus décisionnel. Ainsi, cette distinction permet de considérer la partie physique de l'agent comme une entité à part entière de l'environnement. Un agent ne perçoit pas le système cognitif de ses pairs, il perçoit uniquement leurs corps.

Nous donnons dans un premier temps la formalisation générique du système cognitif de l'agent. Nous donnons ensuite la formalisation du corps de l'agent.

2.2.1 *Système cognitif*

Le système cognitif a la responsabilité de collecter les perceptions, d'en tenir compte dans son processus de raisonnement avant de concrétiser ses décisions à travers l'action. Il représente en somme, le processus de délibération à trois phases identifié par Pnueli (1986). Un modèle PDEVS permet déjà de réaliser ce processus de délibération, nous représentons donc le système cognitif de l'agent par un modèle PDEVS.

Le comportement à donner au système cognitif d'un agent est de la responsabilité du modélisateur. Nous ne donnons aucune directive à propos de l'utilisation d'un modèle atomique, couplé, ou encore dynamique pour représenter le système cognitif. Nous pensons que cette décision revient au modélisateur en fonction du type d'agent qu'il souhaite modéliser et de sa complexité. Les travaux de Uhrmacher et Arnold (1994) puis de Duboz et al. (2005) montrent qu'un agent est analogue à une structure PDEVS, que l'on utilise un modèle atomique ou un couplé dynamique. Ainsi, un agent *réactif* sera par exemple mieux représenté par un PDEVS atomique, tandis qu'un agent *cognitif* capable de raisonnement et basé sur une architecture telle que *Beliefs-Desire-Intentions* (Rao et Georgeff, 1992) sera peut-être mieux représenté par un modèle couplé PDEVS, voire par un DSDE si l'agent est capable d'adapter sa structure pour modifier son comportement.

Interface du système cognitif

Afin qu'il puisse être interfacé avec les autres entités du SMA, un agent doit respecter une interface d'entrées/sorties spécifique que nous détaillons ci-dessous.

L'ensemble des ports-valeurs d'entrée d'un agent est décomposé en trois sous-ensembles :

$$X_{Ag} = X_{Ag_{env}} \cup X_{Ag_{agents}} \cup X_{Ag_{exo}}$$

où

- $X_{Ag_{env}} = \{(p, v) \mid p \in D_E, v \in Pa\}$ est l'ensemble des messages provenant des environnements dans lesquels l'agent est plongé. Chaque port est nommé d'après l'identifiant de l'environnement qui émet le message et la valeur du message représente un percept de l'agent. D_E est l'ensemble des identifiants des environnements et Pa est l'ensemble des percepts. Nous reviendrons plus tard sur la définition complète de l'ensemble des percepts (section 2.4.2).
- $X_{Ag_{agents}} = \{(p, v) \mid p \in D_{Ag}, v \in Msg\}$ représente les messages directs provenant d'autres agents, dans le cas où l'agent évolue dans un environnement social. Chaque port est nommé d'après l'identifiant de l'agent $\in D_{Ag}$ qui émet le message direct $\in Msg$. Nous reviendrons également sur cet aspect ultérieurement (section 2.4.3).
- Enfin, $X_{Ag_{exo}}$ représente les messages provenant de modèles exogènes au système multi-agents, qui peuvent être utiles pour l'initialisation de l'agent par exemple. Leur définition est laissée libre, à la discrétion du modélisateur.

De la même manière, les sorties de l'agent sont formées par trois sous-ensembles :

$$Y_{Ag} = Y_{Ag_{env}} \cup Y_{Ag_{agents}} \cup Y_{Ag_{exo}}$$

où

- $Y_{Ag_{env}} = \{(p, \gamma) \mid p \in D_E, \gamma \in \Gamma\}$ est l'ensemble des messages destinés aux environnements dans lesquels l'agent évolue, le port étant l'identifiant de l'environnement $\in D_E$ dans lequel l'agent souhaite réaliser une action, cette dernière étant représentée par une influence $\gamma \in \Gamma$, que nous détaillerons ultérieurement (section 2.4.1).
- $Y_{Ag_{agents}} = \{(p, v) \mid p \in D_{Ag}, v \in Msg\}$ représente les messages directs destinés à d'autres agents, dans le cas où l'agent évolue dans un environnement social.
- Enfin, $Y_{Ag_{exo}}$ représente les messages destinés à des modèles exogènes au système multi-agent. De la même manière que pour les entrées, leur définition est laissée à la discrétion du modélisateur.

2.2.2 *Corps*

L'agent est représenté au sein de l'environnement par un corps, un n-uplet associé à un esprit à travers l'identifiant de l'agent. Le corps permet de faire le lien entre le système cognitif de l'agent et l'environnement dans lequel il est plongé. Si un agent est plongé dans plusieurs environnements, celui-ci dispose d'un corps dans chacun d'eux.

De la même manière que Saunier (2015), nous prêtons au corps les responsabilités suivantes :

1. *Accès aux ressources et observation de l'environnement* : le corps joue le rôle de médiateur dans l'accès aux ressources de l'environnement. C'est à travers ses propriétés que

sont filtrées les informations perçues par le système cognitif. Par exemple, la situation physique de l'agent au sein de l'environnement détermine ce qu'il est en mesure de percevoir. De la même manière, les propriétés du corps peuvent déterminer si une influence est réellement prise en compte par l'environnement pendant le processus de réaction. Prenons le cas d'un bras robotisé ayant subi des lésions mécaniques. Si le système cognitif de ce dernier émet le souhait de réaliser une action, l'état de son corps peut l'en empêcher.

2. *Accès et observation de l'agent* : le corps offre un état « public » de l'agent, c'est-à-dire observable et qui peut être éventuellement modifié par une influence. Un agent doit pouvoir introspecter son corps.

Dans notre spécification, le corps ne possède pas de dynamique propre. L'état du corps évolue à travers la phase de réaction de l'environnement aux influences produites de manière endogène ou exogène.

Pour chaque agent $a \in D_{Ag}$, il existe un seul esprit Ag_a. Pour chaque environnement $e \in D_E$ dans lequel l'agent a évolue, il existe au sein de l'état de l'environnement au moins un corps $\beta_{a,e}$ défini par le triplet suivant :

$$\beta_{a,e} = (p, \psi, \theta)$$

où

- $p \in P$ représente la position du corps de l'agent dans l'environnement ;
- $\psi \in \wp(P)$ est une partie de l'ensemble des positions et représente la portée de perception/action de l'agent [1] ;
- enfin, θ est utilisé pour définir l'ensemble des variables d'état « publiques » de l'agent au sein de cet environnement et que les autres agents pourront percevoir et/ou modifier.

À travers son corps, l'agent entretient une étroite relation avec son environnement. Si l'agent perçoit et agit au sein de celui-ci, il ne peut traditionnellement le faire que de manière limitée, en fonction de sa position spatiale ou sociale et d'une certaine portée. À travers la définition de cette structure et des variables p et ψ, il est important de noter que l'agent est bien *situé* au sein de son environnement, et que la variable ψ nous permet de matérialiser la portée de ses perceptions et de ses actions. Cette définition du corps nous permet donc de satisfaire la *contrainte de localité* des perceptions/actions (Michel, 2007a).

Notons que pour des raisons de simplicité, nous avons matérialisé une seule portée de perception/action. En pratique, le modélisateur peut juger opportun de différencier la portée de l'action et la portée de perception. De la même manière, il est possible d'envisager différents niveaux de perception correspondant à des sens différents (*eg.* la vue pour une portée lointaine, le toucher pour une portée plus courte). Cette granularité de modélisation est laissée à la discrétion du modélisateur, qu'il est possible d'atteindre en utilisant la variable θ.

1. Notons que pour désigner l'ensemble des parties d'un ensemble donné, nous utiliserons systématiquement la notation $\wp$.

La distinction entre le corps et l'esprit laisse deviner l'étroite relation que l'agent entretient avec son ou ses environnements. Avant de parler des interactions entre agents, de la mécanique de perception et des influences plus en détails, nous étudions le rôle de l'environnement au sein d'un SMA avant de donner sa spécification.

2.3 *Les environnements*

Comme le soulignent Weyns et al. (2005b), l'importance de l'environnement au sein d'un SMA a toujours fait l'objet d'un consensus. Depuis une quinzaine d'années, un certain nombre de contributions ont eu pour objectif de définir plus clairement l'environnement et ses responsabilités ; Weyns et al. (2006) le définissent comme une abstraction de premier ordre qui fournit les conditions permettant aux agents d'exister et qui coordonne les interactions entre agents ainsi que l'accès aux ressources. Nous rejoignons cette définition. Pour notre spécification DPDEMAS, nous nous appuyons sur la sémantique de PDEVS et sur le mécanisme Influence/Réaction afin de permettre à l'environnement d'assumer son rôle de médiation des interactions. Afin de fournir les conditions et ressources nécessaires à l'évolution des agents, deux grand types d'environnements sont généralement distingués (Odell et al., 2003a) :

- les environnements *physiques*, qui fournissent les règles et contraintes qui régissent et permettent l'existence des agents et des ressources ;
- et les environnements *sociaux*, qui permettent la communication entre agents et l'organisation de ces derniers sous forme de société virtuelle.

Pour la spécification des environnements physiques, nous nous appuyons sur la contribution de Mathieu et al. (2016), qui propose d'utiliser un espace topologique comme base de formalisation afin de fournir aux agents un accès aux ressources et à leur voisinage. Enfin, la spécification des environnements organisationnels sera basée sur les travaux de la communauté des SMA centrés organisation (AGRE (Ferber et al., 2005), RIO (Hilaire et al., 2000), MASQ (Dinu et al., 2012)). Pour ce qui est des interactions, nous considérons l'environnement physique comme support pour les interactions indirectes ; l'environnement social comme support pour les interactions directes.

Nous donnons dans un premier temps la formalisation générique de l'environnement. Nous proposons ensuite de spécialiser cette formalisation afin de définir les spécificités des environnements physiques et organisationnels.

2.3.1 *Environnement générique*

De la même manière que pour le système cognitif de l'agent, il n'existe pas un type de modèle adapté à un type d'environnement. Si un modèle atomique PDEVS est adapté à la définition d'un environnement continu, d'autres modèles issus d'extensions du formalisme PDEVS, comme Cell-DEVS (Wainer et Giambiasi, 2001) ou une approche non-modulaire telle que multiDEVS (Zeigler et al., 2000) (dans une version compatible avec PDEVS que nous détaillons en annexe B) peut être plus adaptée à la définition d'un environnement discret cellulaire. Le premier, Cell-DEVS, a été proposé afin de faciliter la définition de systèmes

basés sur le paradigme des automates cellulaires. Le modèle Cell-DEVS peut ensuite être couplé au sein d'une hiérarchie de modèles DEVS. Le second, multiDEVS, permet d'intégrer au sein d'un modèle couplé DEVS un système non-modulaire formé par des composants capables de s'influencer sans couplages. À ce titre, il représente également un outil formel adapté à la représentation d'un modèle cellulaire. Nous renvoyons le lecteur intéressé à l'annexe A et B pour plus d'informations à ce sujet.

Nous pensons qu'il est profitable de tirer partie des capacités de multi-modélisation (*cf.* section II.2.2) du formalisme PDEVS afin de représenter l'environnement au sein de DPDEMAS. Le modélisateur peut ainsi adapter au mieux le modèle en fonction de la topologie de l'environnement visé. Au delà de l'interface, il est possible d'identifier les caractéristiques communes aux différents types d'environnements (l'état) et de définir la mécanique générale de l'environnement (le comportement). Spécifier un comportement générique présente un double intérêt :

- proposer une bibliothèque de modèles d'environnements réutilisables adaptés à différentes topologies ;
- guider le modélisateur dans la spécification d'un nouvel environnement.

Notre objectif étant de formaliser la structure et le comportement de l'environnement, nous utilisons ici sur un modèle PDEVS pour spécifier l'état et le comportement générique d'un environnement. Celui-ci peut ensuite être adapté pour définir un environnement basé sur un autre type de modèle (*e.g.* Cell-DEVS).

Soit un environnement E, ici représenté par un modèle atomique PDEVS :

$$E = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \mathrm{ta})$$

Interface d'entrées/sorties

Le rôle de médiation des interactions de l'environnement se traduit par la structure de ses entrées/sorties. Peu importe la forme que prend un environnement, ce dernier doit fournir une interface spécifique. L'ensemble des ports d'entrée de l'environnement est défini par :

$$X_E = X_{E_\chi} \cup X_{E_{inf}} \cup X_{E_{exo}}$$

où

- $X_{E_\chi} = \{(\chi, v) \mid v \in \{ack, nak\}\}$ est l'ensemble des couples possibles pour le port d'entrée χ, destiné à recevoir les messages envoyés par le modèle exécutif à l'environnement afin d'indiquer si les changements structurels demandés par l'environnement ont été effectués ;
- $X_{E_{inf}} = \{(inf, \gamma) \mid \gamma \in \Gamma\}$ est l'ensemble des couples possibles pour le port inf, destiné à recevoir les influences envoyées par les systèmes cognitifs des agents plongés dans cet environnement. Comme nous l'avons déjà évoqué, nous reviendrons dans la section 2.4.1 sur la définition d'une influence $\gamma \in \Gamma$;
- enfin, $X_{E_{exo}}$ représente les messages d'entrée provenant de modèles exogènes au système multi-agent.

L'ensemble des sorties de l'environnement est défini par :

$$Y_E = Y_{E_\chi} \cup Y_{E_{agents}} \cup Y_{Eexo}$$

où

- $Y_{E_\chi} = \{\chi\} \times Req^b$ est l'ensemble des couples possibles pour le port de sortie χ, destiné à être rattaché au modèle exécutif du modèle SMA et dont les valeurs sont un sac de demandes de modifications structurelles. Ces demandes ont pour but de répercuter sur la structure certains changements d'états de l'environnement. Les valeurs de chaque élément du sac appartiennent à l'ensemble *Req* qui a été défini section 2.1.
- $Y_{E_{agents}} = \{(p, v) \mid p \in A, v \in Pa\}$ représente l'ensemble des messages destinés aux agents plongés dans l'environnement ($A \subseteq D_{Ag}$) dont les valeurs représentent un percept.
- Enfin, Y_{exo} est l'ensemble de port-valeurs éventuels destinés à des modèles exogènes au SMA.

Etat

Nous donnons ici l'état associé à notre modèle d'environnement générique. Comme nous allons le voir dans la suite de cette section, il nous permet de généraliser une bonne partie du comportement de l'environnement. Certaines caractéristiques spécifiques aux environnements physiques et aux environnements sociaux n'y sont pas représentées. Une structure dédiée à ces informations est incluse dans l'état générique, mais nous donnons sa définition dans les sections suivantes.

L'état de notre modèle d'environnement générique est défini par le sextuplet suivant :

$$S = \{(phase, \sigma, A, \kappa, \Gamma, req^b_\chi)\}$$

où

- $phase \in \{natural, perception, req\chi, await\chi\}$ est la variable qui représente le nom donné à l'état courant de l'environnement ;
- $A \subseteq D_{Ag}$ est l'ensemble des identifiants des agents plongés dans l'environnement.
- $\Gamma = \{\Gamma_a \mid a \in A\} \cup \Gamma_e$ est l'ensemble des influences (*cf.* section 2.4.1) pouvant induire un changement d'état. Ces influences peuvent provenir de l'environnement (dynamique endogène) ou de l'activité des agents (dynamique exogène).
- $\kappa \in K$ est une structure associée à chaque état $s \in S$ de l'environnement, qui diffère en fonction du type d'environnement (physique ou social) et que nous détaillons dans les sections 2.3.2 et 2.3.3.
- $\sigma \in \mathbb{R}^+_{0,\infty}$ est une variable permettant d'indiquer le prochain temps d'activation de l'environnement. Si l'environnement n'a pas de dynamique endogène, cette valeur est toujours à l'infini.
- Enfin, $req^b_\chi \in Req^b$ est un sac utilisé pour stocker les requêtes destinées à l'exécutif. Chaque élément du sac appartient à l'ensemble *Req* qui a été défini section 2.1.

Mécanique générale

Nous associons à chaque environnement $e \in D_E$ deux fonctions, $natural_e$ et $reac_e$, permettant respectivement la dynamique environnementale et le calcul de réaction aux influences. Comme nous allons le voir, ces fonctions sont ensuite utilisées pour définir un comportement pour chaque fonction de transition de l'environnement (δ_{int}, δ_{ext}, δ_{con}). À l'instar du modèle IRM4S, la première fonction permet à l'environnement d'avoir une dynamique endogène en produisant à partir de son propre état total un ensemble d'influences :

$$natural_e : Q \to \Gamma$$

où $Q = \{(s, e) \mid s \in S, e \in \mathbb{R}^+\}$ est pour rappel, l'ensemble des états totaux de l'environnement, avec e le temps écoulé depuis la dernière transition. La deuxième fonction correspond au calcul de réaction de l'environnement, qui à partir de son état total et de l'ensemble des influences produites par les agents et l'environnement permet de calculer un nouvel état :

$$reac_e : Q \times \Gamma \to S$$

Ces deux dernières fonctions doivent être définies par le modélisateur et ont vocation à être utilisées au sein même des fonctions de transition de l'environnement. En isolant la production endogène d'influences et la réaction aux influences de l'environnement dans des fonctions spécifiques, il est possible de définir un comportement générique pour les fonctions de transitions de l'environnement.

Nous prêtons à l'environnement les responsabilités suivantes :

1. recevoir les influences des agents et/ou en produire à travers la dynamique endogène de l'environnement ;
2. réagir à l'ensemble des influences en calculant un nouvel état. Si la prise en compte d'une influence nécessite en plus d'un changement d'état de l'environnement un changement structurel, l'environnement a la responsabilité d'en effectuer la requête auprès du modèle exécutif et d'attendre que celle-ci soit traitée.
3. une fois la réaction effectuée (changement d'état), l'environnement envoie si nécessaire de nouveaux percepts aux agents.

Dans notre spécification, le macro-état *natural* correspond à la phase où l'environnement peut recevoir/produire des influences. Une fois le calcul de réaction effectué, le macro-état *perception* correspond à la phase qui consiste à envoyer les percepts aux agents. Si un changement de structure est nécessaire pendant le calcul de réaction, un mécanisme permettant de demander/attendre le nouvel état du réseau de composants est mis en place, à travers les macro-états $req\chi$ et $await\chi$.

Le port d'entrée inf permet de recevoir un ou plusieurs évènements externes correspondant aux influences produites par les agents. La production endogène d'influences est effectuée à travers un évènement interne planifié selon la variable σ. Si une influence endogène et des influences exogènes sont produites au même instant de simulation, la collision est gérée à l'aide de la fonction de transition de confluence. Ainsi, la fonction $natural_e$ peut aussi bien être appelée dans la fonction de transition interne que dans la

fonction de transition de confluence. La fonction $reac_e$ peut être appelée depuis n'importe quelle fonction de transition.

En ce qui concerne les demandes de changements structurels, ceux-ci sont gérés de la manière suivante : après modification de l'état de l'environnement par la fonction $reac_e$, si la variable $req^b_\chi \neq \varnothing$, l'environnement passe dans un état transitoire ($phase = req\chi$) afin de réaliser une sortie permettant d'envoyer les demandes de modifications au modèle exécutif SMA, puis passe dans l'état passif $phase = await\chi$ jusqu'à la réception d'un message de l'exécutif. Si les changements structurels se sont déroulés comme prévu, ou que $req^b_\chi = \varnothing$ après l'exécution de la fonction $reac_e$, alors l'environnement passe dans l'état transitoire $phase = perception$. Celui-ci permet d'effectuer les sorties nécessaires destinées aux agents, avant de repasser dans l'état *natural*, qui est destiné à l'attente/production d'influences.

Le comportement global que nous venons de décrire est représenté par une variante du diagramme d'états-transitions en figure 3.2, dont la syntaxe est inspirée de Le Hung et al. (2015). Chaque état est représenté par sa phase et sa durée. Les différentes transitions (interne, externe ou confluence) sont représentées par une flèche et un trait spécifique (*cf.* légende). Les transitions sont accompagnées de la notation issue des diagrammes d'états-transitions protocole (*Protocol State Machines*), et fait donc apparaître une pré-condition, un déclencheur et une post-condition. Un état de destination peut être atteint si la pré-condition et/ou le déclencheur est vérifié dans l'état d'origine, et si la post-condition est satisfaite dans l'état d'origine. Prenons l'exemple où l'environnement est dans l'état *natural*.

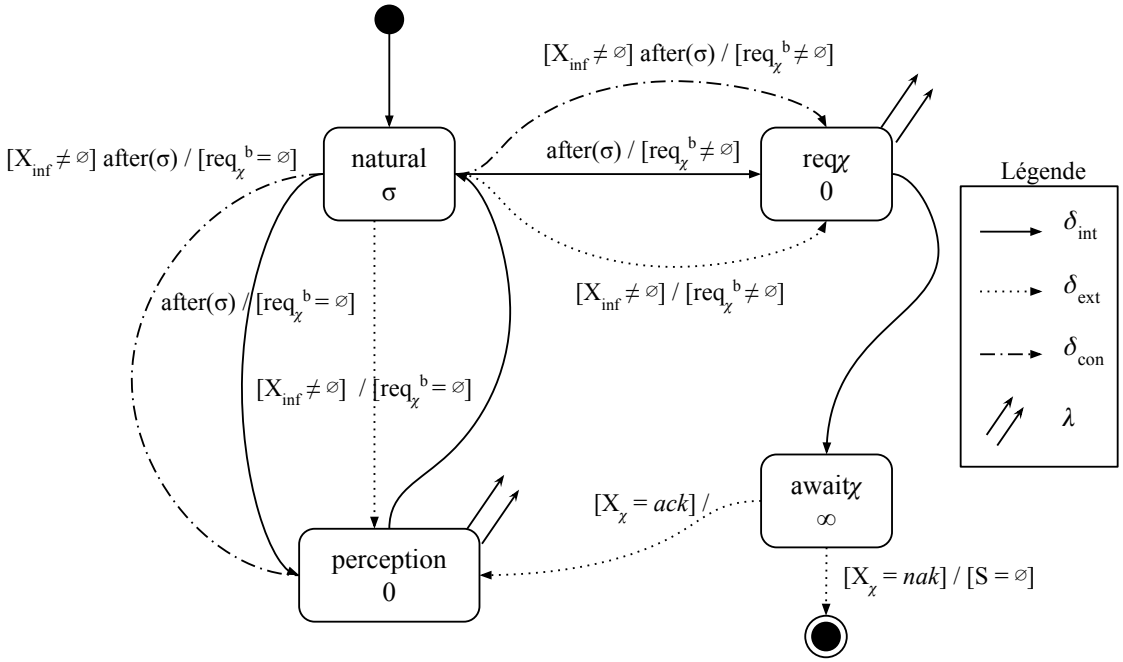


Figure 3.2 Diagramme d'états-transitions de l'environnement

La réception d'influences exogènes (pré-condition $[X_{inf} \neq \emptyset]$ satisfaite), et la production endogène d'influences (déclencheur $after(\sigma)$ actif) entraîne une transition de confluence. Selon le calcul de réaction, l'état de destination peut être soit *perception* (post-condition

$[req^b_\chi = \emptyset]$ vérifiée), soit $req\chi$ (post-condition $[req^b_\chi \neq \emptyset]$ vérifiée).

Maintenant que nous avons décrit le comportement dans son ensemble, nous spécifions le comportement des fonctions de l'environnement. Soit un environnement $e \in D_E$ dans l'état $s_e = (phase, \sigma, A, \kappa, \Gamma, req^b_\chi)$, auquel a été associé les fonctions $natural_e$ et $reac_e$. La fonction d'avancement du temps est définie par :

$$ta(s_e) = \begin{cases} 0 & \text{si } phase \in \{req\chi, perception\} \\ \infty & \text{si } phase = await\chi \\ \sigma & \text{si } phase = natural \end{cases}$$

La sortie associée à l'état $req\chi$ correspond à l'envoi au modèle exécutif des demandes de modifications structurelles stockées dans la variable req^b_χ :

$$\lambda(phase, \sigma, A, \kappa, \Gamma, req^b_\chi) = \{(\chi, req^b_\chi)\}$$

La sortie associée à la phase *perception* ne peut pas être détaillée dans cette section, car elle dépend du type d'environnement, et n'est donc en aucun cas générique. En revanche, quelque soit le type d'environnement (social ou physique), une fonction de voisinage générique pourra être utilisée pour déterminer la perception que les agents auront de l'environnement. Nous associons donc à chaque environnement e une fonction de voisinage permettant de déterminer les agents perceptibles par un agent, à laquelle un comportement différent sera associé selon le type d'environnement :

$$\nu_e : A \to \wp(\beta)$$

En fonction de l'état de l'environnement, la fonction de transition interne a un comportement différent :

$$\delta_{int}(s_e) = \begin{cases} reac_e((s_e, 0), natural_e(s_e, 0) \cup \Gamma) & \text{si } phase = natural \\ (await_\chi, \sigma, A, \kappa, \Gamma, \varnothing) & \text{si } phase = req_\chi \\ (natural, \sigma, A, \kappa, \Gamma, req^b_\chi) & \text{si } phase = perception \end{cases}$$

Si l'environnement est dans l'état *natural*, l'environnement effectue sa dynamique endogène à travers la fonction $natural_e$ puis le nouvel état est calculé grâce à la fonction $reac_e$. La phase qui résulte de la fonction $reac_e$ peut être de deux types : $req\chi$ si $req^b_\chi \neq \varnothing$ ou *perception* si $req^b_\chi = \varnothing$. Si $phase = req\chi$, l'environnement vient d'effectuer une sortie pour le modèle exécutif. Il passe alors dans l'état passif $await\chi$ et vide la variable req^b_χ. Enfin, si $phase = perception$, l'environnement vient d'envoyer les percepts aux agents et passe donc dans l'état *natural*.

En ce qui concerne la fonction de transition externe, sa définition est moins concise, nous donnons donc son comportement pour chaque phase. Lorsque l'environnement est dans la phase *natural*, il peut recevoir à tout moment des influences de la part d'autres agents, qu'il

traite grâce à la fonction $reac_e$:

$$\delta_{ext}((natural, \sigma, A, \kappa, \Gamma, req^b_\chi), e, ((inf, \gamma_1), (inf, \gamma_2), ..., (inf, \gamma_n))) = reac_e(q_e, \bigcup_{i=1}^{n} \gamma_i \cup \Gamma)$$

Lorsque l'environnement reçoit une réponse concernant les changements de structure demandés à l'exécutif du SMA ($phase = await\chi$) sur le port d'entrée χ, un message *ack* indique que le modèle exécutif a bien effectué les modifications demandées. L'environnement passe alors dans l'état *perception* afin d'envoyer les percepts aux agents. À l'inverse, un message *nak* indique que les demandes n'ont pu être traitées par l'exécutif. Dans cette éventualité, l'environnement passe à un état vide, qui entraîne en pratique l'arrêt de la simulation :

$$\delta_{ext}((await\chi, \sigma, A, \kappa, \Gamma, req^b_\chi), 0, (\chi, v)) = \begin{cases} (perception, \sigma - e, A, \kappa, \Gamma, req^b_\chi) & \text{si } v = ack \\ \varnothing & \text{si } v = nak \end{cases}$$

Il nous a semblé nécessaire de couvrir l'éventualité d'un retour négatif de l'exécutif, car l'ignorer pourrait conduire à des incohérences.

Enfin, la fonction de transition de confluence permet de gérer la collision entre la dynamique endogène et la réception d'influences produites par les agents au même instant :

$$\delta_{con}((natural, \sigma, A, \kappa, \Gamma, req^b_\chi), ((inf, \gamma_1), (inf, \gamma_2), ..., (inf, \gamma_n))) \\ = reac_e(q_e, \bigcup_{i=1}^{n} \gamma_i \cup \Gamma \cup natural_e(q_e))$$

Topologie

Afin de « situer » les agents au sein de l'environnement, nous associons à l'environnement la fonction, $dist_e$, dont le rôle est de définir une distance entre deux éléments de l'ensemble P :

$$dist_e : P \times P \to \mathbb{R}^+_\infty$$

Les éléments de l'ensemble P représentent une position physique ou sociale dans un espace. Le tuple $(P, dist_e)$ forme un espace *hémimétrique*[2] pour lequel la fonction de distance définit une topologie. L'hémimétrique généralise la notion de *quasimétrique* (Steen et Seebach Jr, 1978), utilisée par Mathieu et al. (2016) pour formaliser l'environnement.

L'utilisation d'un tel espace nous permet de couvrir un nombre varié de types d'environnements physiques, discrets ou continus, comme nous le verrons dans la section suivante. Par exemple, un environnement euclidien à trois dimensions peut être représenté avec $P = R^3$ avec pour $dist_e$ la distance euclidienne. Comme le font remarquer Mathieu et al. (2016), ce qui différencie les environnements physiques des environnements sociaux, est le fait que les premiers établissent une relation de distance, tandis que les seconds établissent

2. Une *hémimétrique* satisfait les propriétés de réflexivité, de positivité et d'inégalité triangulaire.

des relations topologiques. Cependant, une topologie peut être établie à partir d'une relation de distance, et inversement.

Néanmoins, d'autres différences subsistent, c'est pourquoi nous proposons une structure quelque peu différente pour les environnements physiques et les environnements sociaux, que nous présentons dans les sous section suivantes. Maintenant que nous avons clairement défini la mécanique générale de l'environnement et rendu explicite son rôle de médiateur d'interactions, nous donnons dans les sections suivantes la définition des structures κ, d'abord pour l'environnement physique, puis pour l'environnement organisationnel afin de traiter pour chacun d'eux leurs propriétés spécifiques.

2.3.2 *Environnement physique*

L'environnement physique fournit les règles et contraintes qui régissent et permettent l'existence des agents et des ressources. L'espace hémimétrique que nous avons défini précédemment est la base de formalisation qui va nous permettre de fournir aux agents un accès aux ressources et à leur voisinage. Cependant, pour les environnements physiques, la fonction hémimétrique doit vérifier l'axiome de séparation, qui implique que deux positions distinctes doivent obligatoirement être discernables :

$$\forall p_1, p_2 \in P, \qquad dist_\varphi(p_1, p_2) = 0 \iff p_1 = p_2$$

Cette contrainte vérifiée, l'espace $(P, dist_\varphi)$ devient un espace *quasimétrique*. Celui-ci nous permet de définir un nombre varié de topologies discrètes ou continues. Par exemple :

- un environnement continu 2D peut être représenté par $P = \mathbb{R}^2$ et $dist_\varphi$ la distance euclidienne ;
- un environnement cellulaire 2D peut être représenté par $P = \mathbb{Z}^2$ et $dist_\varphi$ la distance de Tchebychev pour un voisinage de Moore ou la distance de Manhattan pour un voisinage de Von Neumann ;
- pour un graphe, P peut représenter un ensemble de sommets avec $dist_\varphi$ la distance géodésique. La propriété de symétrie ne devant pas nécessairement être vérifiée, il est également possible de représenter un graphe orienté.

Afin de simplifier la spécification, nous utilisons dans cette section la notation « point », conventionnellement utilisée dans les langages de programmation orientés objet (*e.g.* l'élément $\beta_{a,\varphi}.p$ fait référence à l'élémént p du n-uplet $\beta_{a,\varphi}$). Dès à présent, nous utilisons l'identifiant $\varphi \in D_E$ pour désigner l'environnement physique.

Soit $s_\varphi = (phase, \sigma, A, \kappa, \Gamma, req^b_\chi)$, l'état d'un environnement *physique* $\varphi \in D_E$ tel que défini précédemment. La structure κ associée à cet état $s_\varphi \in S_\varphi$ est définie par le couple suivant :

$$\kappa = (\Xi, \theta)$$

où

- Ξ représente l'ensemble des entités situées au sein de l'environnement. Nous distinguons deux types d'entités : les corps des agents tels que définis dans la section 2.2.2 et les

ressources manipulables par les agents. Nous notons l'ensemble des corps situés dans l'environnement $\beta = \{\beta_{a,\varphi} \mid a \in A\}$. Les ressources, quant à elles, sont définies par l'ensemble $R = \{(p, \vartheta) \mid p \in P\}$, où p représente la position de la ressource au sein de l'espace et ϑ est une variable utilisée pour représenter les informations relatives à cette ressource. Ainsi, $\Xi = \beta \cup R$.

- θ est une variable dont la définition est laissée à la discrétion du modélisateur afin qu'il puisse définir de nouvelles variables environnementales.

De la même manière que Mathieu et al. (2016), nous associons à chaque environnement physique $e \in D_E$ un certain nombre de fonctions permettant de donner la position d'un agent (pos_φ), de calculer le voisinage d'un agent (ν_φ) ou de donner le contenu associé à un point ou à une partie de l'espace ($cont_\varphi$).

Situer les agents et les ressources

La fonction pos_φ permet de donner la position d'un agent. Il est possible de lui associer un comportement générique étant donné que la position de chaque agent est déjà spécifiée dans le corps correspondant :

$$\begin{aligned} pos_\varphi : A &\to P \\ a &\mapsto \beta_{a,\varphi}.p \end{aligned}$$

La fonction $cont_\varphi$ permet de donner le contenu associé à un point ou à une partie de l'environnement, soit à une partie de l'ensemble des entités (corps ou ressources) :

$$\begin{aligned} cont_\varphi : \wp(P) &\to \wp(\Xi) \\ \{p\} &\mapsto \{\xi \in \Xi \mid \xi.p = p\} \end{aligned}$$

Calcul du voisinage

La fonction de voisinage ν_φ, que nous avons associée à chaque environnement, permet à un agent de connaître les autres agents avec qui il peut interagir :

$$\nu_\varphi : A \to \wp(\beta)$$

Il est possible d'associer un comportement générique à la fonction de voisinage. Pour l'environnement physique, nous utilisons la variable ψ présente dans chaque corps et qui détermine la portée de perception de l'agent pour calculer les agents présents dans cette zone :

$$\nu_\varphi : a \mapsto \{\beta_{i,\varphi} \mid i \in A \setminus \{a\}, pos_\varphi(i) \in \beta_{a,\varphi}.\psi\}$$

Notons cependant que cette fonction de voisinage tient compte uniquement de la position de l'agent. Dans le cas où le corps d'un agent peut occuper une partie de l'espace (*e.g.* une forme géométrique dans un environnement euclidien), une autre fonction de voisinage peut être envisagée. Considérons par exemple que la variable θ d'un corps $\beta_{a,\varphi}$ soit définie comme un n-uplet contenant une variable $shape \in \wp(P)$ représentant la forme du corps au sein de l'environnement. La fonction de voisinage, plutôt que de prendre en compte

la position du corps des autres agents, peut alors utiliser la variable *shape* pour vérifier si l'intersection avec la portée de perception de l'agent est non vide. Nous aurions alors :

$$\nu_\varphi : a \mapsto \{\beta_{i,\varphi} \mid i \in A \setminus \{a\}, \beta_{a,\varphi}.\psi \cap \beta_{i,\varphi}.\theta.shape \neq \varnothing\}$$

La figure 3.3 permet d'illustrer les deux différentes méthodes de calcul du voisinage. Sur la figure 3.3(a), la méthode de calcul est basée sur la position du corps dans l'environnement. Les agents b et d sont dans le voisinage de a, mais pas c. Ainsi, $\nu_\varphi(a) = \{\beta_{b,\varphi}, \beta_{d,\varphi}\}$. Sur la figure 3.3(b), le calcul est basé sur la géométrie du corps, grâce à la variable *shape*. Les agents b, c et d sont cette fois dans le voisinage de a. Ici, $\nu_\varphi(a) = \{\beta_{b,\varphi}, \beta_{c,\varphi}, \beta_{d,\varphi}\}$.

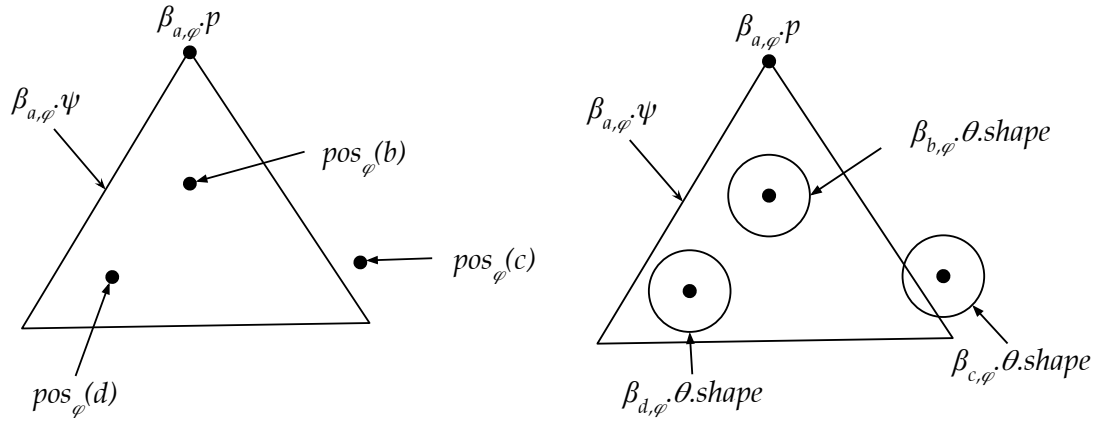


Figure 3.3 Illustration de deux méthodes de calcul de voisinage pour un agent a.

Les agents n'ont pas d'accès direct à ces fonctions. Celles-ci sont utilisées par l'environnement au moment d'envoyer les percepts aux agents. Bien que nous n'ayons pas encore détaillé le comportement de la fonction de sortie pour l'état *perception*, c'est à ce moment que ces fonctions sont utilisées. Nous revenons dans la section 2.4 sur cet aspect.

Bilan

Cette formalisation de l'environnement physique, basée sur un espace topologique, permet de représenter tous les types d'environnement physiques continus ou discrets traditionnellement utilisés dans les SMA : environnement continu 2d ou 3d, environnement cellulaire 2d ou 3d ou encore environnement graphe orienté ou non.

Parmi les catégories d'environnements physiques que Soulié (2001) distingue (*centralisé*, *distribué* et *agent*), notre proposition est à placer dans celle des environnements centralisés, étant donné que notre environnement forme une structure unique dans laquelle l'ensemble des informations sont rassemblées. Toutefois, comme nous l'avons souligné en introduction de cette section, il est possible de tirer partie des capacités de multi-modélisation du formalisme PDEVS pour adapter l'environnement en fonction de la topologie visée. Cette remarque reste valable pour envisager une version distribuée de l'environnement.

Par exemple, il est possible de proposer un environnement continu distribué en couplant plusieurs environnements entre eux (l'union de leurs états forme l'état global de l'environnement). L'extension Cell-DEVS (*cf.* annexe A) peut être utilisée à cette fin, comme le suggèrent Duboz et al. (2005). En revanche, il est nécessaire d'envisager une interface d'entrées/sorties étendue pour permettre le couplage des environnements et de mettre en place un mécanisme permettant l'échange des entités entre chaque parcelle. Si on considère le déplacement d'un agent, il peut être nécessaire de transférer son corps d'une parcelle à l'autre. De la même manière, la portée de perception d'un agent peut éventuellement chevaucher plusieurs parcelles ; un mécanisme permettant de lire l'état de plusieurs parcelles doit être mis en place. Ces questions ne sont pas abordées dans ce mémoire.

La dernière catégorie d'environnement, celle des *agents* n'est pas abordée dans le cadre de notre spécification DPDEMAS. Bien que nous ne mentionnons pas de manière explicite qu'un environnement puisse être considéré comme un agent, la frontière entre les deux est mince. La possibilité de modéliser des processus environnementaux par exemple, est possible à travers une dynamique endogène de l'environnement : celui-ci produit des influences de manière proactive, au même titre qu'un agent. Notre spécification soumet également l'environnement aux mêmes contraintes qu'un agent : son état n'est pas directement modifiable : il est donc autonome. Selon nous, la frontière entre les deux doit être considérée selon une perspective conceptuelle. L'environnement est une entité à part entière aux responsabilités bien définies, qui permet de faire coexister et interagir plusieurs agents. Le considérer comme un agent, si l'on suit notre spécification, implique qu'il puisse exister au sein d'un autre environnement et qu'il puisse influencer celui-ci. Nous ne considérons pas cette perspective, néanmoins elle peut être explorée afin de représenter un lien de cause à effet entre deux environnements. Par exemple, lorsqu'un agent effectue une action dans un environnement qui a pour conséquence de modifier l'état d'un autre environnement.

2.3.3 *Environnement social*

La communauté des SMA centrés organisation s'est intéressée à la manière d'organiser les agents pour former une société virtuelle. Nous avons classé les contributions relatives à l'organisation selon trois axes (*cf.* I.3.4) : l'organisation structurelle, l'organisation des comportements et l'organisation de la connaissance collective. Pour définir l'environnement social, nous nous basons sur les concepts définis par les travaux sur l'organisation structurelle, ceux-ci s'attachant à structurer les agents afin d'établir les liens qui les unissent ; ou au contraire, qui les opposent. L'environnement que nous décrivons dans cette section est basé sur les concepts de groupes et de rôles, que l'on retrouve notamment dans les métamodèles AGRE (Ferber et al., 2005) ou RIO (Hilaire et al., 2000).

Notre spécification ne traite pas les aspects relatifs aux normes, à l'engagement social, aux sanctions, aux institutions ou encore à la culture. Toutefois, ces notions peuvent être considérées dans de futurs travaux visant à faire évoluer la spécification DPDEMAS.

Nous avons choisi de dissocier l'environnement social de l'environnement physique bien le premier soit basé sur un espace topologique, et par conséquent, suffisamment général

pour représenter un graphe formé par des groupes et des rôles. Etant donné que nous donnons à l'environnement social la responsabilité des interactions directes (comme nous l'avons précisé chapitre I), nous avons choisi de tirer partie de la topologie formée par les couplages entre les différents modèles PDEVS pour établir les interactions directes entre les agents. L'environnement social est alors responsable d'effectuer les requêtes d'ajout ou de suppression des couplages directs entre le système cognitif des agents et le modèle d'exécutif. Ces derniers sont établis en fonction de l'appartenance d'un agent à un groupe et des rôles qu'il y joue.

L'utilisation d'un environnement social permet au modélisateur de spécifier la manière dont les agents sont organisés d'un point de vue macroscopique, à travers une approche descendante. L'intérêt d'une telle organisation, en plus d'offrir aux agents la possibilité de communiquer, est de créer des structures hiérarchiques bien définies. À travers les notions d'appartenance à des groupes et en rendant explicite les rôles des agents, il est possible de restreindre les communications entre agents de groupes différents et ainsi, de maintenir une hiérarchie sans que les agents n'aient à gérer ces aspects sociaux (Weyns et al., 2005a).

Au sein d'un environnement social, la communication repose sur des envois de messages entre les différents agents (mode d'interaction direct). Si l'unique rôle de l'environnement social est de restreindre les communications entre agents d'un même groupe, il est possible de mettre en place des couplages directs entre les systèmes cognitifs des agents. Ceux-ci étant représentés par des modèles atomiques ou couplés, la gestion de la simultanéité (réception de plusieurs messages provenant d'autres agents) peut être effectuée de manière décentralisée étant donné que la sémantique de PDEVS est identique à celle du modèle IRM4S. L'environnement social est alors responsable de la gestion dynamique de ces couplages en fonction de son état. Par ailleurs, les agents continuent d'utiliser des influences pour demander un changement d'état à l'environnement (demande d'ajout à un groupe, modification de ses rôles, ...) et peuvent percevoir les membres et rôles des groupes.

En ce qui concerne son état, nous utilisons la structure κ afin d'y faire apparaître la notion de groupe et de rôles. Soit $s_o = (phase, \sigma, A, \kappa, \Gamma, req^b_\chi)$, l'état d'un environnement social $o \in D_E$ tel que défini dans la section 2.3.1. La structure $\kappa \in K$ associée à chaque état $s_o \in S_o$ de l'environnement est définie par :

$$\kappa = (G, \{\Im_g\}, Edges, \beta, \theta)$$

où

- G est l'ensemble des identifiants des groupes ;
- $\{\Im_g \mid g \in G\}$ est un ensemble de structures permettant de définir les groupes.
- $Edges = \{(r_1, r_2)\} \in P^2$ est l'ensemble des liaisons possibles entre les rôles. Notons que dans le cadre d'un environnement social, les positions sont assimilées aux rôles. Ces derniers sont donc considérés comme des positions sociales [3].
- $\beta = \{\beta_{(a,r),o} \mid a \in A, r \in P\}$ représente l'ensemble des corps des agents. Nous notons

3. Nous notons donc les éléments de l'ensemble des positions $r \in P$ afin de rappeler au lecteur qu'une position sociale est un rôle.

que pour un environnement social, la notation $\beta_{a,o} \subseteq \beta$ fait référence à l'ensemble des corps de l'agent a.

- θ peut être utilisé pour permettre l'ajout de nouvelles variables environnementales.

Pour chaque identifiant de groupe $g \in G$, la structure $\eth_g$ correspondante est formée par une partie de l'ensemble des couples agent-rôle :

$$\eth_g = \wp(A \times R_g)$$

où R_g est l'ensemble des identifiants des rôles possibles pour le groupe g. Cette structure permet d'identifier les membres d'un groupe ainsi que les rôles joués pour chacun d'eux. Pour chaque couple (a, r) d'un groupe g, il existe un corps $\beta_{(a,r),o} \in \beta_{a,o}$. Un agent peut donc avoir plusieurs corps au sein de l'environnement social, pour chacune de ses positions sociales.

Au même titre qu'un environnement physique, il est possible de « situer » le corps d'un agent en fonction de sa position à travers l'ensemble P utilisé pour former l'espace topologique $(P, dist_o)$. Nous posons le principe qu'une position sociale est définie par le rôle joué dans un groupe. Ainsi, pour un environnement social, l'ensemble P est défini par :

$$P = \bigcup_{g \in G} R_g$$

Comme nous l'avons déjà mentionné, l'environnement social est le support des communications directes entre les agents. Sa responsabilité est de maintenir les couplages entre les systèmes cognitifs des agents en fonction de leur appartenance aux différents groupes. De manière comparable au métamodèle AGR (Ferber et al., 2003), deux agents peuvent communiquer uniquement s'ils sont membres du même groupe. Nous verrons dans la suite de cette section que la définition des couplages entre agents dépend directement de la topologie formée par les rôles. Il est possible de voir les rôles sous forme de graphe, où ceux-ci sont représentés par les sommets et les arêtes qui représentent les interactions possibles entre les agents en fonction du rôle qu'ils jouent. Puisque les interactions ont lieu entre agents membres d'un même groupe, les rôles prenant place dans un groupe peuvent être liés par une arête. Afin de permettre au modélisateur de définir plus précisément les différentes interactions possibles, nous lui donnons la possibilité de définir les rôles « atteignables » depuis un rôle à travers l'ensemble *Edges*, qui représentent les différentes arêtes du graphe. Les interactions possibles sont donc représentées par le graphe (*P*, *Edges*).

La topologie de l'environnement étant déterminée par la fonction de distance sur les éléments de l'ensemble P, nous exprimons formellement le fait qu'un agent ne puisse communiquer qu'avec les membres de son groupe en ajoutant une contrainte supplémentaire sur la fonction $dist_o$:

$$\forall r_1, r_2 \in P, \qquad 0 < dist_o(r_1, r_2) < \infty \implies \exists g \in G \mid r_1, r_2 \in R_g$$

Etant donné que les interactions sont restreintes aux membres d'un même groupe, seul un agent membre de deux groupes distincts peut faire l'intermédiaire entre ces deux

groupes. Afin d'exprimer cette contrainte, nous utilisons la possibilité pour la fonction hémimétrique $dist_o$ de définir une distance égale à 0 pour deux positions différentes. Le rôle étant considéré comme une position, deux rôles, pourtant distincts, peuvent être définis de manière indiscernable. Nous considérons que si la distance entre deux rôles r_a et r_b est égale à 0, ces derniers doivent être joués par le même agent. Cette nouvelle contrainte est également ajoutée à la fonction $dist_o$:

$$\forall r_a, r_b \in P \mid a, b \in A, \qquad dist_o(r_a, r_b) = 0 \implies a = b$$

Dès lors, il est possible de représenter la notion de « verticalité » dans une organisation hiérarchique. Par exemple, si on considère que le responsable de session d'un congrès scientifique doit également être un membre du comité de programme, ces deux rôles doivent être joués par le même agent au sein de deux groupes différents : le groupe d'examination et le comité de programme. Il est important de souligner que la propriété de symétrie n'étant pas nécessairement vérifiée par la fonction $dist_o$, la contrainte peut également être « orientée ». Pour reprendre notre exemple, un membre du comité de programme n'est pas forcément responsable de session, seul l'inverse est vrai.

Les relations topologiques établies par le graphe de rôles et les groupes permettent de construire la relation de distance. Afin de définir la fonction de $dist_o$ associée à l'environnement, nous nous basons sur la distance entre deux sommets du graphe de rôles $(P, Edges)$, qui correspond à la métrique des plus courts chemins. Nous notons celle-ci $d_P(r_a, r_b)$. Cependant, puisque nous considérons deux rôles indiscernables lorsqu'ils sont liés malgré leur appartenance à deux groupes distincts, la métrique des plus courts chemins sur le graphe de rôles n'est pas suffisante. Il est nécessaire de tenir également compte du nombre de groupes distincts qui permettent d'atteindre un rôle r_b depuis un rôle r_a. Si nous notons cette relation $d_G(r_a, r_b)$, alors la fonction $dist_o$ pour les environnements sociaux est définie par :

$$dist_o(r_1, r_2) = d_P(r_1, r_2) - d_G(r_1, r_2)$$

La portée de perception d'un agent $a \in A$ jouant un rôle r_a est définie par l'ensemble des agents avec qui a peut communiquer. Pour cela, la notion de rôle est utilisée : pour le rôle r_a, l'agent a peut communiquer avec l'ensemble des agents jouant un rôle à une distance de 1 :

$$\beta_{(a,r_a),o}.\psi = \{r_i \in P \mid i \in A, dist_o(r_a, r_i) = 1\}$$

Calcul du voisinage

Etant donné qu'au sein d'un environnement social, un agent dispose d'un corps pour chaque rôle qu'il joue, le comportement de la fonction de voisinage diffère de celui que nous avons associé aux environnements physiques. Afin de faciliter la définition de son comportement, nous associons la fonction $roles_o$ à chaque environnement social, qui permet

de donner l'ensemble des rôles joués par l'agent :

$$\begin{aligned} roles_o : A &\to \wp(P) \\ a &\mapsto \{\beta_{(a,p),o}.p\} \end{aligned}$$

Pour calculer le voisinage d'un agent, nous nous appuyons sur le graphe de rôle. Pour chaque rôle joué par un agent, il est possible de déterminer les autres agents avec qui il est possible d'interagir en suivant le graphe de rôles. Nous associons à l'environnement social le comportement de la fonction de voisinage suivant :

$$\begin{aligned} \nu_o : A &\to \wp(\beta) \\ a &\mapsto \{\beta_{(i,r_i),o} \in \beta_{i,o} \mid i \in A \setminus \{a\}, r_i \in \bigcup_{r_a \in roles(a)} \beta_{(a,r_a),o}.\psi\} \end{aligned}$$

2.4 *Interaction*

L'interaction désigne une action réciproque permettant à deux agents d'échanger des informations, de manière directe ou indirecte. Le mode indirect est possible à travers l'environnement physique ; le mode direct à travers l'environnement social. La communication indirecte s'effectue grâce aux influences (qui permettent de modifier l'état de l'environnement), et la perception des agents (qui permet de découvrir ces changements d'états). Les communications directes sont possibles à travers la définition de couplages directs entre les systèmes cognitifs des agents. Ces couplages sont établis par l'environnement social, qui les maintient selon les rôles joués par les agents ainsi que leur appartenance aux différents groupes. Afin de définir plus précisément la nature des interactions, nous définissons maintenant les influences, les percepts, la mécanique permettant de gérer les couplages directs et enfin, la manière dont le calcul des percepts est effectué par l'environnement.

Cette section est organisée en trois parties : les deux premières définissent l'influence et le calcul des percepts, qui sont tous les deux le support des communications indirectes. La dernière section montre la gestion des couplages agent-agent par l'environnement social, supports des communications directes.

2.4.1 *Influence*

Conformément au mécanisme Influence/Réaction (Ferber et Müller, 1996) et au modèle IRM4S (Michel, 2007a), les actions des agents sont représentées par des tentatives d'influences sur l'état de leur environnement. Ainsi, le agents ne modifient pas l'état de l'environnement à leur gré. Les influences sont traitées par l'environnement pendant la phase de réaction à travers la fonction de réaction $reac_e$ (*cf.* section 2.3).

Dans le modèle IRM4S (Michel, 2007b), la structure d'une influence n'a pas fait l'objet de formalisation particulière ; elle représente une intention d'action, sous forme de message

arbitraire destiné à l'environnement. Le formalisme PDEVS, qui est basé sur l'échange de messages pour assurer la communication entre composants remplit donc parfaitement ce rôle. Néanmoins, nous proposons de matérialiser une influence par le nom de l'action que l'agent souhaite effectuer, accompagnée d'éventuels paramètres. Dans le cadre de notre spécification, nous donnons la structure générique d'une influence afin d'y inclure un certain nombre de paramètres utiles à la fois pour son traitement par la fonction *reac* de l'environnement et pour le calcul de la perception.

Une influence $\gamma \in \Gamma$ est définie par le quintuplet suivant :

$$\gamma = (phase, n, p, a, \theta)$$

où

- $phase \in \{pending, satisfied, rejected\}$ représente le macro-état de l'influence, qui évolue lorsque celle-ci est traitée par l'environnement (plus particulièrement par la fonction *reac*) ;
- n est une variable qui représente le nom donné à l'intention d'action, par exemple *move* pour se déplacer ;
- $p \in P$ est une position (physique ou sociale) au sein de l'environnement afin de matérialiser la cible de l'influence ;
- $a \in D_{Ag}$ est l'identifiant de l'agent qui est l'auteur de l'influence ;
- enfin, θ est une variable permettant de définir d'autres paramètres qui peuvent être représentés par une structure arbitraire.

Afin d'illustrer le cycle de vie d'une influence, prenons l'exemple suivant : un agent a plongé dans un environnement cellulaire 2D ($P = \mathbb{Z}^2$), souhaite se déplacer à une position (x, y). Suivant notre spécification, le système cognitif de cet agent envoie l'influence suivante à l'environnement :

$$\gamma = (pending, move, (x, y), a, \emptyset)$$

De manière instantanée, l'environnement effectue le calcul de réaction à travers la fonction *reac*. En fonction des règles définies par le modélisateur, si le déplacement est considéré comme valide (*e.g.* la position visée est une cellule adjacente, pas de franchissement d'obstacles, ...), l'influence peut être prise en compte. Pour cela, la fonction de réaction modifie l'état de l'environnement à partir de l'état courant et des informations présentes dans une influence. Dans notre exemple, *reac* peut modifier le corps de l'agent souhaitant se déplacer en utilisant la cible de l'influence ($\beta_{a,e}.p = \gamma.p$). Parmi les changements d'états de l'environnement, la fonction *reac* a la responsabilité de faire évoluer la phase de l'influence. Si celle-ci est prise en compte, $phase = satisfied$; sinon, $phase = rejected$. Comme nous le verrons dans la prochaine section, la phase représente un moyen pour les agents de savoir si l'état de l'environnement a évolué en leur faveur.

2.4.2 *Perception*

L'utilisation de l'approche à évènements discrets pour la formalisation du paradigme agent pose le problème de la perception des agents. En effet, lorsqu'une influence est validée

par l'environnement pendant la phase de réaction, il est nécessaire de s'assurer que les agents inactifs puissent percevoir le changement d'état de l'environnement. À noter que l'utilisation du temps discret ne pose pas cette problématique : étant donné que tous les agents sont activés simultanément de manière périodique, l'ensemble des états de l'environnement peuvent être perçus par l'ensemble des agents. Deux stratégies différentes peuvent donc être envisagées : (1) l'agent sollicite l'environnement (perception active) ; (2) l'environnement envoie les percepts aux agents. Dans un SMA à évènements discrets, certains agents peuvent demeurer inactifs indéfiniment. En conséquence, l'emploi de la première stratégie signifie qu'un agent inactif peut « manquer » un état de l'environnement. Ceci n'étant pas acceptable, il est nécessaire de mettre en place un mécanisme permettant d'informer les agents concernés par un changement d'état de l'environnement.

À notre connaissance, cette problématique a été abordée uniquement par Quesnel (2006), où une approche hybride est utilisée. Pour s'assurer que les agents ne manquent aucune information, Quesnel (2006) propose le *déclencheur*, un modèle associé à une variable d'état de l'environnement auquel les agents peuvent souscrire. Le modèle est activé depuis un évènement de sortie de l'environnement lorsque la variable en question est modifiée en son sein. Le modèle déclencheur relaie ensuite ce message aux agents. Le principal inconvénient de cette approche réside dans le fait qu'elle demande au modélisateur d'anticiper l'ensemble des perceptions possibles pour chaque agent et de les concrétiser à travers des couplages. De plus, si l'agent est capable d'adapter son comportement, il peut être nécessaire de solliciter un certain nombre de modifications du graphe de couplages auprès de l'exécutif du modèle couplé SMA. Comme le note Soulié (2012), cette technique s'adapte mal à une augmentation de sa complexité, lorsque le nombre d'agents ou d'interactions devient important.

Dans la spécification DPDEMAS, c'est l'environnement qui a la responsabilité d'envoyer les percepts aux agents. L'environnement envoie à chaque agent toutes les variables ayant été modifiées et qui sont susceptibles de l'intéresser, autrement dit tout ce qu'il est capable de percevoir. Ainsi, contrairement à l'approche de Quesnel (2006), l'agent n'a pas besoin de « souscrire » à une variable d'état de l'environnement. Nous évitons l'utilisation d'un modèle intermédiaire (déclencheur). Ainsi, le modélisateur n'a pas besoin de gérer la souscription aux différentes variables d'intérêt pour chaque agent. Dans notre spécification, une influence est ciblée à travers sa variable p. Grâce à la portée de perception/action ψ définie dans le corps des agents (*cf.* section 2.2.2), il est possible de calculer l'ensemble des agents concernés par une influence si celle-ci a été validée pendant la phase de réaction, et qui a donc entraîné un changement d'état de l'environnement.

Soit une influence telle que définie dans la section précédente produite pour un environnement physique $\varphi \in D_E$:

$$\gamma = (phase, n, p, a, \theta)$$

Pour chaque corps $\beta_{a,\varphi} \in \beta$:

$$\beta_{a,\varphi} = (p_{a,\varphi}, \psi_{a,\varphi}, \theta_{a,\varphi})$$

L'agent a est concerné par l'influence γ si et seulement si $p \in \psi_{a,\varphi}$. Autrement dit si la cible $p \in P$ de l'influence est incluse dans la partie $\psi_{a,\varphi} \in \wp(P)$ représentant la portée de

perception/action de l'agent.

La figure 3.4 permet de visualiser le calcul des agents concernés par une influence. Dans cet exemple, trois agents (a_1, a_2 et a_3) sont plongés dans un environnement physique. Leur portée de perception est délimitée par un trait formé de tirets. L'environnement contient une ressource r, représentée par une pomme. L'agent a_2 a produit une influence $\gamma_{eat} = (pending, eat, r.p, a_2, r)$, ciblée sur la position de la pomme, afin de tenter de la manger. Si l'influence est validée par la fonction $reac_\varphi$, c'est-à-dire que la pomme est mangée par l'agent a_2, l'environnement est en mesure de prévenir a_2 mais également a_1 car la position p de la pomme appartient à sa portée de perception. L'agent a_3, par contre, n'est pas informé de ce changement d'état de l'environnement.

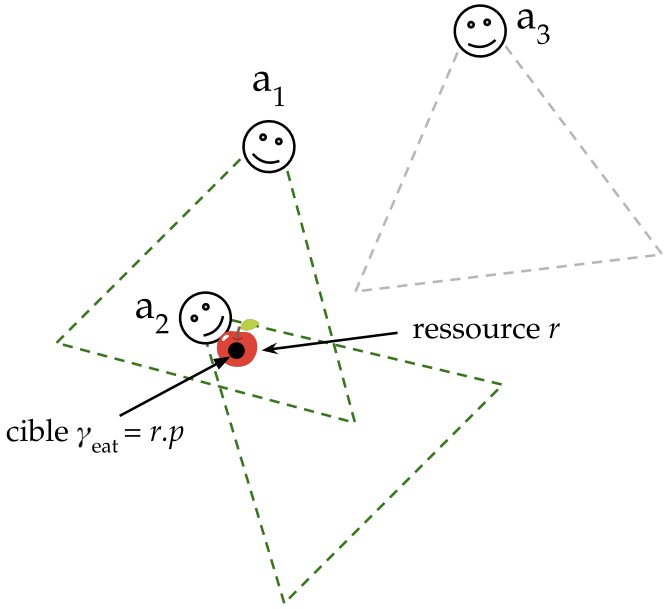


Figure 3.4 Illustration du calcul des agents concernés par la validation d'une influence grâce à la portée de perception des agents.

Fonction de sortie de l'environnement physique

Les percepts des agents, que nous avons noté Pa jusqu'à présent, sont envoyés par l'environnement à travers la fonction de sortie. L'ensemble des percepts est constitué d'une partie de l'ensemble des entités et d'une partie de l'ensemble des influences :

$$Pa_\varphi = \wp(\Xi) \times \wp(\Gamma)$$

Le listing 3.2 décrit sous forme de pseudo-code la fonction de sortie d'un environnement physique. Son rôle est d'envoyer les percepts des agents suite à un changement d'état de l'environnement.

```
1 fonction λφ(phaseφ, σφ, Aφ, (Ξφ, θφ), Γφ, reqbχ,φ) faire
2   si phaseφ = perception alors
```

```
3     // pour chaque influence ayant été validée
4     pour chaque γ_i ∈ Γ où γ_i.phase = satisfied faire
5       // pour chaque agent concerné par l'influence
6       pour chaque a ∈ {j ∈ A_φ | γ_i.p ∈ β_{j,φ}.ψ} faire
7         // envoyer sur le port de sortie dédié à l'agent : son corps, son
              voisinage, les ressources et les influences situées dans son rayon
              de perception
8         Y_a ← ν_φ(a) ∪ βa, φ ∪ {r | r ∈ cont_φ(βa, φ.ψ), r ∈ R}) × {γ ∈ Γ | γ.p ∈ β_{a,φ}.ψ}
9       fin pour
10    fin pour
11    // pour chaque influence n'ayant pas été validée
12    pour chaque γ_i ∈ Γ où γ_i.phase = rejected faire
13      // envoyer les percepts à l'agent ayant produit l'influence
14      Y_{γ_i.a} ← ν_φ(γ_i.a) ∪ βγ_i.a, φ ∪ {r | r ∈ cont_φ(βγ_i.a, φ.ψ), r ∈ R}) × {γ ∈ Γ | γ.p ∈ β_{γ_i.a,φ}.ψ}

15    fin pour
16  fin si
17 fin
```

Listing 3.2 Envoi des percepts suite à un changement d'état de l'environnement physique.

Notons que le comportement que nous venons de définir est sujet à la même remarque que pour le calcul du voisinage (*cf.* section 2.3.2). La structure d'une influence telle que nous l'avons définie cible une position dans l'espace. Si celle-ci cible plutôt une partie de l'espace et que nous la notons ψ_γ (variable potentiellement définie au sein de $\theta_{a,\varphi}$), alors les agents concernés par cette influence sont ceux dont l'intersection entre leur portée de perception et l'espace occupé par l'influence est non vide ($\psi_\gamma \cap \psi_{a,e} \neq \varnothing$).

Fonction de sortie de l'environnement social

Au sein d'un environnement social, les agents communiquent les uns avec les autres de manière directe, à travers des couplages PDEVS. Les messages directs ne transitent pas par l'environnement, la perception s'effectue à travers la mécanique du formalisme. Les agents peuvent cependant réaliser des actions au sein de l'environnement social à travers des influences. Par exemple, un agent peut produire une influence consistant à demander à jouer un nouveau rôle dans un groupe. Ainsi, de la même manière qu'un environnement physique, l'environnement social doit envoyer des percepts aux agents lorsqu'un changement d'état les concernent.

Chaque influence cible une position sociale particulière, c'est-à-dire un rôle. Les agents qui jouent un rôle « atteignable » depuis le rôle ciblé sont donc concernés par le changement d'état. Pour reprendre notre exemple, si un nouveau rôle est affecté à un agent pendant la phase de réaction de l'environnement social (l'influence représentant la demande a été validée, les nouveaux couplages directs ont été ajoutés par le modèle exécutif), il est alors nécessaire d'envoyer un percept aux agents qui peuvent interagir avec ce rôle, afin de les prévenir de l'arrivée d'un nouvel interlocuteur. L'ensemble des percepts pour

l'environnement social est défini par :

$$Pa_o = \wp(\beta) \times \wp(\Gamma)$$

Le comportement associé à la fonction de sortie λ_o de l'environnement est présenté sous forme de pseudo-code (listing 3.3) :

```
1  fonction λo(phaseo,σo,Ao,(Go,{∋g},Edgeso,βo,θo),Γo,req^b_{χ,o}) faire
2    si phaseo = perception alors
3      // pour chaque influence ayant été validée
4      pour chaque γi ∈ Γ où γi.phase = satisfied faire
5        // pour chaque agent concerné par l'influence
6        pour chaque a ∈ {j ∈ Ao | γi.p ∈ ⋃_{rj∈roleso(j)} β_{(j,rj),o}.ψ} faire
7          // envoyer sur le port de sortie dédié à l'agent : son corps, son
               voisinage et les influences situées dans son rayon de perception
8          Ya ← νo(a) ∪ βa,o × {γ ∈ Γ | γ.p ∈ ⋃_{ra∈roleso(a)} β_{(a,ra),o}.ψ}
9        fin pour
10     fin pour
11     // pour chaque influence n'ayant pas été validée
12     pour chaque γi ∈ Γ où γi.phase = rejected faire
13       // envoyer les percepts à l'agent ayant produit l'influence
14       Y_{γi.a} ← νo(γi.a) ∪ βγi.a,o × {γ ∈ Γ | γ.p ∈ ⋃_{r_{γi.a}∈roleso(γi.a)} β_{(γi.a,r_{γi.a}),o}.ψ}
15     fin pour
16   fin si
17 fin
```

Listing 3.3 Envoi des percepts suite à un changement d'état de l'environnement social.

Le comportement de la fonction λ est maintenant défini pour l'environnement physique et pour l'environnement social. Celle-ci est en charge d'envoyer les percepts aux agents concernés par une influence. Cette approche permet de s'assurer que l'ensemble des agents sont en mesure de percevoir tout changement d'état de l'environnement (physique ou social) sans que le modélisateur n'anticipe l'ensemble des perceptions possibles. La possibilité de généraliser le calcul des percepts repose entièrement sur l'idée qu'une action cible une position explicite, qu'elle soit spatiale ou sociale. Nous soulignons cependant qu'en pratique, cette approche peut entraîner un nombre important de messages entre l'environnement et le système cognitif des agents. Ce nombre dépend directement du nombre d'agents plongés dans l'environnement, des influences qu'ils produisent et de leur portée de perception. Cette approche comporte un deuxième inconvénient, celui d'envoyer systématiquement des messages à des agents qui ne sont pas forcément intéressés par la perception d'un certain type de ressources.

Le comportement de la fonction λ ayant été défini pour l'environnement physique et pour l'environnement social, l'ensemble des fonctions formant la dynamique du modèle d'environnement sont définies. La gestion des couplages directs par l'environnement social n'a en revanche pas été définie, celle-ci reposant sur des influences spécifiques et sur la

définition de la fonction de réaction de l'environnement. La sous-section suivante s'attache à proposer un comportement générique complet pour l'environnement social.

2.4.3 *Interaction directe*

La gestion des couplages entre les systèmes cognitifs est demandée au modèle exécutif par l'environnement social en fonction du rôle joué par les agents dans les différents groupes. L'arrivée ou le départ d'un agent dans un groupe ou un changement de rôle entraînent automatiquement une modification des couplages. Ces différentes demandes s'effectuent à travers l'envoi d'influences à l'environnement social. C'est à travers la fonction de réaction que la demande de changement de couplage est effectuée, si l'influence est considérée valide. Pour cela, la fonction *reac*, qui effectue une transition d'état de l'environnement, doit remplir le sac req^b_χ de requêtes destinées au modèle exécutif. Rappelons que l'ensemble des requêtes *Req* a été défini section 2.1.2. Bien que la fonction de réaction soit laissée libre à la définition du modélisateur, nous donnons ici un comportement possible pour l'environnement social à des fins pédagogiques.

Afin de donner un exemple concret, nous définissons dans cette section deux types d'influences : les influences de demande de rôle ($\Gamma_{reqrole}$) et les influences demandant à se démettre d'un rôle ($\Gamma_{resignrole}$), définies par :

$$\Gamma_{reqrole} = \{(phase, n, p, a, (g))\}$$

où $n \in \{reqrole\}$, et la variable θ est définie par le singleton (g) où $g \in G$ représente le groupe pour lequel l'agent souhaite devenir un membre. La position ciblée par l'influence représente le rôle $p \in R_g \subseteq P$ que l'agent a souhaite jouer au sein du groupe g.

$$\Gamma_{resignrole} = \{(phase, n, p, a, (g))\}$$

où $n \in \{resignrole\}$, θ est également définie par le singleton (g) où $g \in G$ représente le groupe dans lequel l'agent a souhaite se démettre du rôle $r \in R_g$.

Nous donnons dans le listing 3.4 un exemple de la façon dont ces influences peuvent être traitées par l'environnement social $o \in D_E$ à travers la fonction de réaction $reac_o$ sous forme de pseudo-code. Dans un souci de clarté, nous ne définissons pas de règles environnementales particulières qui viendraient complexifier le comportement de la fonction $reac_o$. Nous supposons ici que l'environnement social est ouvert et qu'il n'y a pas de contraintes particulières sur les rôles et les groupes. Ainsi n'importe quel agent peut devenir membre d'un groupe (membres illimité), et les rôles peuvent être joués par un nombre illimité d'agents. En conséquence, toutes les influences sont considérées comme valides.

```
1 fonction reac_o((s_o, e_o), Γ'_o) :
2   // traitement des demandes de rôles
3   pour chaque γ_i ∈ Γ'_o où γ_i.n = reqrole faire
4   // création du nouveau corps de l'agent
5     β_(γ_i.a,γ_i.p),o ← (γ_i.p, {r_j ∈ P | j ∈ A, dist_o(γ_i.p, r_j) = 1}, ∅)
```

```
6       s_o.β ← s_o.β ∪ β_(γ_i.a,γ_i.p),o
7
8       // création des couplages pour la communication directe
9       // pour les agents qui peuvent atteindre l'agent γ_i.a sur le rôle demandé
10      pour chaque a ∈ {j ∈ A \ {γ_i.a} | γ_i.p ∈ ⋃_{r∈roles(j)} β_(j,r,o).ψ} faire
11        // si aucun couplage n'existe entre a et γ_i.a
12        si (roles_o(γ_i.a) \ {γ_i.p}) ∩ (⋃_{r∈roles_o(a)} β_(a,r),o.ψ) = ∅ alors
13          // demande d'ajout d'un couplage entre a et γ_i.a
14          s_o.req^b_χ ← s_o.req^b_χ ∪ (add_coupl, ((a, γ_i.a), (γ_i.a, a)))
15        fin si
16      fin pour
17      // pour les agents que l'agent γ_i.a peut atteindre depuis le rôle demandé
18      pour chaque a ∈ {j ∈ A \ {a} | roles_o(a) ∩ β_(γ_i.a,γ_i.p),o.ψ ≠ ∅} faire
19        // si aucun couplage n'existe entre γ_i.a et a
20        si (⋃_{r∈roles(γ_i.a)\{γ_i.p}} β_(γ_i.a,r),o.ψ) ∩ roles(a) = ∅ alors
21          // demande d'ajout d'un couplage entre γ_i.a et a
22          s_o.req^b_χ ← s_o.req^b_χ ∪ (add_coupl, ((γ_i.a, a), (a, γ_i.a)))
23        fin si
24      fin pour
25
26      // marquer l'influence comme validée
27      γ_i.phase ← satisfied
28    fin pour
29
30    // traitement des démissions
31    pour chaque γ_i ∈ Γ'_o où γ_i.n = resignrole faire
32      // suppression du corps de l'agent correspondant à ce rôle
33      s_o.β ← s_o.β \ β_(γ_i.a,γ_i.p),o
34
35      // suppression des couplages pour la communication directe
36      // pour les agents qui peuvent atteindre l'agent γ_i.a sur le rôle à
            supprimer
37      pour chaque a ∈ {j ∈ A \ {γ_i.a} | γ_i.p ∈ ⋃_{r∈roles(j)} β_(j,r,o).ψ} faire
38        // si le couplage entre a et γ_i.a peut être supprimé
39        si (roles_o(γ_i.a) \ {γ_i.p}) ∩ (⋃_{r∈roles_o(a)} β_(a,r),o.ψ) = ∅ alors
40          // demande de suppression du couplage entre a et γ_i.a
41          s_o.req^b_χ ← s_o.req^b_χ ∪ (del_coupl, ((a, γ_i.a), (γ_i.a, a)))
42        fin si
43      fin pour
44      // pour les agents que l'agent γ_i.a peut atteindre depuis le rôle à
            supprimer
45      pour chaque a ∈ {j ∈ A \ {a} | roles_o(a) ∩ β_(γ_i.a,γ_i.p),o.ψ ≠ ∅} faire
46        // si le couplage entre γ_i.a et a peut être supprimé
47        si (⋃_{r∈roles(γ_i.a)\{γ_i.p}} β_(γ_i.a,r),o.ψ) ∩ roles(a) = ∅ alors
```

```
48          // demande de suppression du couplage entre γ_i.a et a
49          s_o.req^b_χ ← s_o.req^b_χ ∪ (add_coupl, ((γ_i.a, a), (a, γ_i.a)))
50        fin si
51      fin pour
52
53      // marquer l'influence comme validée
54      γ_i.phase ← satisfied
55    fin pour
56  fin
```

Listing 3.4 Exemple de comportement possible pour la fonction de réaction d'un environnement social.

Dans l'objectif d'en faciliter la conception, nous avons proposé dans cette section une spécification formelle et générique des différentes entités d'un SMA. Nous avons fait en sorte de fournir un comportement générique au modèle exécutif et aux modèles d'environnement afin de généraliser la gestion des interactions et de la dynamique structurelle. L'étendue des définitions que nous avons proposées rend cependant difficile la compréhension globale de l'approche. Pour cette raison, nous présentons dans la section suivante une vue d'ensemble de l'approche, dans le but de faciliter la compréhension des concepts et de proposer au modélisateur une méthode de spécification des modèles basés sur DPDEMAS.

3 *Vue d'ensemble de l'approche*

La spécification DPDEMAS que nous avons proposée dans ce chapitre permet de faciliter la spécification formelle d'un modèle SMA. Les différentes entités que nous avons formalisées constituent un cadre permettant de structurer le modèle lors de sa conception. Tel qu'il a été défini, ce cadre a également permis de définir le comportement générique de l'environnement. Ceci permet non seulement de faciliter la construction de modèles d'environnements adaptés à différentes topologies, mais également de définir des influences réutilisables pour des actions récurrentes, comme le déplacement d'un agent. DPDEMAS cherche à guider le modélisateur dans la spécification d'un modèle multi-agents, et permet au modélisateur de concentrer ses efforts de modélisation sur le comportement des agents et de leurs interactions. En prescrivant une manière de concevoir et de définir un SMA, notre spécification constitue une démarche couvrant une étape du processus de modélisation et de simulation. Comme nous l'avons précisé en introduction de ce chapitre, nous ne proposons pas de méthodologie permettant de couvrir l'ensemble des étapes du processus, mais bien un cadre générique permettant de guider le modélisateur dans l'étape de spécification formelle du modèle. Cette démarche pourra éventuellement être intégrée dans une méthodologie plus complète qui reste à définir. En attendant, celle-ci peut tout à fait prendre place dans les éléments méthodologiques donnés par Duboz et al. (2012), que nous avons présentés section I.4.3.

Afin de clarifier la manière d'exécuter l'activité de spécification formelle du modèle SMA,

nous donnons dans un premier temps une vue d'ensemble de la spécification DPDEMAS avant de définir une méthode à proprement parler. Nous proposons dans une première section une représentation semi-formelle du métamodèle correspondant à la spécification DPDEMAS. Afin d'ouvrir d'autres perspectives, nous proposons une autre représentation d'un système multi-agents DPDEMAS à l'aide du formalisme SES (*System Entity Structure*). Enfin, nous présentons une méthode basée sur DPDEMAS pour la spécification d'un système multi-agent.

3.1 *Métamodèle SMA*

Afin de donner une vision globale de notre approche, nous donnons dans cette section une représentation semi-formelle du métamodèle que nous avons adopté. Par rapport à un langage semi-formel, un langage formel tel que nous l'avons utilisé pour DPDEMAS nécessite un niveau d'expertise important. À l'inverse, un langage semi-formel est plus intuitif mais dispose d'une part d'ambiguïté puisque sa sémantique est sujette à interprétation. En complément de la spécification formelle, nous utilisons le langage UML afin de donner une représentation graphique du métamodèle adopté. Celui-ci permet d'appréhender plus facilement les concepts manipulés et leurs relations.

La figure 3.5 donne un diagramme UML simplifié du métamodèle et permet de visualiser les relations entre agents, corps, esprits, environnements, positions, ou encore influences. Nous faisons également apparaître une partie du métamodèle du formalisme PDEVS sous la forme d'un paquetage dédié afin d'en représenter les concepts. Par exemple, le système multi-agents DPDEMAS est un modèle DSDE ; le système cognitif est un `Model`, tout comme l'environnement. Les types de messages qui transitent entre les ports des différents modèles sont représentés sous forme de dépendance. Par exemple, l'environnement envoie des requêtes dynamiques, et le modèle exécutif DPDEMAS les traite. Ces relations de dépendance représentent indirectement les différents types de couplage entre les modèles PDEVS. Ainsi, le système cognitif envoie et reçoit des messages directs ; envoie des influences qui sont traitées par l'environnement.

Les autres concepts représentés correspondent aux différentes entités qui composent l'état des modèles d'environnements ou l'état des systèmes cognitifs, tels que nous les avons défini dans DPDEMAS.

En ce qui concerne l'environnement, il réalise une topologie à travers la fonction de distance, qui place les positions les unes par rapport aux autres. Les concepts qui forment son état sont représentés par des relations de composition ou d'association : les influences pour lesquelles l'environnement a réagi, les agents qui y sont plongés et les corps des agents. Comme l'environnement, l'agent est un concept central. Chaque système cognitif est associé à un agent, et chaque corps appartient à un agent. C'est à travers le corps que l'agent est plongé dans l'environnement. Deux spécialisations de l'environnement permettent de faire apparaître le concept d'environnement social et d'environnement physique. Le premier dispose d'une relation de composition avec les groupes qui le composent, où chaque groupe est composé d'un ensemble de rôles que les agents peuvent jouer. L'environnement physique,

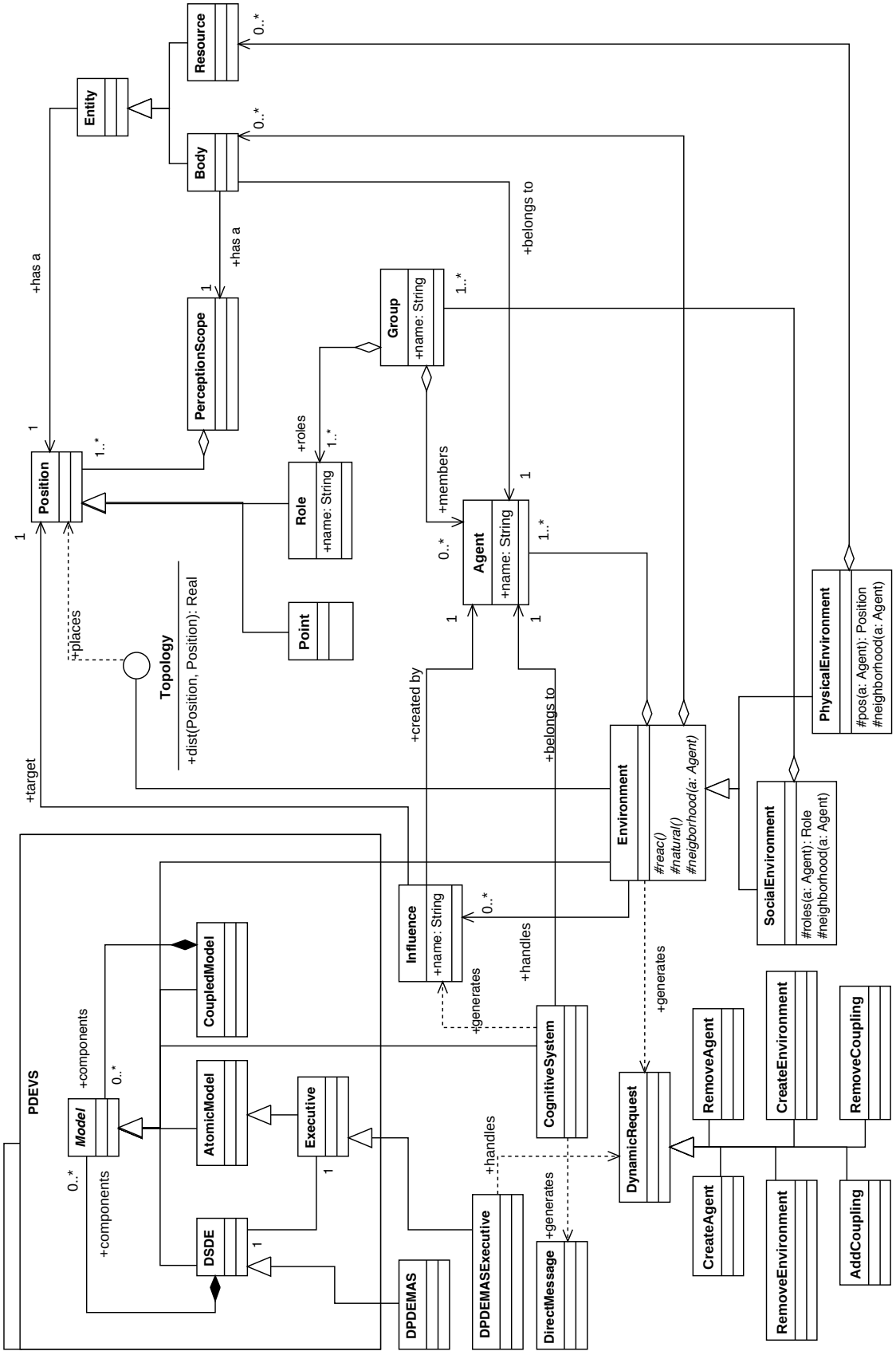


Figure 3.5 Métamodèle UML du système multi-agents adopté par la spécification DPDEMAS.

lui, est également composé de ressources.

Ce métamodèle est un récapitulatif des concepts introduits par la spécification, dans une représentation simplifiée. Il fournit une description de l'organisation d'un système multi-agents indépendamment d'une solution spécifique. S'il permet de synthétiser les concepts et leurs relations à un niveau d'abstraction élevé, il ne permet pas de visualiser la structure du modèle SMA lui-même. Nous proposons dans la sous-section suivante une autre représentation graphique de DPDEMAS permettant de visualiser la structure hiérarchique du SMA mais également les différentes variantes possibles du système avec les différentes spécialisations que nous avons suggérées.

3.2 *System Entity Structure*

Etant donné que nous encourageons le modélisateur à se diriger vers la multi-modélisation, c'est-à-dire à utiliser les extensions du formalisme PDEVS pour représenter et exprimer au mieux les SMA selon leurs propriétés, il existe un nombre important de variantes d'un modèle spécifié avec DPDEMAS.

Le formalisme SES, pour *System Entity Structure* (Zeigler et al., 2013) permet de matérialiser les différentes variantes possibles des SMA sous la forme d'un arbre, qui permet de faire apparaître les composants du système à modéliser de manière hiérarchique et de représenter des relations de spécialisation ou d'aspect multiple. Les différents éléments d'un SES sont identifiés à partir du vocabulaire suivant :

Entité. Une entité représente un système qui peut soit être identifié de manière indépendante, soit être décomposé en suivant les relations que nous décrivons ci-après. Des variables peuvent être associées à une entité.

Aspect. Un aspect représente une relation de décomposition hiérarchique. Les enfants d'un aspect sont des entités qui composent une entité parente.

Spécialisation. La relation spécialisation permet de classifier différentes entités et d'exprimer des choix alternatifs pour les composants du système. Les enfants d'une relation de spécialisation sont des entités représentant des variantes de leur parent.

Multi-aspect. Un multi-aspect est une relation de décomposition hiérarchique. Les enfants d'un multi-aspect sont plusieurs entités du même type qui composent l'entité parente.

En donnant une représentation SES de la spécification DPDEMAS, il est possible de représenter un système multi-agents dans toute sa variabilité et d'en déduire l'ensemble des systèmes multi-agents qu'il est possible de représenter avec DPDEMAS. La figure 3.6 donne la représentation SES d'un système multi-agents DPDEMAS. L'ensemble des entités représentent des modèle PDEVS (atomique, couplé, dynamique ou issu d'une extension) qui composent le SMA. Les relations de spécialisation expriment les choix qu'il est possible d'effectuer en terme de type d'environnement ou de type d'agent. Par exemple, un environnement peut être social ou physique. Un environnement physique peut être discret ou continu, et ainsi de suite. Les relations d'aspects multiples indiquent au modélisateur qu'il doit établir un nombre d'instances particulières d'une entité. Par exemple, le système multi-agents étant

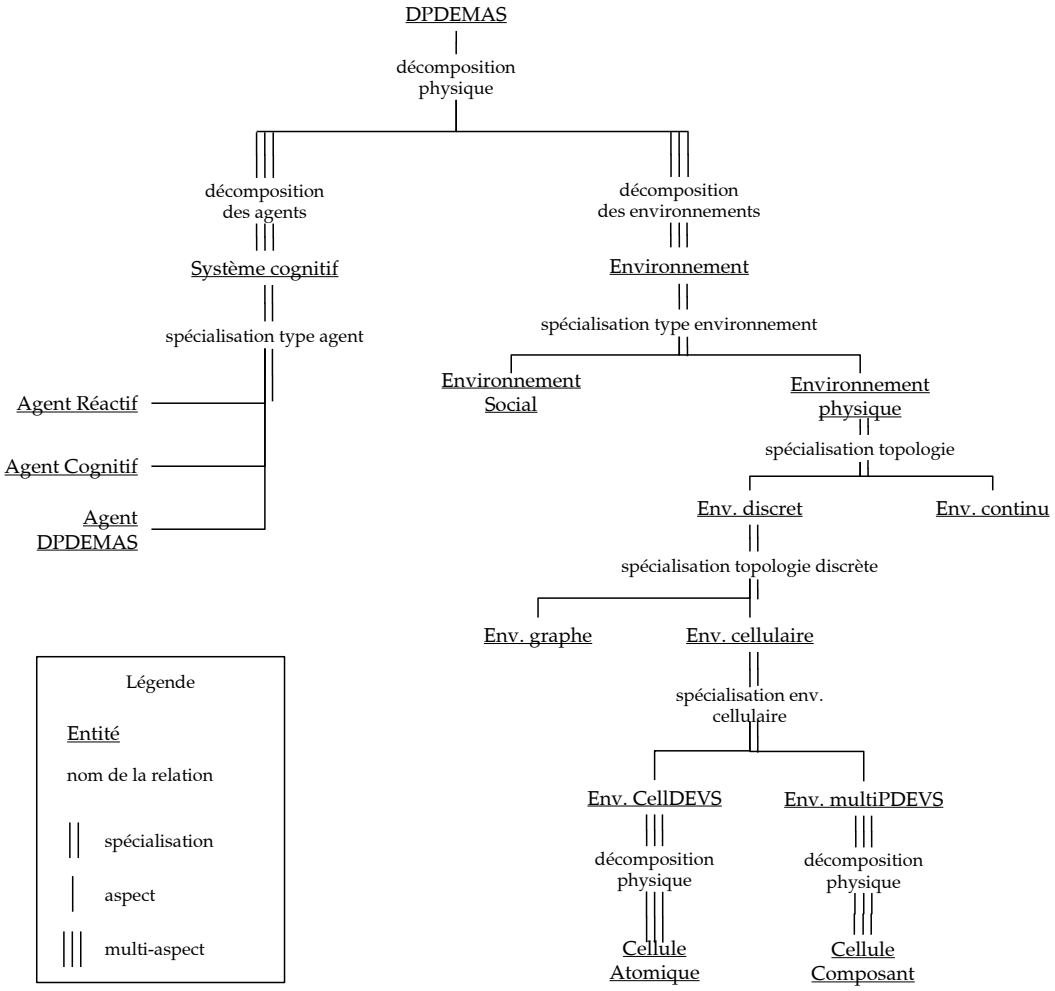


Figure 3.6 Représentation de l'arbre *System Entity Structure* d'un SMA spécifié à l'aide de DPDEMAS.

composé d'agents et d'environnements, les relations d'aspects multiples permettent d'établir le nombre d'agents et le nombre d'environnements qui composent le SMA.

Le processus de *pruning* (élagage) d'un SES correspond à la construction d'une structure adaptée aux objectifs de modélisation. Le *pruning* permet de générer une instance concrète d'un SES qui forme un nouvel arbre, appelé *pruned entity structure* (PES). Celui-ci représente une variante unique du système, toutes les variabilités du SES ayant été résolues : un choix est effectué pour chaque relation de spécialisation et toutes les relations d'aspect multiples sont développées. Si le modélisateur suivant notre spécification souhaite modéliser un système multi-agents comprenant 4 agents réactifs tous plongés dans le même environnement physique dont la topologie discrète est formée par un espace 2D de 4x4 cellules, il peut effectuer un *pruning* du SES représenté en figure 3.6. Le PES qui résulte du pruning pour les objectifs de simulation que nous avons énumérés est donné en figure 3.7. Au delà de donner une vue d'ensemble de la spécification, il est possible de représenter un SES dans une structure informatique (Han et al., 2010). Cette possibilité offre donc des perspectives

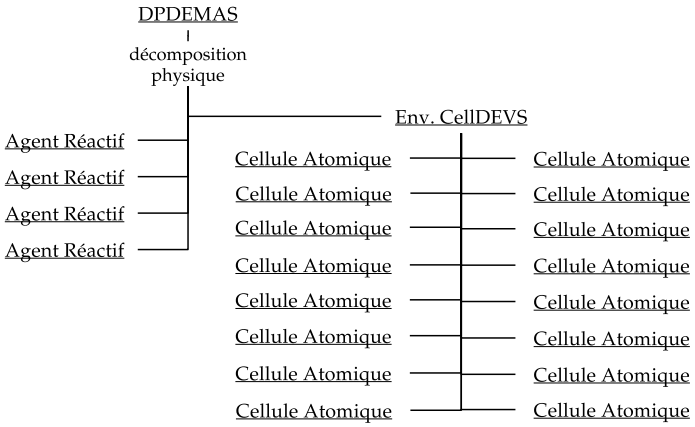


Figure 3.7 Exemple de *pruning* du SES de DPDEMAS sous forme d'arbre PES formant un SMA concret.

d'automatisation du processus de pruning.

Notons que que nous aurions également pu décrire la variabilité d'un système multi-agents DPDEMAS à l'aide d'un *feature model* (Kang et al., 1990).

3.3 *Méthode pour la spécification d'un SMA*

Nous proposons dans cette section de définir une méthode permettant de guider le modélisateur dans la spécification d'un modèle de système multi-agent. Une méthode prescrit une manière d'exécuter un type d'activité dans un processus donné afin de produire correctement un résultat spécifique à partir d'une entrée spécifique (Cernuzzi et al., 2005). Le processus dont il est question dans notre cas est celui de la modélisation et de la simulation. Comme nous l'avons évoqué dans le chapitre I, le processus de M&S peut être considéré comme un processus itératif dont l'objectif est de faire évoluer un modèle initial jusqu'à obtenir un modèle en adéquation avec les objectifs de simulation. Cette idée d'itération est également présente dans le schéma (*cf.* figure 1.13) proposé par Sargent (2005). Or, comme le soulignent Kubera et al. (2011), il n'existe pas de consensus quant à la définition des activités du processus de M&S.

En s'inspirant de plusieurs définitions proposées dans la littérature, notamment celle de Shannon (1998), ces derniers proposent le processus illustré par la figure 3.8. Les huit activités qu'il comprend peuvent être synthétisées de la manière suivante :

1. **Définition du problème** : le processus débute par l'identification des objectifs de simulation.
2. **Définition du système** : d'après le système réel et son fonctionnement, identification informelle des éléments permettant de répondre aux objectifs.
3. **Formulation du modèle conceptuel** : spécification formelle du modèle.
4. **Conception expérimentale préliminaire** : la nature des entrées/sorties, les observations

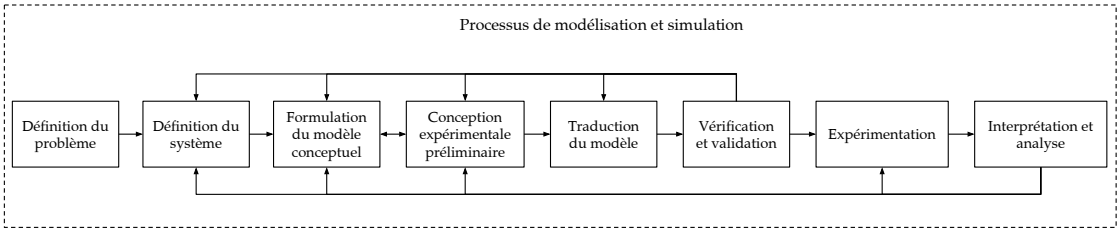


Figure 3.8 Schéma du processus de modélisation et de simulation proposé par Kubera et al. (2011).

et les expérimentations menées sur le système réel permettent éventuellement d'affiner le modèle conceptuel.

5. **Traduction du modèle** : le modèle conceptuel est traduit en programme informatique.
6. **Vérification et validation** : si le modèle est considéré comme correct, la phase d'expérimentation peut débuter. Dans le cas contraire, une nouvelle itération peut amener à revoir les objectifs de simulation, à affiner le modèle conceptuel, à collecter de nouvelles données expérimentales ou à revoir l'implémentation du modèle conceptuel.
7. **Expérimentation** : exécution des simulations selon un plan d'expérience, afin de générer les données attendues ou de conduire une analyse de sensibilité.
8. **Interprétation et analyse** : analyse et interprétation des résultats produits par les simulations. Cette activité peut amener à remettre en question l'expérimentation, les hypothèses établies ou encore le modèle conceptuel et ainsi, conduire à une nouvelle itération.

Dans le contexte de nos travaux, le résultat attendu de l'activité de formulation du modèle conceptuel est d'obtenir une spécification formelle d'un modèle conceptuel basé sur le paradigme agent. Comme illustré par la figure 3.8, le prérequis de cette activité est de disposer de la définition du système. Bien que notre objectif ne soit pas de fournir une méthodologie couvrant toutes les activités du processus, nous dressons tout de même une liste des différents éléments pouvant apparaître dans la définition informelle d'un système multi-agents. Nous nous concentrons donc dans cette section sur deux activités du processus de modélisation et de simulation : la définition du système et la formulation du modèle conceptuel. Pour chacune de ces activités, nous proposons une méthode associée permettant de couvrir une partie du processus de M&S. Nous dédions une section à chacune d'elles.

3.3.1 *Définition du système multi-agents*

Le prérequis pour la spécification formelle du modèle conceptuel est de recueillir la définition du système. Etant donné que nous nous focalisons sur les systèmes multi-agents, nous proposons d'identifier un certain nombre de points permettant d'établir la définition d'un SMA. Nous donnons différentes étapes à considérer, toutes inspirées du protocole ODD (Grimm et al., 2010) et de la méthodologie IODA (Kubera et al., 2011). Ces étapes ne constituent pas une référence absolue, bien que nous invitons l'expert du domaine à considérer chacun d'entre eux. De la même manière, celui-ci peut tout à fait compléter la définition du SMA de points additionnels qui lui semblent appropriés.

Dans le but de construire un modèle SMA formel compatible avec la spécification DPDEMAS, nous proposons la méthode suivante pour guider le modélisateur ou l'expert du domaine à établir une définition du système :

1. **Représentation du temps**. De quelle manière est-il représenté ? Déterminer l'unité de temps et les différentes échelles. Quel intervalle de temps est ciblé pour une simulation ?
2. **Déterminer les environnements**. Dans quel(s) type d'environnement(s) les agents évoluent-ils ? Pour chaque environnement, si celui-ci est :
   a. **Physique**. Déterminer sa topologie ; est-elle discrète ou continue ? Quelle est l'unité spatiale ? Quelle est la taille de l'environnement ciblée ? Lister les ressources pouvant exister au sein de l'environnement et leurs variables d'état.
   b. **Social**. Identifier les différents groupes d'agents. Pour chacun d'eux, établir une liste de rôles pouvant être joués. Donner les différentes contraintes entre les rôles s'il y en a. Cette étape peut être complétée par un diagramme similaire à celui présenté section I.3.4.

   Déterminer pour chaque environnement les différentes variables. Par exemple, la température, la pluviométrie ou la courantologie pour un environnement physique. Pour chacune d'elle, déterminer si elle est soumise à une dynamique environnementale.
3. **Déterminer les familles d'agents**. Identifier les différents types d'agents prenant part à la simulation ainsi que les variables d'état qui les caractérisent, et si possible leurs unités, dans quelle mesure elles peuvent évoluer et si elles sont observables par d'autres agents. Déterminer les relations de spécialisation entre ces familles si il y en a (cette étape peut être complétée par un diagramme). Si plusieurs environnements ont été identifiés déterminer dans quel(s) environnement(s) les différentes familles d'agents sont plongées.
4. **Déterminer les interactions**. Lister les différentes interactions qui peuvent prendre place au sein de chaque environnement. Pour chaque type d'interaction, donner les familles d'agents sources et les familles d'agents cibles. Cette étape peut être complétée par une matrice d'interaction (Kubera et al., 2011), qui prend la forme d'un tableau. Donner également les conditions préalables à leur apparition. Si l'interaction est de type directe, préciser la nature des messages envoyés. Si l'interaction est de type indirecte, préciser l'effet escompté sur l'environnement.
5. **Etablir les règles environnementales**.
   a. Pour chaque interaction indirecte, établir les conditions qui permettent de satisfaire l'interaction. Par exemple, l'agent est distant de la cible de x unités, et la variable environnementale y est inférieure à un seuil.
   b. Déterminer les situations de compétition et la façon de les résoudre. Par exemple, si deux agents a et b souhaitent accéder à un plat de nourriture simultanément et qu'il n'y a pas suffisamment de reste pour satisfaire les deux interactions, a a une probabilité de 0,5 de manger.
   c. Déterminer l'ordre dans lequel la dynamique environnementale et les différentes interactions sont traitées par l'environnement pour changer d'état.

6. **Perception**. Pour chaque famille d'agent, déterminer pour chaque environnement dans lequel il est susceptible d'être plongé ce qu'il est en mesure de percevoir en terme de variables environnementales, de ressources ou de variables d'autres agents. Définir également la portée de perception de l'agent.
7. **Comportement**. Pour chaque famille d'agent, décrire le comportement de l'agent. De la même manière, pour chaque environnement doté d'une dynamique environnementale, décrire la dynamique endogène. Exprimer les unités temporelles (périodicité, échelle, ...).
8. **Description du SMA au niveau macro**. Au niveau du système multi-agent, définir :
   a. **L'état initial**. Définir pour l'ensemble des environnements, des agents et des ressources, l'état initial du système multi-agent.
   b. **Les sorties**. Définir les sorties permettant d'observer, d'analyser ou de tester des variables au niveau du SMA. Par exemple, une population d'agent. Préciser la façon dont ces variables doivent être extraites des agents ou des environnements.
   c. **Les entrées**. Définir les différents paramètres d'entrées avec leur unité, leur source et les entités à qui elles sont destinées.

Le suivi de ces différentes étapes permet d'établir une définition informelle du système multi-agents à modéliser. Cette définition forme le prérequis indispensable qui va permettre d'établir une spécification formelle du SMA. La méthode que nous proposons pour formuler le modèle conceptuel dans la section qui suit utilise le résultat de la méthode que nous avons présentée dans cette section.

3.3.2 *Formulation du modèle conceptuel*

L'étape de formulation du modèle conceptuel s'inscrit dans le processus de modélisation et de simulation (*cf.* figure 3.8). Celle-ci consiste à partir de la définition du système, à spécifier formellement le modèle du système. La seconde méthode que nous proposons s'appuie sur la spécification DPDEMAS que nous avons présentée section 2 et constitue en définitive, un guide d'utilisation de la spécification.

Afin de définir le modèle conceptuel d'un système multi-agents, nous préconisons la méthode suivante, composée de neuf étapes :

1. **Définition des identifiants**.
   a. Pour chaque environnement physique ou social ayant été identifié, définir l'ensemble des identifiants des environnements D_E du modèle DPDEMAS. Par exemple, si le SMA est composé d'un seul environnement physique identifié par φ, alors $D_E = \{\varphi\}$.
   b. De la même manière, définir un identifiant pour chaque agent dans l'ensemble des identifiants des agents D_{Ag}. Définir un sous ensemble pour chaque famille d'agent. Par exemple, dans un modèle proie/prédateur, si deux familles d'agents sont identifiées (les proies et les prédateurs), alors

$$D_{Ag} = Preys \cup Predators$$

   où $Preys = \{prey_i \mid 1 \leq i \leq m\}$ et $Predators = \{pred_j \mid 1 \leq j \leq n\}$.

2. **Choix des modèles d'environnement**. Pour chaque environnement ayant été identifié, si celui-ci est :
   a. **Physique**. En fonction de la topologie exigée par la définition du SMA, choisir dans la bibliothèque d'environnements un modèle adapté à la topologie. Si aucun modèle d'environnement ne permet de satisfaire les besoins :
      i. déterminer le type de modèle PDEVS (atomique, couplé ou issu d'une extension) le plus adapté pour représenter la topologie ;
      ii. définir l'environnement adapté à la topologie d'après la spécification de l'environnement physique générique (*cf.* section 2.3.1 et 2.3.2).

      D'après la définition du SMA, définir les ressources possibles dans cet environnement à l'aide de Ξ ainsi que leurs variables d'état à l'aide de la variable θ.
   b. **Social**. En fonction des groupes et des rôles identifiés dans la définition du SMA et d'après la spécification de l'environnement social (*cf.* section 2.3.3) :
      i. définir l'ensemble des identifiants des groupes G. Pour chacun d'eux, définir l'ensemble des rôles possibles à l'aide de l'ensemble R_g.
      ii. définir les interactions possibles entre les rôles à l'aide de $Edges$;
3. **Définition des corps**. Pour chaque famille d'agents et pour chaque environnement dans lequel ils sont susceptibles d'être plongés, définir le corps des agents d'après la définition du système en précisant :
   a. la portée de perception de l'agent ;
   b. les différentes variables d'états de l'agent correspondant qui ont été identifiées comme observables dans la définition du SMA.
4. **Définition des influences**. Pour chaque interaction indirecte qui a été identifiée ainsi que pour chaque dynamique environnementale identifiée dans la définition du SMA, définir un sous ensemble de Γ pour chaque famille d'interaction en précisant les familles d'agents pouvant réaliser ce type d'influence ainsi que les différentes variables qui caractérisent l'influence. Pour reprendre notre exemple de SMA proie/prédateur, seuls les prédateurs peuvent manger les proies. Ainsi l'ensemble Γ_{eat}, qui représente l'ensemble des influences permettant aux proies de se nourrir peut être défini de la manière suivante :
   $$\Gamma_{eat} = \{(phase, n, p, a, (a_t))\}$$
   où $n \in \{eat\}$, $a \in Predators$, et la variable θ est définie par le singleton (a_t) où $a_t \in Preys$ représente l'identifiant de l'agent proie ciblé par l'agent prédateur a. Enfin, p représente la position de la proie.
5. **Définition des messages directs**. Si les agents sont plongés dans un environnement social et qu'ils prennent part à une communication directe, définir l'ensemble Msg.
6. **Définition du calcul de réaction**. Pour chaque environnement, définir la fonction $reac$ correspondante d'après la définition du SMA. Celle-ci doit suivre les règles environnementales qui ont été établies afin de déterminer si une influence émise par un agent ou par la dynamique environnementale est considérée comme valide en fonction de l'état courant de l'environnement. Lorsqu'une influence est considérée comme valide, la

fonction *reac* doit modifier l'état de l'environnement et éventuellement effectuer une demande de changement structurel auprès du modèle exécutif en suivant l'effet escompté qui a été déterminé pendant l'activité de définition du corps.

7. **Etat de l'environnement et dynamique**. Pour chaque environnement, définir les variables environnementales dans l'état de l'environnement. Si une dynamique est associée, définir la fonction *natural* associée à l'environnement permettant de produire l'influence associée. Utiliser la variable σ pour déterminer le temps d'activation de l'environnement.
8. **Définition des agents**. Pour chaque famille d'agent et en fonction du comportement décrit par la définition du SMA, déterminer le type de modèle PDEVS (atomique, couplé ou issu d'une extension) le plus adapté pour représenter l'agent. D'après l'interface préconisée pour un agent (*cf.* section 2.2), définir le modèle d'agent. Définir l'état de l'agent d'après les variables non observables identifiées pendant l'activité de définition du SMA. Enfin, définir le comportement de l'agent.
9. **Définition du SMA**.
   a. D'après les différents modèles d'agents et d'environnements retenus, appliquer l'opération de pruning de l'arbre SES de DPDEMAS afin d'obtenir l'arbre PES du SMA (*cf.* section 3.2).
   b. Définir l'interface d'entrées/sorties du SMA.
   c. Définir l'état initial de l'ensemble des entités du SMA (agents, corps, environnements, ressources).

Les deux méthodes que nous avons présentées dans cette section couvrent une partie du processus de M&S et permettent de faciliter la définition formelle d'un modèle conceptuel SMA. La méthode repose entièrement sur la spécification DPDEMAS, qui permet de faciliter la définition d'un SMA. Bien que nous ne proposons pas de méthode permettant de générer automatiquement le modèle en tant que programme informatique, nous verrons dans le prochain chapitre que la spécification DPDEMAS permet de faciliter l'étape de traduction du modèle du processus de M&S, notamment parce-qu'elle est basée sur le formalisme PDEVS.

4 *Résumé du chapitre*

Nous avons proposé dans ce chapitre une spécification formelle des SMA, qui s'appuie sur la théorie de la M&S proposée par Zeigler et al. (2000), et plus particulièrement sur le formalisme PDEVS et une partie de ses extensions. Comme nous l'avons indiqué dans le chapitre II, le formalisme PDEVS a pour particularité d'être compatible avec le principe d'Influence/Réaction, qui permet d'apporter une solution à la problématique d'ordonnancement des agents et à la simultanéité des actions. Les différentes propriétés de la spécification découlent de notre positionnement vis-à-vis des concepts du paradigme agent (*cf.* section I.5). La spécification permet de modéliser une dynamique structurelle (*e.g.* fluctuation de la population d'agents), de tirer partie de l'aspect hiérarchique de PDEVS en permettant de décomposer les modèles d'agents et d'environnements et de réaliser des SMA multi-niveaux. Elle permet également de concevoir une grande variété de modèles SMA, composés d'un ou plusieurs environnements physiques et/ou sociaux. La généricité des modèles d'environnements permet de faciliter la gestion des interactions et le calcul des percepts des

agents. Cette particularité permet également d'envisager la définition de plusieurs modèles d'environnements, adaptés à différentes topologies, en vue de leur réutilisabilité.

Afin de guider le modélisateur dans la définition d'un SMA basé sur notre spécification, nous proposons deux méthodes couvrant une partie du processus de M&S. La première permet de donner une définition du SMA ; la seconde utilise cette définition pour guider la formulation du modèle formel. L'intérêt de la spécification du modèle est double : elle forme une description non-ambigüe du modèle en vue de son partage et lui permet d'être traduite en modèle exécutable.

Chapitre IV

IMPLÉMENTATION D'UNE PLATEFORME DE SIMULATION

Sommaire

1 *Introduction*

Lors de la conception d'un modèle, une étape du processus de M&S est celui de la traduction du modèle (*cf.* figure 3.8). L'un des objectifs de nos travaux est de proposer un outil permettant la modélisation et la simulation de systèmes multi-agents. Nous avons jusqu'ici proposé une approche formelle basée sur le formalisme PDEVS, nommée DPDEMAS, associée à une méthode de spécification de modèles de systèmes multi-agents. Dans le but de faciliter l'étape de traduction du modèle SMA, nous proposons de développer une plateforme de modélisation et de simulation basée sur le formalisme PDEVS et sur nos travaux de formalisation des SMA. Baptisée *Quartz*, cette plateforme prend la forme d'un cadriciel facilitant l'implémentation de modèles PDEVS, de certaines de ses extensions et de modèles SMA basés sur DPDEMAS. Celle-ci permet de simuler ces modèles conformément au protocole de simulation défini par Zeigler et al. (2000).

Dans ce chapitre, nous présentons l'architecture de *Quartz*, afin de mieux situer cette plateforme vis-à-vis de celles existantes en présentant ses fonctionnalités et l'implémentation des extensions. Nous présentons ensuite les différentes méthodes d'optimisation que nous avons développées pour obtenir de meilleures performances. Enfin, nous donnons des détails d'implémentation concernant nos travaux de formalisation des systèmes multi-agents.

2 *Comparaison des simulateurs DEVS existants*

Un nombre important d'outils permettant la modélisation et la simulation de modèles DEVS ou PDEVS ont été développés, que ce soit à des fins académiques ou industrielles. Pour la plupart, ces outils sont associés à des domaines d'applications spécifiques et ont des objectifs en lien avec ces domaines d'application. En conséquence, les langages utilisés et les API [1] développées sont très hétérogènes. De plus, les fonctionnalités proposées sont également extrêmement variées : les outils peuvent être assez minimalistes, représentant de simples « noyau de simulation » ou au contraire, offrir de véritables environnements de modélisation intégrés couvrant plusieurs aspects du processus de M&S. Enfin, au vu du nombre de variantes du formalisme DEVS proposées dans la littérature, seule une partie de ces variantes sont réellement implémentées dans les environnements. Afin de donner une vue d'ensemble des plateformes existantes, différentes études comparatives ont été proposées pour éclaircir le choix du modélisateur (Franceschini et al., 2014a ; Van Tendeloo et Vangheluwe, 2016).

Lorsque les objectifs sont clairement indiqués, ou en fonction des différentes fonctionnalités supportées, il est possible d'identifier les outils les plus adaptés. Une bonne partie d'entre eux constituent des bibliothèques légères pouvant être considérées comme des cadriciels. C'est le cas par exemple d'aDEVS (Nutaro, 1999), de PythonPDEVS (Van Tendeloo et Vangheluwe, 2014), ou encore de DEVS-Ruby (Franceschini et al., 2014c), qui sont des noyaux de simulation pouvant être intégrés dans un environnement de modélisation plus large. Par contre, DEVSimPy (Capocchi et al., 2011) est un environnement graphique de modélisation, d'expérimentation et de débogage qui utilise PythonPDEVS comme noyau de simulation. Cependant, la plupart des outils offrant un environnement graphique de modélisation sont basés sur un simulateur développé conjointement. C'est le cas du simulateur CD++ (Wainer, 2002), dont le développement des modèles est facilité à travers l'environnement graphique CD++Builder (Bonaventura et al., 2013). De la même manière, la plateforme VLE (Quesnel et al., 2009) accompagne son simulateur d'un IDE [2], appelé GVLE, permettant la conception graphique de systèmes couplés. Les plateformes DEVS-Suite (Kim et al., 2009), ProDEVS (Le Hung et al., 2015), MS4Me (Seo et al., 2013), ou encore PowerDEVS (Bergero et Kofman, 2011) offrent également des outils de conception graphique de modèles.

Certaines plateformes mentionnent explicitement les systèmes multi-agents comme domaines d'application, en mettent en œuvre les travaux concernant les SMA et DEVS que nous avons présentés dans le chapitre II (*cf.* section 4.2) :

1. *Application Programming Interface*, ou interface de programmation.
2. *Integrated Development Environment*, ou environnement de développement intégré.

James II (Himmelspach et Uhrmacher, 2007a) est une plateforme de modélisation et de simulation générique. Conçue pour être extensible, elle intègre différents formalismes et différents algorithmes de simulation. Les formalismes DEVS et PDEVS y sont supportés. Bien qu'un nombre important de travaux basés sur cette plateforme mentionnent la modélisation d'agents (Schattenberg et Uhrmacher, 2001 ; Steiniger et al., 2012 ; Uhrmacher et al., 2000), aucune abstraction permettant d'en faciliter la modélisation n'est fournie par le métamodèle de la plateforme. Les différentes extensions implémentées, comme dynDEVS, ρ-DEVS ou ML-DEVS permettent de modéliser des systèmes dotés d'une dynamique structurelle.

MIMOSA (Müller, 2004) pour Méthodes Informatiques de MOdélisation et Simulation Agents est également un environnement de M&S générique permettant une modélisation multi-formalisme. Cette plateforme met l'accent sur la définition d'ontologies dont les concepts peuvent ensuite être associés à une dynamique comportementale. La dynamique peut être décrite à l'aide d'un outil graphique basé sur les diagrammes d'état-transitions. MIMOSA peut être considéré comme un environnement de modélisation multi-formalisme qui utilise le formalisme DEVS pour simuler les différents modèles. Plus précisément, cette plateforme implémente la variante de PDEVS baptisée M-DEVS (Müller, 2009) que nous avons présentée dans le chapitre II. Une API expose un ensemble de classes permettant de faciliter la définition du comportement d'agents ainsi que la génération de population d'agents.

Virtual Laboratory Environment (Quesnel et al., 2009) que nous avons rapidement présenté dans le paragraphe précédent a été le support pour l'implémentation de travaux concernant le paradigme agent (Duboz, 2004 ; Quesnel, 2006) et plus précisément de la spécification DSDEMAS (Duboz et al., 2005).

Les différents langages de programmation employés (C++, Java, Python, Ruby) et la particularité des API exposées par toutes ces plateformes montrent clairement un manque d'unité. Une des conséquences de telles disparités est que le partage d'un modèle d'une plateforme à l'autre ne peut être effectué qu'au prix d'un portage du code. Pour pallier à cette problématique, divers groupes de standardisation ont été créés (DEVS Standardization Group, 2001 ; RED, 2014). L'interopérabilité entre simulateurs, l'élaboration d'une certification de conformité avec le formalisme sont au cœur des problèmes abordés. Un autre élément de solution réside dans l'utilisation de techniques issues de l'ingénierie dirigée par les modèles (IDM), vers lesquelles plusieurs auteurs de plateformes de M&S se sont orientés. L'IDM (Bézivin, 2005) est une approche de développement logiciel qui met l'accent sur la notion de modèle pour représenter des artefacts logiciels (Object Management Group, 2015). Bien que la modélisation en ingénierie logicielle a toujours eu une place importante dans la conception d'un logiciel, les modèles ont uniquement une valeur contemplative (Selic, 2003), étant conçus à des fins de description ou de documentation. Les techniques de l'IDM permettent de mettre à profit ces modèles, en appliquant des opérations de transformation afin de générer du code exécutable par exemple. L'IDM offre également des perspectives concernant la vérification ; une phase de vérification automatique sur le modèle peut être appliquée afin de s'assurer qu'il soit bien formé.

Certains outils offrent un environnement intégré de modélisation. Ils permettent de

simplifier la définition de modèles à travers une syntaxe spécifique et de générer un programme informatique automatiquement. AToM3 (De Lara et Vangheluwe, 2002) par exemple, est un outil multi-formalisme générique de M&S basé sur ces concepts. Il supporte notamment les automates à états finis, les réseaux de Petri et permet de générer des simulateurs respectant la sémantique opérationnelle du formalisme cible. Les travaux de Touraille et al. (2010) avec la plateforme SimStudio se concentrent sur le formalisme DEVS. L'environnement de modélisation proposé permet la définition de modèles DEVS à l'aide d'outils graphiques et textuels. Les modèles définis sont conformes au métamodèle du formalisme DEVS défini par les auteurs, baptisé DML. Une phase de transformation permet de générer un code exécutable des modèles pour le noyau de SimStudio, baptisé DEVS-MetaSimulator. Une approche similaire a été proposée avec la plateforme ProDEVS (Le Hung et al., 2015), un environnement de M&S dédié à la simulation de systèmes hybrides basé sur le formalisme DEVS. Une interface propose de décrire graphiquement le couplage des composants et la hiérarchie du système ; une notation graphique basée sur les diagrammes d'états-transitions permet également de décrire le comportement des modèles atomiques. Les modèles issus de cette notation sont conformes au métamodèle SiML (Foures, 2015), sur lequel des vérifications sont appliquées et qui permet de générer un code exécutable. Ces deux dernières approches sont intéressantes puisqu'elles permettent d'automatiser une partie du processus de M&S. Les métamodèles proposés dans ces approches forment un langage spécifique, dédié à un domaine, que l'on appelle DSL pour *Domain Specific Language* (Fowler, 2010). Ils peuvent être associés à une syntaxe graphique ou textuelle ; dans les deux cas, leur grammaire est régie par le métamodèle.

Bien avant l'introduction du concept de DSL par l'IDM, la communauté de la M&S proposait des langages dédiés comme GPSS ou SLAM permettant de faciliter la modélisation. L'IDM a généralisé cette approche en permettant de décorréler le DSL de sa plateforme d'exécution. Que l'approche soit générique (*e.g.* basée sur les concepts de l'IDM [3]) ou spécifique à une plateforme, un modèle exprimé à l'aide d'un DSL est conforme au métamodèle du domaine, qui représente la syntaxe abstraite du DSL. La plateforme DEVS-Ruby (Franceschini et al., 2014c) propose un DSL interne permettant de faciliter la définition de modèle DEVS ou PDEVS et suggère la définition de nouveaux DSL pour faciliter l'utilisation de modèles définis. Les syntaxes graphiques représentent une stratégie pour faciliter l'utilisation de DEVS ; DEVS-Ruby suit une stratégie différente en essayant de tirer partie des connaissances des experts du domaine et de leur aptitude à programmer. Notons qu'il existe d'autres approches de ce type, qui consistent à appliquer des contraintes sémantiques sur un métamodèle DEVS (Garredu et al., 2012), définissant un langage spécifique (Garredu et al., 2011).

Notre objectif principal est d'exécuter des modèles de systèmes multi-agents basés sur notre spécification DPDEMAS. Afin de respecter la spécification, la plateforme sur laquelle nous nous appuyons doit être basée sur le formalisme PDEVS. Celle-ci doit également supporter les extensions DSDE pour exprimer la dynamique du système, ainsi que CellDEVS pour la définition d'environnements discrets. La plateforme doit également être évolutive,

3. Définition d'un métamodèle indépendant d'une plateforme (*Platform Independent Model*, puis transformation vers un modèle dédié à une plateforme spécifique (*Platform Specific Model*).

afin d'intégrer les extensions susceptibles de venir compléter l'approche de modélisation. Parmi les douze différentes plateformes que nous avons présentées, seules VLE, CD++, et DEVS-Ruby réunissent à notre connaissance les trois extensions dont nous avons besoin pour implémenter la spécification DPDEMAS. Notons cependant que la licence qui accompagne la plateforme CD++ n'est pas spécifiée (Van Tendeloo et Vangheluwe, 2016). Or, la mise à disposition du code source du modèle et des outils associés sont considérés comme une condition permettant d'assurer la reproductibilité des expériences numériques (Dao et al., 2016), c'est pourquoi l'utilisation de cette plateforme n'est pas envisagée. La plateforme VLE est mature et réunit toutes les qualités que nous attendons. Son architecture est extensible, ce qui permet d'envisager le support de nouveaux formalismes. En revanche, au vu de l'ampleur du projet il est difficile de saisir tous les aspects de son architecture. À l'inverse, DEVS-Ruby (Franceschini et al., 2014c) est une bibliothèque légère développée en interne ayant des performances en retrait par rapport à VLE, puisqu'elle est pénalisée par l'utilisation d'un langage interprété mais qui a l'avantage de faciliter la définition de langages dédiés. Son architecture a également été conçue pour être extensible. Puisque DEVS-Ruby a été développée par nos soins, son architecture nous est plus familière. De plus, l'expressivité de la syntaxe de Ruby permet d'implémenter des fonctionnalités avec un rendement accru par rapport à l'utilisation de C++. Nos travaux s'intéressent notamment à étendre la plateforme pour supporter des modèles non-modulaires. Il nous a donc semblé plus judicieux de jouir d'une liberté totale sur l'architecture de la plateforme, plutôt que d'intégrer ces changements sur une architecture dont l'évolution a été contrainte du fait de son utilisation concrète. La plateforme que nous présentons dans ce chapitre est basée sur DEVS-Ruby, dans une variante que nous avons baptisé *Quartz*.

3 *Plateforme de M&S Quartz*

Le développement de cette plateforme a été guidé par son but premier : celui de permettre la modélisation et la simulation de systèmes multi-agents basés sur nos travaux de formalisation. Pour cela, nous avons retenu du point de vue du formalisme l'implémentation de PDEVS pour sa gestion simultanée des évènements, ainsi que de l'extension DSDE afin de permettre une dynamique structurelle au cours de la simulation. La simulation de systèmes multi-agents peut mettre à rude épreuve les systèmes informatiques. Les besoins en ressources computationnelles ou en ressources mémoire sont notamment liés au nombre et à la complexité des agents qui composent le système mais également à l'étendue et à la résolution des environnements modélisés. Nous avons donc veillé à réaliser un certain nombre d'optimisations afin d'obtenir des performances décentes (*cf.* annexe C).

La plateforme *Quartz* est un héritage direct de la plateforme DEVS-Ruby (Franceschini et al., 2014c). Comme son nom le laisse supposer, l'implémentation de DEVS-Ruby a été réalisée avec Ruby, un langage interprété au typage dynamique. *Quartz* constitue une variante de DEVS-Ruby, qui est basée sur Crystal [4], un langage compilé au typage statique (avec inférence de types), dont la syntaxe est très similaire à Ruby ; les principales différences résident dans la possibilité pour Crystal d'exprimer des annotations de type. Crystal est

4. https://crystal-lang.org

un langage compilé de haut-niveau, orienté objet et fonctionnel, doté d'un système de typage basé sur les *union types* (Igarashi et Nagira, 2006), dont les valeurs ne peuvent pas être nulles. Cette dernière particularité associée au fait qu'un ramasse-miettes gère l'allocation et la désallocation de la mémoire permet de prévenir les erreurs liées à sa gestion. Ceci constitue un avantage certain vis-à-vis d'un langage comme C++, où la gestion de la mémoire est laissée à l'utilisateur. Malgré la présence d'un ramasse-miettes, Crystal permet de garder la maîtrise des allocations effectuées sur le tas ou sur la pile, puisqu'il supporte les *value types*. Aussi, à travers le support de la généricité et d'un système de macros, le langage permet d'effectuer de la métaprogrammation. Pour ces raisons, Crystal représente un compromis idéal entre l'expressivité de Ruby, qui permet un prototypage rapide des modèles, et les performances et les garanties d'un langage compilé. Deux bases de code distinctes sont disponibles sur un dépôt en ligne [5] pour DEVS-Ruby et Quartz. Les deux versions sont sous licences libres. L'intérêt de réaliser un tel portage réside essentiellement dans le gain de performance, étant donné que Crystal utilise l'infrastructure de compilation LLVM [6] (*Low Level Virtual Machine*) pour générer un code exécutable optimisé. De plus, comme nous le verrons dans la suite de cette section, l'utilisation d'un langage compilé permet de réaliser une analyse statique sur les modèles afin de vérifier certaines propriétés. Compte tenu de la taille de la base de code de DEVS-Ruby (~10000 LOC) et des faibles différences syntaxiques entre Ruby et Crystal, l'effort de portage était relativement faible. Ainsi, quelques différences notables existent entre les deux plateformes, notamment le fait que Quartz supporte uniquement la sémantique du formalisme PDEVS, contrairement à DEVS-Ruby qui supporte le formalismes DEVS et PDEVS.

Nous proposons dans la suite de cette section de donner une vue d'ensemble de Quartz. Dans une première sous section, nous décrivons de l'architecture de la plateforme, puis nous abordons ensuite les différentes fonctionnalités permettant de faciliter l'utilisation de l'outil.

3.1 *Architecture*

Le cadre applicatif que nous avons conçu vise à aider le modélisateur dans la conception d'un modèle exécutable. À ce titre, il doit être considéré comme un cadriciel plutôt qu'une plateforme de modélisation et de simulation. Son architecture a été pensée pour être générique et extensible, afin de permettre l'ajout de nouveaux formalismes. Afin de guider le modélisateur lors de la conception d'un modèle, nous avons veillé à exposer une interface de programmation suffisamment claire, en concordance avec la terminologie et les différents concepts définis par la théorie de la M&S. Bien qu'il n'existe pas d'interface graphique ou d'outils associés à Quartz, un certain nombre de composants génériques permettent de faciliter l'utilisation du cadriciel dans un environnement de modélisation intégré. Il est en effet basé sur des mécanismes évènementiels, permettant de réagir à un certain nombre d'évènements liés à la simulation, ou à l'observation des états et des ports des modèles.

L'architecture respecte autant que possible le concept de séparation explicite entre

5. Dépôt DEVS-Ruby : `https://github.com/devs-ruby/devs` ; dépôt Quartz : `https://github.com/rumenzu/quartz`.
6. `http://llvm.org`

la modélisation et la simulation du formalisme PDEVS. L'avantage de cette conception modulaire réside dans la possibilité de « masquer » les composants liés à l'exécution des modèles : le modélisateur manipule uniquement les composants liés à la modélisation. Un seul composant lié à la simulation est destiné à être utilisé par le modélisateur afin de lui permettre de maîtriser l'exécution du modèle. Puisque le cadriciel suit l'architecture des simulateurs abstraits proposés dans Chow et al. (1994), le modélisateur dispose d'une architecture connue, et peut facilement passer de la spécification à l'implémentation. Afin de les présenter plus en détail, nous regroupons les différents composants du cadriciel selon l'aspect de la modélisation et celui de la simulation. Comme nous allons le voir, le caractère générique de l'outil est dû à l'utilisation de nombreux patrons de conception (Gamma et al., 1994).

3.1.1 *Classes pour la modélisation*

La figure 4.1 représente l'architecture des classes dédiées à la modélisation sous forme de diagramme UML. La classe `Model` est l'abstraction sur laquelle repose l'ensemble du cadriciel ; elle généralise l'ensemble des types de modèles.

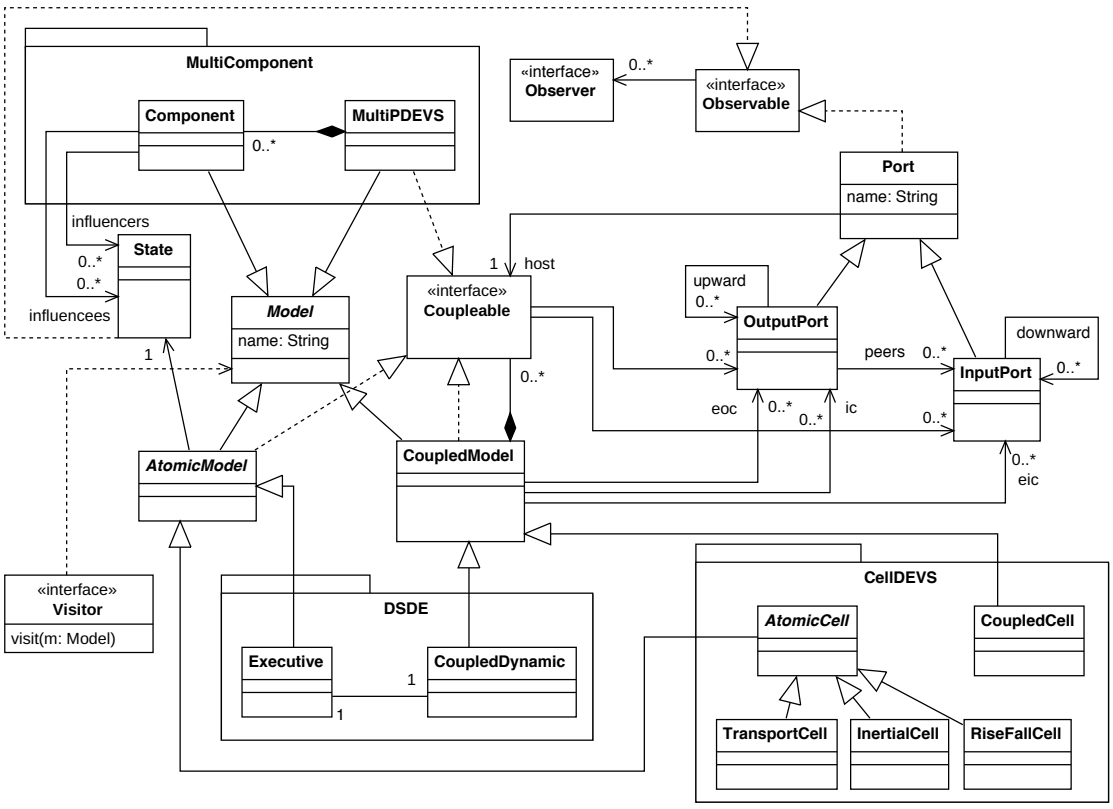


Figure 4.1 Diagramme UML du domaine de PDEVS, partie modélisation.

Les modèles atomiques PDEVS sont représentés par la classe `AtomicModel`, qui est destinée à être étendue par le modélisateur. Conformément à la spécification PDEVS, celle-ci fournit l'ensemble des méthodes permettant d'associer un comportement au modèle atomique, et dispose d'un état associé. Bien qu'abstraite, cette classe définit un comportement

passif par défaut, où chaque instance reste indéfiniment dans son état initial. Les cinq méthodes `external_transition`, `internal_transition`, `confluent_transition`, `output` et `time_advance` doivent être redéfinies pour implémenter un comportement spécifique au modèle atomique. La transition de confluence implémente un comportement par défaut. Comme suggéré dans Zeigler et al. (2000), elle active séquentiellement la transition interne puis la transition externe afin d'obtenir un comportement similaire au formalisme DEVS classique. Afin de faciliter la gestion du temps, nous avons ajouté par commodité l'attribut `sigma`, qui représente le prochain temps d'activation du modèle. Par défaut, la méthode `time_advance` renvoie la valeur de la variable `sigma`, qui est assignée à la constante `INFINITY` . Nous proposons également une méthode spécifique, `post`, destinée à être utilisée dans le corps de la méthode `output` pour déposer une valeur sur un port de sortie. Ainsi, le modélisateur n'a pas à gérer le retour d'une structure de données et certaines optimisations peuvent être réalisées.

Notons que la capacité pour un modèle atomique de disposer d'une interface d'entrée/-sortie n'est pas fournie par la classe `Model`, comme on pourrait s'y attendre, mais à travers l'interface `Coupleable`. Nous avons décomposé la notion de modèle et la notion d'interface d'entrées/sorties afin de pouvoir représenter des modèles non modulaires, comme les composants issus de l'approche multi-composant (Zeigler et al., 2000). Un modèle modulaire, tel que le modèle atomique, est donc représenté par une spécialisation de la classe `Model` et la réalisation de l'interface `Coupleable`. En pratique, `Coupleable` est représentée en Crystal et en Ruby par un *mixin*, une sorte d'interface à laquelle un comportement par défaut peut être associé, et qui représente un moyen idiomatique de réaliser un héritage multiple.

Le caractère hiérarchique des modèles couplés est représenté par le patron de conception *composite*, qui permet de représenter l'ensemble des modèles modulaires sous forme d'arbre. Les composants sont représentés par les objets qui réalisent l'interface `Coupleable`. À travers cette architecture, l'objet composite est représenté par le modèle couplé, qui est implémenté par la classe `CoupledModel`. Celle-ci fournit une API permettant d'ajouter de nouveaux composants ou d'ajouter/supprimer les différents couplages entre les modèles.

Afin de permettre au modélisateur de réagir aux changements d'états ou aux différents messages envoyés par les modèles durant l'exécution du modèle, nous utilisons le patron de conception *observateur* pour envoyer des signaux aux différents observateurs lorsque les objets observables (`Port` et `State`) sont modifiés. Cette architecture permet d'envisager le développement de composants logiciels destinés à réagir à l'évolution des modèles. Par exemple, une interface graphique permettant de visualiser l'état de l'environnement d'un SMA peut être périodiquement rafraîchie. Basé sur le même principe, l'observation d'un port permet par exemple de concevoir un composant destiné à tracer sur un graphe les sorties d'un modèle.

Toujours dans un souci de généricité, notre architecture donne la possibilité de traverser la hiérarchie de modèles à travers le patron de conception *visiteur*. Ainsi, tout composant qui réalise l'interface `Visitor` peut réaliser un traitement en fonction de la hiérarchie de modèles, de l'état des modèles et de leurs couplages. Les visiteurs peuvent notamment être

utilisés pour appliquer une transformation sur la hiérarchie de modèle ou pour générer un fichier. Cette interface est utilisée de manière interne pour implémenter un certain nombre de fonctionnalités. Notons qu'il est possible d'envisager l'implémentation des simulateurs sous cette forme, avec un visiteur par phase de simulation (initialisation, collecte des sorties et exécution des transitions).

Afin de représenter les extensions du formalisme, nous avons défini de nouvelles spécialisations de modèles. Pour permettre la dynamique structurelle, le paquet `DSDE` contient deux nouvelles classes : le modèle couplé dynamique (`CoupledDynamic`), qui hérite du modèle couplé ; et le modèle exécutif (`Executive`), qui hérite du modèle atomique. Le modèle exécutif, contrairement aux autres modèles, peut accéder au modèle couplé dynamique afin d'effectuer des changements structurels. De la même manière, les modèles associés à l'extension CellDEVS (*cf.* annexe A) sont associés au paquet du même nom. La classe `CoupledCell` hérite du modèle couplé afin d'offrir une interface facilitant la création de topologies cellulaires, conformément à la spécification de CellDEVS. Afin de représenter les cellules, nous avons repris la hiérarchie de classe proposée dans l'implémentation de référence de CellDEVS (CD++). La classe abstraite `AtomicCell` hérite du modèle atomique et permet de définir l'interface d'une cellule ainsi qu'une partie du comportement des fonctions formant la dynamique d'un modèle atomique. Trois spécialisations de la classe `AtomicCell` sont définies pour chaque type de délai d'une cellule, afin de définir l'ensemble de fonctions de transition. Enfin, le paquet `MultiComponent` rassemble les composants relatifs à l'extension multiPDEVS, une variante parallèle du formalisme multi-composant (*cf.* annexe A). Le système non-modulaire peut-être couplé à une hiérarchie de modèles PDEVS à travers le modèle `MultiPDEVS`, qui réalise l'interface `Coupleable`. Celui-ci est formé par un ensemble de composants dénués de ports d'entrée/sortie, dont la classe (`Component`) hérite de la classe `Model`. Plutôt que de s'influencer à travers l'envoi de messages, les composants d'un système non-modulaire peuvent s'influencer à travers leurs états.

3.1.2 *Classes pour la simulation*

L'architecture de simulation permet d'exécuter un modèle ayant été défini à l'aide des classes que nous avons présentées dans la figure 4.1. Les classes liées à la simulation implémentent les simulateurs abstraits associés au formalisme PDEVS, ainsi que les simulateurs spécifiques aux différentes extensions. La figure 4.2 représente un diagramme UML simplifié des classes dédiées à l'exécution des modèles.

L'organisation des algorithmes de simulation suivent de près la hiérarchie des modèles. Ils correspondent ainsi à l'organisation suggérée par les simulateurs abstraits (*cf.* II.3.3). Quartz repose donc sur différents types de processeurs pour exécuter les modèles : les simulateurs, les coordinateurs et le coordinateur racine. L'intérêt de cette architecture est qu'elle facilite le développement de nouveaux processeurs, à la fois pour proposer des variantes des algorithmes de simulation (version séquentielle, parallèle, distribuée), mais également pour implémenter les algorithmes associés à une extension du formalisme.

La classe `Processor` est associée à un modèle et permet de représenter de manière

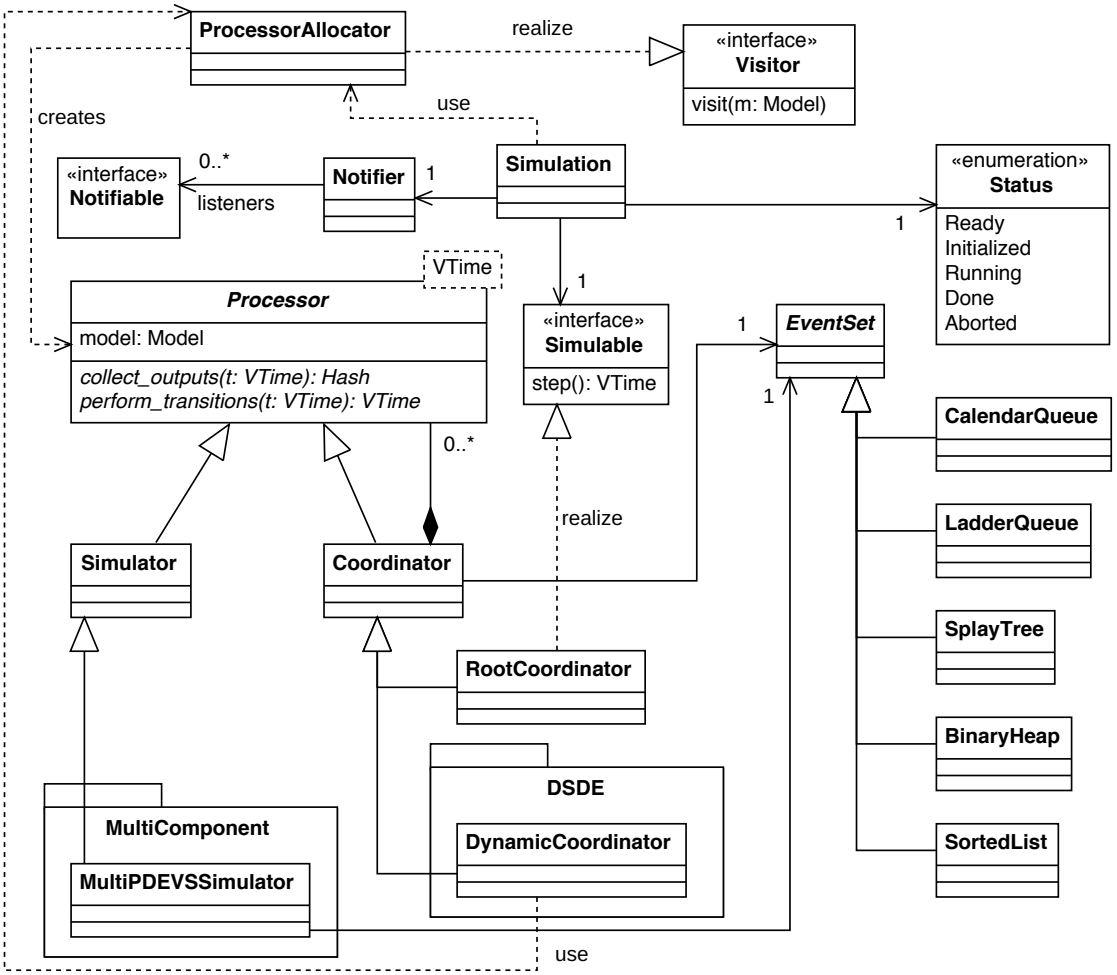


Figure 4.2 Diagramme UML du domaine, partie simulation.

abstraite un composant chargé d'exécuter le modèle. Le processeur concrétise la séparation entre la modélisation et la simulation. Il encapsule l'ensemble des variables externes au modèle et nécessaires à son exécution, comme les variables représentant le prochain temps d'activation (`time_next`) et le dernier temps d'activation (`time_last`). Cette classe fournit également un ensemble de méthodes abstraites que les simulateurs et coordinateurs doivent implémenter et qui représentent le protocole de communication.

Les classes concrètes spécialisant le processeur, `Simulator` et `Coordinator`, sont respectivement responsables de l'exécution du modèle atomique et du modèle couplé. De la même manière que pour les modèles, le patron de conception *composite* permet de représenter la hiérarchie de processeurs, qui est identique à celle des modèles. La classe `Simulator` implémente les algorithmes du simulateur abstrait. À ce titre, elle est responsable de l'activation des fonctions de transitions appropriées et de la récupération des sorties des modèles atomiques pour les faire remonter aux coordinateurs. La classe `Coordinator` implémente les algorithmes du coordinateur. Elle est donc responsable de l'activation des processeurs imminents et de l'acheminement des entrées/sorties produites par les modèles en fonction des couplages des modèles couplés. Afin de supporter la dynamique structurelle, l'exten-

sion DSDE définit une variante des algorithmes de simulation associés au coordinateur. La classe `DynamicCoordinator` spécialise le coordinateur afin de redéfinir le comportement d'une méthode liée à l'exécution des modèles. La classe `MultiPDEVSSimulator` spécialise le simulateur afin d'être associé aux modèles de type `MultiPDEVS`. Pour permettre la simulation de systèmes non-modulaires, ce simulateur est chargé de coordonner la simulation des différents composants d'après les simulateurs abstraits définis dans l'annexe B. Les processeurs chargés d'activer les modèles imminents (`Coordinator` et `MultiPDEVSSimulator`) ont une référence à un échéancier (`EventSet`), chargé d'ordonner les différents processeurs selon leur prochaine date d'activation. Comme nous allons le voir dans la prochaine sous-section, les performances d'exécution peuvent être largement impactées par l'échéancier. C'est pourquoi différents types d'échéanciers sont fournis afin de laisser le choix au modélisateur de l'implémentation à utiliser.

À l'heure où nous écrivons ces lignes, la représentation du temps de simulation (ou temps virtuel) est basée par défaut sur un nombre à virgule flottante codé sous 64 bits, respectant la norme IEEE 754 (IEEE, 2008). Ce format de données n'est pas forcément le plus adapté pour représenter le temps dans un simulateur étant donné qu'il peut être sujet à des erreurs d'arrondi pouvant s'accumuler, et qu'il ne vérifie pas toujours la propriété d'associativité (Goldberg, 1991). Cependant, le temps virtuel est associé au type générique `VTime` et peut donc être remplacé par un autre type de données supportant la comparaison et les opérations d'addition, de soustraction, de division et de multiplication. Le modélisateur peut donc choisir une représentation qui convient à son modèle : un nombre entier, un nombre à virgule flottante ou fixe de précision arbitraire ou encore des types de donnée spécifiquement conçus pour la simulation, tel que celui proposé par Vicino et al. (2014).

La classe `RootCoordinator` est destinée à être placée à la racine de la hiérarchie de processeurs et permet de coordonner l'ensemble de la simulation en propageant les messages au sein de la hiérarchie. Sa seule différence avec un coordinateur est qu'elle réalise l'interface `Simulable`, afin d'implémenter les méthodes liées à l'initialisation de la simulation et à l'exécution d'une étape de simulation (évolution de t à δt). C'est toujours dans un souci d'extensibilité du cadriciel que nous avons mis en place l'interface `Simulable`. Celle-ci permet d'envisager l'implémentation de simulateurs basés sur un protocole de simulation différent. Contrairement aux simulateurs abstraits de PDEVS, la boucle principale de simulation ne se situe pas au niveau du coordinateur racine, mais au niveau de la classe `Simulation`. Celle-ci est destinée à être manipulée par le modélisateur souhaitant exécuter son modèle, et permet d'exécuter le modèle étape par étape, de l'annuler, de la recommencer ou encore de l'exécuter jusqu'à une condition d'arrêt. L'allocation des processeurs est transparente pour le modélisateur ; elle est effectuée à travers le visiteur `ProcessorAllocator`, chargé d'initialiser le processeur correspondant à chaque modèle.

Afin de laisser la possibilité de réagir aux évènements relatifs à l'exécution du modèle, nous avons mis en place un mécanisme basé sur le principe « publier/souscrire », à travers la classe `Notifier`. Les composants réalisant l'interface `Notifiable` peuvent souscrire à certains messages émis par le simulateur. Il est ainsi possible de réaliser un traitement spécifique avant ou après l'initialisation de la simulation, avant ou après le début de la simulation ou

entre chaque étape de la simulation. Ce mécanisme évènementiel permet de faciliter une éventuelle future intégration du cadriciel dans une plateforme de M&S.

3.2 *Fonctionnalités*

Maintenant que nous avons donné une vue d'ensemble de l'architecture de Quartz, nous présentons les différentes fonctionnalités offertes par l'outil afin de faciliter la modélisation et l'exécution des modèles.

3.2.1 *Initialisation des modèles*

La description mathématique du formalisme PDEVS amène à réaliser certains choix pratiques au moment de son implémentation. L'initialisation des modèles en est un exemple. Mathématiquement, l'état initial d'un modèle s'exprime en définissant la variable s_0 (l'état associé au temps 0 de la simulation). Le protocole de simulation de PDEVS dispose bien d'un message *(i,t)* lié à l'initialisation de la simulation. Celui-ci permet de calculer le prochain temps d'activation (*tn*) des modèles en se basant sur leurs état initial (`tl = t-e; tn = tl + ta(s)`). Le message d'initialisation constitue donc le candidat idéal pour initialiser l'état du modèle. Cependant, le simulateur abstrait ne prévoit pas de possibilité d'initialisation par ce biais.

En pratique, l'état initial d'un modèle peut dépendre de paramètres externes. C'est le cas pour un système formé par plusieurs modèles aux comportements homogènes, mais dont les états sont hétérogènes (système spatialisé). Aussi, lors de l'utilisation d'un plan d'expérience, différents états initiaux pour un même modèle sont possibles ; différentes simulations d'un même modèle pour chaque condition expérimentale doivent être effectuées. Afin de faciliter la construction des modèles, plusieurs solutions sont envisageables :

1. Le modélisateur prend en compte les différents paramètres pouvant être initialisés de manière externe, définit un port d'entrée pour chacun d'eux et utilise le premier cycle de simulation pour initialiser l'état des modèles.
2. L'utilisation d'un langage orienté objet permet au modélisateur d'initialiser l'état du modèle au moment de son instanciation, à travers le constructeur de classe.
3. En plus de leur état, les modèles disposent d'une référence vers une structure de données comportant une liste de paramètres. Ces derniers sont utilisés lors de la propagation du message *(i,t)* en argument d'une nouvelle fonction de transition d'état, associée au modèle atomique.

La première solution est tout à fait envisageable, bien que fastidieuse. Comme alternative, la classe du modèle atomique dispose d'un constructeur permettant d'instancier le modèle à partir d'une structure de donnée de type dictionnaire (*e.g.* `MyModel.new("model1", param1 : 42, param2: 0.12)`). Bien qu'intuitive, cette solution n'est pas toujours optimale. L'état du modèle étant encapsulé, si le modélisateur a besoin de recommencer une simulation avec la même hiérarchie de modèles, alors revenir à l'état s_0 (*e.g.* pour effectuer un réplicat de simulation) ou dans un état initial différent (*e.g.* nouvelle condition expérimentale dans un plan d'expérience), nécessite l'allocation d'une nouvelle hiérarchie de modèles. Une

telle opération pouvant être coûteuse selon la complexité du modèle, nous avons mis en place la troisième alternative, qui permet d'éviter de définir des ports d'entrées dédiés à l'initialisation. Formellement, la fonction d'initialisation peut être définie de la manière suivante :

$$\delta_{init} : Params \rightarrow Q$$

où *Params* est un ensemble de valeurs permettant d'initialiser l'état initial du modèle atomique. L'algorithme du simulateur abstrait lié au message d'initialisation implémenté dans Quartz est définit dans le listing suivant :

```
1 when receive i-message (i, t) at time t
2   s, e = δ_init(params)
3   tl = t - e
4   tn = tl + ta(s)
```

Listing 4.1 Algorithme du simulateur abstrait – gestion du message d'initialisation implémenté dans Quartz.

3.2.2 *Développement de modèles*

Etant donné que nous ne fournissons pas de langage graphique de haut niveau permettant de spécifier la structure et le comportement des modèles, le modélisateur doit développer les modèles en utilisant l'API que nous offrons dans le langage hôte. Le fait de devoir programmer les modèles peut constituer une barrière pour le modélisateur ; en faciliter la lecture et l'écriture est donc un aspect important. Dans cet objectif, nous avons mis en œuvre des techniques de métaprogrammation permettant de rendre le codage des modèles moins fastidieux, en minimisant le code répétitif à fournir.

Dans Franceschini et al. (2014c), nous avons décrit les prémisses d'un langage dédié interne textuel (DSEL[7]) pour DEVS-Ruby, afin de fournir un langage de haut niveau permettant de décrire le comportement et la structure des systèmes à travers un vocabulaire spécifique au domaine. Récemment, les auteurs de la plateforme DesignDEVS (Goldstein et al., 2017) se sont également concentrés sur la définition de primitives permettant de faciliter la modélisation d'un modèle atomique. Les avantages d'un DSL par rapport à un langage généraliste sont nombreux. Ils fournissent des abstractions adaptées au domaine et permettent à l'expert du domaine de gagner en expressivité et en productivité. Un DSL textuel peut être externe ou interne. Le premier définit un langage avec sa propre syntaxe et sa propre sémantique, et existe indépendamment d'un autre langage. Son développement peut nécessiter un effort important, bien que des outils permettant de faciliter le processus existent, comme Lex et Yacc (Levine et al., 1992), ou plus récemment des outils d'ingénierie dirigée par les modèles (Voelter et Pech, 2012). À l'inverse, un DSL interne repose sur le langage hôte ; il est définit en utilisant la syntaxe du langage hôte et vient augmenter ce dernier. Cela implique que chaque expression du langage dédié soit une expression valide du langage hôte. Il est donc possible de tirer partie des capacités du langage hôte tout en proposant des expressions idiomatiques d'un domaine. Le DSEL hérite de toutes

7. *Domain-Specific Embedded Language* (Fowler, 2010).

les fonctionnalités du langage hôte (Hudak, 1998) et ne nécessite pas le développement d'une chaîne complète de compilation (*lexer*, *parser* et génération de code). Le langage Ruby est un langage particulièrement adapté à la définition de DSL internes (Fowler, 2010). Le langage Crystal étant en majeure partie influencé par Ruby, il hérite d'un certain nombre de fonctionnalités permettant d'égaler Ruby sur ce point : il supporte les clôtures lexicales (*closures*) et compense son manque de réflexivité par un système de macros permettant d'effectuer de la métaprogrammation.

Etant donné que pour le modélisateur, la majeure partie du code à fournir se situe dans les modèles atomiques, nous présentons rapidement dans cette section quelques expressions que nous avons implémentées sous forme de macro afin de faciliter la définition d'un modèle atomique et qui font partie du langage dédié que nous mettons à disposition du modélisateur. Nous proposons de nous attarder sur deux expressions. La première permet de faciliter la définition des ports et la seconde la définition des variables d'états du modèle.

Définition des ports

L'API d'un modèle atomique comprend différentes méthodes permettant d'ajouter des ports. Lors de la définition d'un modèle, l'ajout de ports peut être effectué au sein du constructeur de la classe de la manière suivante :

```
class MyModel < AtomicModel
  def initialize(name)
    super(name)
    add_input_port(:gamma)
    add_output_port(:theta)
  end
end
```

Afin d'éviter au modélisateur de surcharger le constructeur de cette façon, nous avons ajouté deux expressions permettant de définir les ports d'un modèle au niveau de la définition de classe. Ainsi, l'ajout d'un port d'entrée peut s'effectuer avec le mot-clé `input` et un port de sortie avec le mot-clé `output`. Ces derniers peuvent être définis formellement à travers l'expression BNF suivante :

```
<identifier> ::= /a-zA-Z_/ { /a-zA-Z0-9_/ }
<portname>   ::= <identifier> | ('"' <identifier> '"') | (':' <identifier>)
<portdef>    ::= ("input" | "output") <portname> { ',' <portname> }
```

La définition de `MyModel` peut être défini de manière équivalente par l'algorithme ci-dessous. Chaque instance du modèle dispose alors des ports déclarés au niveau de la classe.

```
class MyModel < AtomicModel
  input  :gamma
  output :theta
end
```

Notons que Quartz étant un portage de DEVS-Ruby, de nombreuses perspectives d'évo-

lution de l'outil sont envisageables afin de pleinement tirer parti du système de typage. Par exemple, il serait possible de typer les ports à travers la généricité dans une future version de l'outil.

Définition des variables d'état

De la même manière que la définition des ports, nous proposons une expression dédiée à la définition d'une variable d'état du modèle. Pour chaque variable d'état, le modélisateur spécifie le nom et le type de la variable à travers le mot-clé `state_var`, et peut également préciser une valeur par défaut. L'expression BNF suivante permet d'en formaliser la syntaxe :

```
<type>     ::= /A-Z/ { /a-zA-Z0-9/ }
<statevar> ::= "state_var" <identifier> ':' <type> [('=' <arg>) | <block>]
```

Notons que les termes `<arg>` et `<block>` font respectivement référence à un argument pouvant être assigné et à une clôture lexicale, tels que proposé dans Crystal. Au delà de rendre la définition d'une variable d'état plus concise, cette expression permet d'identifier les variables d'instances qui font partie de l'état du modèle. Un ensemble de macros permettent la génération automatique de code afin de fournir au modélisateur un certain nombre de fonctionnalités de manière transparente. `state_var` permet au modélisateur d'éviter la définition de constructeurs, ceux-ci étant générés automatiquement. L'état du modèle est encapsulé à travers des variables d'instances. Néanmoins, afin de permettre son observation tout en préservant l'autonomie du modèle, un type de donnée spécifique qui hérite de `State` est généré automatiquement. Une méthode associée au modèle permet de récupérer une instance de ce type de donnée et celui-ci comporte également des méthodes de sérialisation et de désérialisation. Les deux listings ci-dessous donnent une idée d'une partie du code généré à travers `state_var`.

```
class MyModel < AtomicModel
  state_var x : Int32 = 42
end
```

Le second listing présente la définition d'un modèle équivalent, en donnant une partie du code généré par l'expression `state_var` :

```
class MyModel < AtomicModel
  @x : Int32

  def initialize(name)
    super(name)
    @x = 42
  end

  def initialize(name, x : Int32 = 42)
    super(name)
    @x = x
  end
```

```
  struct State < Quartz::State
    ...
  end
  ...
end
```

Nous ne donnons pas de détails sur la construction des modèles couplés. Ceux-ci peuvent être instanciés ou spécialisés, et disposent d'une API permettant d'ajouter des composants, des ports d'entrée/sortie et des couplages entre le couplé et ses composants.

3.2.3 *Vérification*

La vérification d'un modèle consiste à s'assurer que celui-ci ne comporte pas d'erreurs, et peut être effectuée à différents niveaux (*cf.* figure 1.13). Elle ne doit pas être confondue avec la validation des modèles qui vise à vérifier si le modèle est représentatif du système étudié. Les différentes relations de vérification consistent à comparer un même modèle exprimé sous différentes formes afin de détecter d'éventuelles incohérences. La vérification de l'implémentation d'un modèle, c'est-à-dire le passage de la spécification du modèle de simulation au modèle de simulation sous forme de programme informatique, peut être effectuée de différentes manières, et de façon plus ou moins exhaustive. L'approche IDM est certainement la plus adaptée pour effectuer des vérifications, étant donné qu'elle permet d'avoir une représentation informatique de la spécification du modèle, dont les éléments sont conformes à un métamodèle de PDEVS. Etant donné que tous ces éléments sont manipulables sous forme d'entités informatiques, cela offre de nombreuses possibilités de vérifications, d'autant plus que celles-ci peuvent être détectées très tôt dans le processus de développement d'un modèle. Toutefois, en dehors d'une approche IDM, il est possible de réaliser une analyse automatique, moins exhaustive, de l'implémentation d'un modèle.

Il est important de distinguer les erreurs liées aux propriétés du modèle des erreurs génériques, qui peuvent être vérifiées pour tous les modèles basés sur le formalisme PDEVS. Voici une liste non-exhaustive de ces propriétés : l'état d'un modèle ne doit pas être modifié en dehors de ses fonctions de transitions ; les couplages peuvent uniquement lier deux composants appartenant à la liste des composants d'un même modèle couplé ; un modèle atomique ne peut pas modifier sa structure dynamiquement.

Les différentes propriétés peuvent être détectées de manière statique ou en cours d'exécution. Si notre outil ne permet pas la vérification exhaustive des modèles, il effectue néanmoins un certain nombre de vérifications génériques durant l'exécution du modèle. Nous offrons également des abstractions permettant au modélisateur d'effectuer des vérifications à l'exécution sur l'état du modèle. Nous donnons également une preuve de concept de vérification statique qui offre des perspectives d'évolutions intéressantes pour Quartz.

Vérifications à l'exécution

Dans Quartz, nous effectuons un certain nombre de vérifications à l'exécution. Celles-ci concernent des propriétés génériques qui peuvent être vérifiées pour tous les modèles

PDEVS et comprennent uniquement les sous-ensembles suivants :

1. la vérification de la positivité de la fonction d'avancement du temps à travers la vérification de la causalité ;
2. la vérification de la validité des couplages (boucle de rétroaction, modèles impliqués appartenant au même modèle couplé ou au couplé lui-même) ;
3. la vérification de la sortie des modèles atomiques : les ports impliqués doivent appartenir au modèle ;
4. la vérification de la compatibilité du processeur chargé d'exécuter un modèle.

Pour les vérifications spécifiques aux modèles, une méthode recommandée (Zeigler et al., 2000) consiste à définir un modèle particulier, appelé cadre expérimental, dont l'objectif est de vérifier un certain nombre de relations entre les entrées et les sorties du modèle. Cependant, cette technique ne permet pas de vérifier des propriétés directement sur l'état du modèle. Afin d'y remédier, nous avons mis en place des abstractions permettant de vérifier l'état d'un modèle après chaque transition d'état. Ces états peuvent donc être différents d'une simulation à l'autre en fonction des évènements externes et des paramètres initiaux. L'espace d'état d'un modèle pouvant être infini, ce type de vérification est loin d'être exhaustif. Le mécanisme permet cependant de détecter différentes erreurs pour une simulation donnée.

La figure 4.3 illustre à travers un diagramme UML les composants chargés de réaliser les vérifications. Les modèles atomiques réalisent l'interface `Verifiable`. Chaque type marqué comme `Verifiable` dispose d'un ensemble de composants (`RuntimeChecker`) chargés de tester des propriétés spécifiques. Le modélisateur peut alors créer son propre composant ou utiliser un validateur générique existant. Quartz fournit par exemple un composant permettant de vérifier certaines propriétés sur des valeurs numériques (positivité, valeur dans un intervalle, valeur finie, ...).

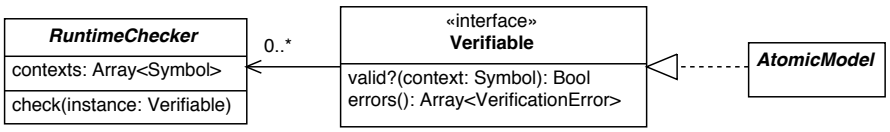


Figure 4.3 Diagramme UML illustrant les abstractions permettant au modélisateur d'effectuer des vérifications à l'exécution sur l'état du modèle après chaque transition.

Afin d'effectuer les vérifications après chaque transition d'état, une adaptation des algorithmes du simulateur abstrait a été nécessaire afin d'appeler la méthode `valid?` après chaque transition d'état du modèle. Ces changements affectent la gestion du message *(*,t)* (activation des transitions) ainsi que le message *(i,t)* (initialisation) étant donné que nous avons ajouté la fonction δ_{init}. Si des erreurs sont détectées en cours de simulation, un mode strict permet d'interrompre l'exécution. Le mode normal affiche les erreurs sous forme d'avertissement en utilisant la sortie erreur standard. Les vérifications à l'exécution pouvant affecter les performances d'exécution du modèle, une option de simulation permet de désactiver celles qui sont liées aux modèles. De cette manière, si le modélisateur considère le modèle comme valide, il peut réaliser des expériences numériques en favorisant les performances d'exécution.

Toujours dans l'objectif de faciliter le développement des modèles atomiques, nous avons rajouté une expression spécifique (`check`) permettant d'énumérer les différentes vérifications à effectuer sur les variables d'états lors de la définition d'un modèle. Sans entrer dans les détails, nous donnons un exemple dans le listing 4.2 de la définition d'un modèle doté d'une variable d'état qui représente une valeur angulaire en degrés. Le type de donnée utilisé est un entier codé sur 2 octets[8] et permet d'exprimer bien plus de valeurs qu'il n'en faut. Il est possible d'exprimer une contrainte à respecter afin de s'assurer que la valeur angulaire soit toujours dans l'intervalle $[0;360]$.

```
1 class NavigationModel < AtomicModel
2   state_var bearing : Int16 = 90
3   check :bearing, numericality: { gte: 0, lt: 360 }
4   ...
5   def internal_transition
6     @bearing = -123
7   end
8 end
```

Listing 4.2 Exemple de définition d'un modèle avec spécification de contraintes vérifiées à l'exécution.

La transition interne du modèle ci-dessus (listing 4.2) assigne une valeur invalide à la variable `bearing`. L'exécution du modèle avec l'option de vérification activée en mode normal permet de détecter l'erreur en cours de simulation et d'afficher l'erreur donnée dans le listing 4.3 ci-dessous.

```
1 (10:17:51:273) > 'nav' is invalid (context: 'internal_transition', vtime: 1.0).
     Errors: ["'bearing' must be greater than or equal to 0"]
```

Listing 4.3 Exemple d'erreur détectée durant l'exécution du modèle.

Notons que ce mécanisme offre une perspective d'évolution intéressante de l'outil. Si à l'heure actuelle ces vérifications sont appliquées sur les états du modèle, il est possible de généraliser le concept et d'appliquer des vérifications sur les valeurs reçues et envoyées sur les ports d'entrée/sortie du modèle.

Vérifications statique

Par rapport aux vérifications effectuées durant l'exécution du modèle, la vérification statique permet de détecter les erreurs beaucoup plus tôt dans le processus développement d'un modèle. Cette technique est préférable puisqu'elle permet d'empêcher la génération du programme informatique ou de l'exécutable. Dans une approche IDM, un processus de vérification peut être appliqué sur le modèle afin d'autoriser ou non la transformation du modèle. Dans une approche plus classique de compilation de programme, des vérifications peuvent être appliquées sur l'arbre syntaxique (AST) après la phase d'analyse syntaxique. Ainsi, si des erreurs sont détectées, le processus de compilation peut être interrompu afin d'éviter la génération d'un exécutable erroné. En fonction du métamodèle adopté, le système

8. Nous partons du principe qu'une précision d'un degré est une abstraction suffisante pour ce modèle.

de typage du langage permet déjà de lever un certain nombre d'erreurs structurelles de manière statique. Les erreurs liées à la sémantique sont plus difficiles à détecter et nécessitent une analyse spécifique de l'arbre syntaxique du programme.

Actuellement, Quartz n'effectue pas de vérification sémantique spécifique. Cependant, le langage Crystal offre des perspective intéressantes pour réaliser ce type de vérification. Un système de macro permet de réagir à certains évènements durant la phase de compilation. Par exemple, la définition de la macro `inherited` associée à une classe permet d'effectuer un certain nombre de vérifications à chaque fois qu'une spécialisation de cette classe est définie.

En utilisant ce mécanisme, il est possible d'effectuer un certain nombre de vérifications sémantiques sur les modèles. Afin de donner une preuve de concept, nous donnons un exemple de vérification de la sémantique de la fonction d'avancement du temps d'un modèle atomique. Dans Quartz, l'état d'un modèle atomique est stocké sous forme de variables d'instances. La fonction d'avancement du temps étant une méthode, l'état est accessible et muable depuis celle-ci. Le modélisateur n'est pas supposé modifier l'état du modèle depuis cette méthode bien que le paradigme objet le lui permette. Le listing 4.4 définit un ensemble de macro permettant de réagir à la définition d'une méthode dans l'ensemble des spécialisations de la classe `AtomicModel`. Si la méthode correspond à celle d'avancement du temps, le corps est passé en revue afin de vérifier si une variable d'instance y subit une affectation. Si c'est le cas, la compilation est stoppée avec un message d'erreur contextuel.

```
1  class AtomicModel
2    macro inherited
3      macro method_added(method)
4        \{% if method.name.stringify == "time_advance" %}
5          \{% for expr in method.body.expressions %}
6            \{% if expr.is_a?(Assign) %}
7              \{% if expr.target.is_a?(InstanceVar) %}
8                \{% expr.raise "you're not supposed to mutate the state of an
                     atomic model from the #time_advance method." %}
9              \{% end %}
10           \{% end %}
11         \{% end %}
12       \{% end %}
13     end
14   end
15 end
16
17 class MyModel < AtomicModel
18   state_var x : Int32
19   def time_advance
20     @x = 54
21     return INFINITY
22   end
```

```
23 end
```

Listing 4.4 Preuve de concept de vérification statique d'un modèle, basée sur le système de macro du langage Crystal.

La définition du modèle atomique de type `MyModel` donnée dans le listing 4.4 est valide pour la sémantique du langage mais invalide vis-à-vis de la sémantique de PDEVS. La vérification que nous avons appliquée permet de détecter l'erreur de modélisation pendant la compilation. Le listing 4.5 donne un exemple de la sortie affichée.

```
1 Error in examples/mymodel.cr:21: "you're not supposed to mutate the state of an
     atomic model from the #time_advance method."
2     @x = 54
3     ^~~~~~
```

Listing 4.5 Exemple d'erreur détecté de manière statique par Quartz lors de la compilation du modèle décrit par le listing 4.4.

Ce mécanisme permet d'envisager de futures améliorations pour notre outil. Notons que dans notre exemple, la nécessité de vérifier la contrainte concernant la modification de l'état peut être évitée en adoptant un métamodèle plus fidèle à la définition mathématique du formalisme. Par exemple, au lieu de stocker l'état dans des variables d'instance du modèle, il est possible d'encapsuler systématiquement les variables dans un type de donnée immuable et de passer en paramètre celui-ci à la fonction d'avancement du temps. Cependant, le modélisateur perdrait selon nous en expressivité. De plus, pour empêcher véritablement la modification de l'état, il est nécessaire de s'assurer que l'instance de l'état passée en paramètre de la fonction soit une copie profonde de l'état courant du modèle, ce qui entraîne des pénalités en terme de performance d'exécution.

3.3 *Implémentation de la spécification DPDEMAS*

L'implementation de la spécification *Dynamic Parallel Discrete Event Multi-Agent Specification* (DPDEMAS) que nous avons définie dans le chapitre III a été réalisée à l'aide du cadriciel Quartz que nous venons de présenter. Cette implémentation prend la forme d'une bibliothèque qui a pour dépendance notre environnement de M&S générique. L'objectif principal de cette bibliothèque est d'augmenter le cadriciel Quartz en fournissant les différentes abstractions permettant de faciliter le développement d'un modèle de système multi-agents. Ces différentes abstractions permettent également de fournir au modélisateur un vocabulaire qui se rapproche de celui du paradigme agent. Nous espérons ainsi réduire l'écart qui peut exister entre la communauté SMA et la communauté de la TMS.

Dans cette section, nous ne nous attardons pas sur l'architecture des différentes entités venant compléter ou étendre celles de notre environnement de M&S. En effet, l'implémentation de DPDEMAS est assez fidèle au métamodèle que nous avons présenté dans le chapitre III (*cf.* section III.3.1). Dans un souci de concision, cette section est donc axée sur des aspects plus pragmatiques. Nous présentons dans une sous-section les différentes problématiques qui ont nécessité des ajustements lors de l'implémentation, dans l'objectif

d'offrir des performances d'exécution convenables tout en restant fidèles à la spécification.

3.3.1 *Représentation de topologies continues*

Par définition, un environnement basé sur une topologie continue englobe une infinité de valeurs. Afin de fournir une implémentation d'un environnement 2D continu, nous avons dû adapter le métamodèle de DPDEMAS afin de pouvoir représenter les variables formellement définies comme une partie de l'ensemble des positions, $\wp(P)$, par exemple la portée de perception de l'agent. La représentation informatique de ce type de variable est problématique, plus particulièrement pour des valeurs continues. De même, bien que des positions discrètes soient dénombrables, représenter une partie de l'espace par une liste de positions constitue une approche naïve. Nous nous sommes donc tournés vers les structures de données et les algorithmes issus du domaine de la géométrie algorithmique (Berg et al., 2008) afin de représenter ce type de variable.

Ainsi, le modélisateur peut utiliser différents types de données représentant des formes géométriques qui peuvent être situées dans l'espace. Nous avons défini l'interface `Shape` afin de généraliser le concept de forme géométrique. La figure 4.4 illustre l'ajout du concept de forme géométrique dans le métamodèle de DPDEMAS. On peut y voir que la portée de perception de l'agent, représentée par le type abstrait `PerceptionScope` généralise celle basée sur les formes géométriques `ShapeScope` utilisée pour les environnements continus, et la portée de perception `DistanceScope` utilisée pour les environnements cellulaires. Le type `ShapeScope` dispose d'une référence vers une forme géométrique. L'interface `Shape` impose

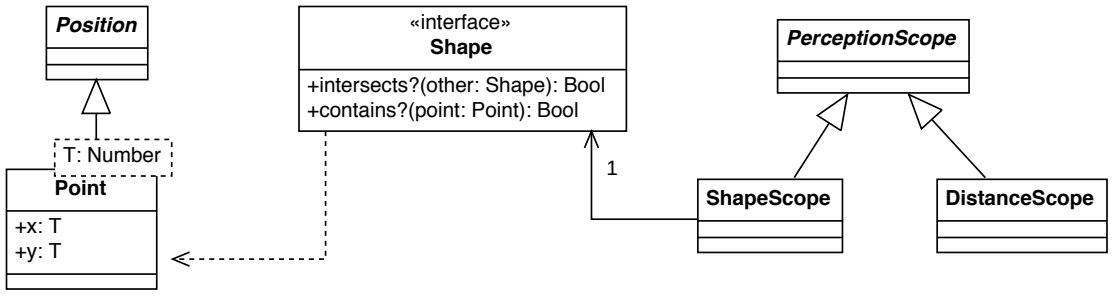


Figure 4.4 Métamodèle UML faisant apparaître le concept de forme géométrique pour représenter des parties de l'espace.

la définition de différentes méthodes permettant de déterminer si une forme contient une position, ou si deux formes entrent en collision. Celles-ci sont primordiales puisqu'elles permettent de faciliter l'implémentation du calcul des percepts et du calcul du voisinage.

Bien que les types de données dédiés aux formes géométriques permettent d'optimiser les ressources en mémoire, lorsqu'il s'agit de représenter un environnement continu, déterminer si une forme ou un point est inclus dans une autre forme peut potentiellement être coûteux en terme de performances d'exécution. Afin d'optimiser ces types de calculs, nous avons utilisé des techniques d'indexation spatiale pour implémenter l'environnement continu. Celle-ci repose sur une structure de donnée permettant de discrétiser l'espace en

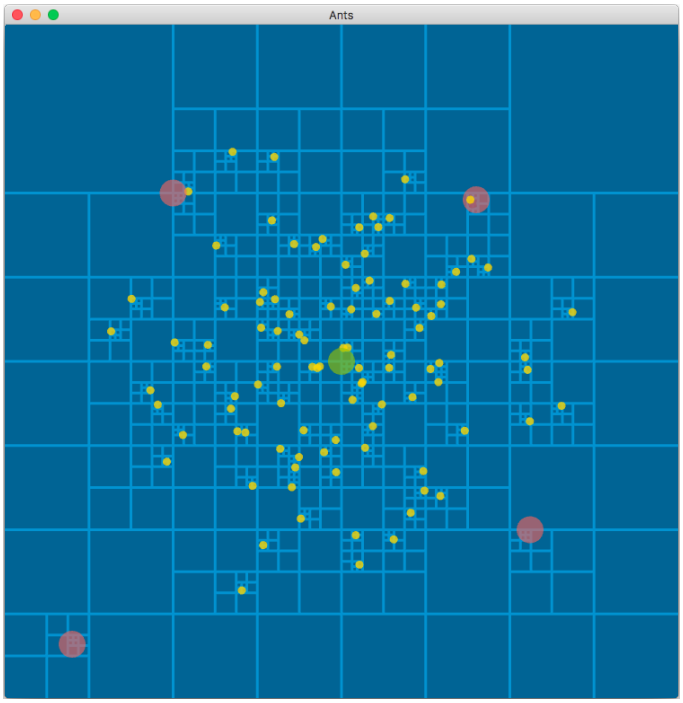


Figure 4.5 Illustration de l'indexation spatiale basée sur un *quadtree* (Finkel et Bentley, 1974) d'un environnement 2d continu dans Quartz.

plusieurs partitions. La structure que nous avons utilisée est le *quadtree* (Finkel et Bentley, 1974), qui subdivise l'espace récursivement dès que le nombre d'éléments dans une partition dépasse un certain seuil. La figure 4.5 illustre un environnement 2d continu dans lequel sont situés un certain nombre d'agents et de ressources. L'état du *quadtree* est représenté par les lignes claires. Lorsqu'il s'agit de rechercher des entités situées à proximité d'une position donnée (*e.g.* calcul du voisinage de l'agent), cette structure permet de réduire l'espace de recherche.

3.3.2 *Réaction aux influences*

Au moment d'intégrer un environnement dans son modèle de système multi-agents, l'intérêt d'utiliser une implémentation basée sur DPDEMAS devient évidente. Le modélisateur peut en effet spécialiser l'environnement générique adapté à la topologie qui convient à son modèle. Celui-ci doit cependant donner un comportement à la méthode abstraite `reaction`, afin de valider ou non les différentes influences émises par les agents. Lorsqu'une influence est considérée comme valide, le modélisateur doit définir le changement d'état associé à l'influence. Pour prendre en compte ces influences, la fonction de réaction peut mettre à jour la position du corps de l'agent.

Lorsqu'un nombre important d'influences doivent être traitées par la fonction de réaction, cette dernière peut rapidement devenir complexe. Afin de simplifier sa définition, nous avons légèrement adapté la spécification formelle afin de laisser la possibilité au modélisateur de décomposer les changements d'états. Dans notre implémentation, la classe abstraite

`Influence` définit une méthode abstraite nommée `commit`, qui reçoit en paramètre une référence de l'état de l'environnement. Ainsi, lorsque le modélisateur définit une influence, le comportement lié à l'action est associé à celle-ci. Formellement, la fonction `commit` peut être associée à une influence $\gamma \in \Gamma$ et définie de la manière suivante :

$$commit : S_{env} \times \Gamma \rightarrow S_{env} \times \Gamma$$

avec $env \in D_E$ l'identifiant de l'environnement auquel l'état est associé.

Afin de faciliter la gestion du cycle de vie d'une influence, la phase est encapsulée et mise à jour uniquement par l'influence. Le cycle de vie est illustré par le diagramme d'état-transition en figure 4.6. Au moment de sa création par l'agent, l'influence est en attente de validation ou de rejet. Lorsque celle-ci est reçue par l'environnement, l'influence peut être validée ou rejetée durant le calcul de réaction avec les méthodes respectives `satisfy` ou `rejected`. En fonction de la décision appliquée, la phase de l'influence est mise à jour en conséquence. Si la méthode `satisfy` est appelée, celle-ci active la méthode `commit` définie par l'utilisateur afin de modifier l'état de l'environnement.

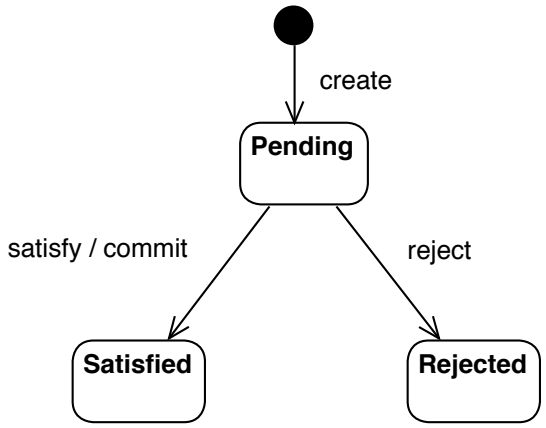


Figure 4.6 Diagramme UML d'état-transition d'une influence.

Cette décomposition des changements d'états de l'environnement évite au modélisateur de dissocier l'influence de l'effet qu'elle est supposée produire sur l'environnement. Elle a également l'avantage de réduire la complexité de la fonction de réaction puisque celle-ci ne contient plus que la logique destinée à valider ou non les différentes influences.

4 *Éléments d'optimisation*

À mesure que nos connaissances sur les systèmes étudiés grandissent, les modèles se complexifient. Cette tendance, associée à l'apparition des paradigmes de modélisation microscopiques, entraîne inévitablement une augmentation du temps nécessaire à la simulation d'un modèle de manière préoccupante. Pour un système multi-agents, les besoins en ressources computationnelle et mémoire sont directement liées au nombre d'agents et à la complexité de leurs architectures. Le ou les environnements que les agents partagent, au delà

de représenter une source de contention, sont également responsables de l'augmentation des besoins en ressources. Par exemple, en fonction de la topologie, de l'étendue et de la résolution spatiale à représenter, un modèle d'environnement peut à lui seul solliciter une part importante des ressources. Nous avons tenu compte de ces problématiques en optimisant un certain nombre d'aspects du simulateur afin d'obtenir des performances d'exécution satisfaisantes.

Un nombre important de travaux permettant d'améliorer le temps d'exécution des simulateurs DEVS ont été proposés. Les différents aspects sur lesquels nous avons travaillé concernent :

- l'élimination du modèle de communication par message à travers la séquentialisation des simulateurs abstraits ;
- la simplification de la hiérarchie de modèles, lorsque celle-ci a une profondeur supérieure à 1 ;
- l'implémentation d'un échéancier configurable et performant, en s'appuyant sur différentes files de priorité et en prenant en compte uniquement les modèles actifs ;
- la possibilité d'effectuer des simulations en tirant partie d'architectures parallèles.

Une section est dédiée à chacune de ces optimisations.

4.1 *Exécution séquentielle des modèles*

Le formalisme PDEVS, de la même manière que le formalisme DEVS, est accompagné de la définition de simulateurs abstraits (Chow et al., 1994), qui décrivent la façon dont un modèle PDEVS doit être exécuté. Leur objectif premier est de définir la sémantique opérationnelle du formalisme. Lorsqu'il s'agit d'en réaliser une implémentation, la mise en application de certains concepts décrits dans les simulateurs abstraits, comme la communication par passage de message, n'est pas la manière la plus efficace d'exécuter un modèle. En effet, les simulateurs abstraits suggèrent que chaque modèle soit simulé de manière concurrente par un composant particulier : le simulateur pour les modèles atomiques et le coordinateur pour les modèles couplés. Le modèle de communication suggéré entre ces composants est le passage de message ; chaque composant attend la réception d'un message pour effectuer un traitement sur le modèle associé. En pratique, comme le soulignent Himmelspach et Uhrmacher (2006), implémenter un simulateur basé sur ces concepts s'adapte mal à l'augmentation du nombre de modèles à simuler, en raison des limitations de la mémoire disponible ; ou des limites du nombre de fils d'exécution imposées par le système d'exploitation.

De manière similaire aux approches proposées par Muzy et Nutaro (2005), Himmelspach et Uhrmacher (2006) ou plus récemment Vicino et al. (2015), l'architecture de notre plateforme de simulation *Quartz* est basée sur une exécution séquentielle des modèles. De fait, la communication par message n'est plus nécessaire ; elle peut être remplacée de manière équivalente par un simple appel de fonction et l'utilisation de valeurs de retour. Comme le fait remarquer Reynolds Jr. (1988) :

« *We note that messages can be simulated in environments where message passing is not required. [...] Message passing is a part of an assumed logical model that may not be required in an actual implementation.* »

L'architecture de *Quartz*, que nous avons présentée dans la section précédente, conserve les concepts de simulateurs et de coordinateurs, où chaque modèle dispose d'un processeur dédié. Le protocole de simulation associé au formalisme PDEVS (Chow et Zeigler, 1994), que nous avons présenté dans le chapitre II (*cf.* figure 2.4) comporte au total six[9] types de messages. Trois grandes phases du protocole de communication peuvent être identifiées : l'initialisation et pour chaque étape de simulation, d'abord la phase collecte puis la phase d'activation des transitions. Pour chacune d'elles, nous remplaçons les différents messages associés par une méthode spécifique (*cf.* tableau 4.1).

Phase	**Messages impliqués**	**Fonction**
Initialisation	*(i,t)* ↓, *(d,t)* ↑	`initialize_state`
Collecte des sorties	*(@,t)* ↓, *(y,t)* ↑, *(x,t)* ↓, *(d,t)* ↑	`collect_outputs`
Activation des transitions	*(*,t)* ↓, *(d,t)* ↑	`perform_transitions`

Tableau 4.1 Synthèse des différents appels de méthodes utilisés dans *Quartz*.

- L'initialisation de la simulation, où le message *(i,t)* est propagé depuis la racine et où les différents processeurs font ensuite remonter leur date d'activation *tn* via le message *(d,tn)*. Ces deux messages sont remplacés dans *Quartz* par la méthode `initialize_state (t : VTime): VTime`, dont la valeur de retour correspond au *tn*.
- Pour la phase de collecte, le message *(@,t)* est propagé depuis la racine pour l'ensemble des composants imminents. Les simulateurs font remonter les sorties des modèles à leur coordinateur parent via le message *(y,t)*. En fonction des couplages, le coordinateur propage les messages plus haut *(y,t)* ou plus bas *(x,t)* dans la hiérarchie. Dans *Quartz*, la méthode `collect_outputs(t : VTime): Hash(Port, Object)` remplace le message *(@,t)*. La valeur de retour correspond à l'ensemble des sorties qui doivent remonter au parent, et remplace ainsi le message *(y,t)*. Enfin les paires de messages intermédiaires *(x,t)* et *(d,tn)* sont remplacées par une structure de donnée associée à chaque processeur, qui permet de stocker temporairement les entrées des modèles attendant d'être traitées dans une seconde phase de l'étape de simulation.
- Enfin, pour la phase d'activation des transitions, le message *(*,t)* est propagé depuis la racine pour l'union de l'ensemble des composants imminents et des composants ayant été influencés par une entrée. Les différents processeurs utilisent à leur tour le message *(d,t)* pour faire remonter leur prochain temps d'activation. La méthode de substitution utilisée dans notre plateforme est définie par `collect_outputs(t : VTime): VTime`, où la valeur de retour correspond au *tn*.

9. Dans l'ouvrage de Zeigler et al. (2000), le protocole de communication associé à PDEVS diffère de celui proposé par Chow et Zeigler (1994), bien que la sémantique opérationnelle d'origine soit préservée. La collecte des sorties associée au message *(@,t)* est associée au message *(*,t)*. L'activation des transitions, représentée par *(*,t)*, est associée au message *(x,t)*. Enfin, le message *(x,t)* destiné à acheminer les entrées est supprimé, cette opération étant intégrée à l'activation des transitions.

Parmi les différentes approches que l'on trouve dans la littérature, les algorithmes que nous avons implémentés sont similaires à l'algorithme « *thread-less variant* » proposé par Himmelspach et Uhrmacher (2006) dont l'efficacité a été mesurée par comparaison à des variantes qui suivent de plus près les simulateurs abstraits (Chow et al., 1994). Cet algorithme a lui-même de nombreuses similitudes avec l'algorithme décrit par Vicino et al. (2015), dont l'implémentation a été comparée au simulateur aDEVS (Nutaro, 1999). Bien que le simulateur aDEVS soit considéré comme le plus performant selon plusieurs études comparatives (Franceschini et al., 2014a ; Van Tendeloo et Vangheluwe, 2016), Vicino et al. (2015) ont montré que leur implémentation est au moins aussi efficace que celle d'aDEVS, voire plus performante dans certains cas.

Comme le font remarquer Vicino et al. (2015), l'utilisation du mécanisme appel/retour permet d'avoir une pile d'appels prévisible, linéaire à la profondeur de la hiérarchie de modèles. En effet, les simulateurs enfants n'ont alors plus besoin d'effectuer des appels au coordinateur parent afin de propager les messages y permettant de faire remonter les sorties des modèles, ou les messages d, qui permettent de signaler la fin d'un traitement. Notons cependant que si un nombre important de modèles couplés sont imbriqués, la profondeur de la hiérarchie peut devenir problématique. La pile d'appels étant plus importante, les performances d'exécution peuvent s'en trouver dégradées ; pire, une profondeur trop élevée peut entraîner un dépassement de pile. C'est cette problématique que nous abordons maintenant.

4.2 *Connexion directe des modèles*

La connexion directe (Chen et Vangheluwe, 2010) est une technique d'optimisation assez courante, dont l'intérêt a déjà été mis en évidence (Bae et al., 2016 ; Jafer et Wainer, 2009 ; Zacharewicz et al., 2010). La connexion directe consiste à réaliser une transformation de la hiérarchie de modèles dans le but de la simplifier. Dans une hiérarchie de modèles, les messages qui transitent entre les différents coordinateurs et simulateurs durant une simulation peuvent engendrer un coût plus ou moins important en fonction de la profondeur de la hiérarchie de modèles, du nombre de modèles ou encore du graphe de couplage entre les modèles. La transformation consiste à réduire la profondeur de la hiérarchie en supprimant les modèles couplés de niveau intermédiaire, ce qui permet de réduire le nombre d'étapes nécessaires pour qu'un message soit acheminé au sein de la hiérarchie. Cette transformation peut être appliquée à la hiérarchie des modèles et à la hiérarchie des simulateurs abstraits. Pour la hiérarchie de modèle, les couplages entre modèles doivent être réajustés en conséquence. La figure 4.7 donne l'exemple d'une hiérarchie « mise à plat » [10], où la profondeur est réduite de 3 à 1.

Afin de vérifier l'amélioration des performances en terme de temps d'exécution sur notre plateforme, nous avons conduit une expérience basée sur le test DEVStone (Wainer et al.,

10. La connexion directe et la mise à plat sont des termes souvent utilisés pour désigner le processus de transformation de la hiérarchie et de réajustement des couplages. Elle ne doit pas être confondue avec le processus utilisé par Chen et Vangheluwe (2010) pour transformer un modèle couplé en son résultant atomique de manière automatique.

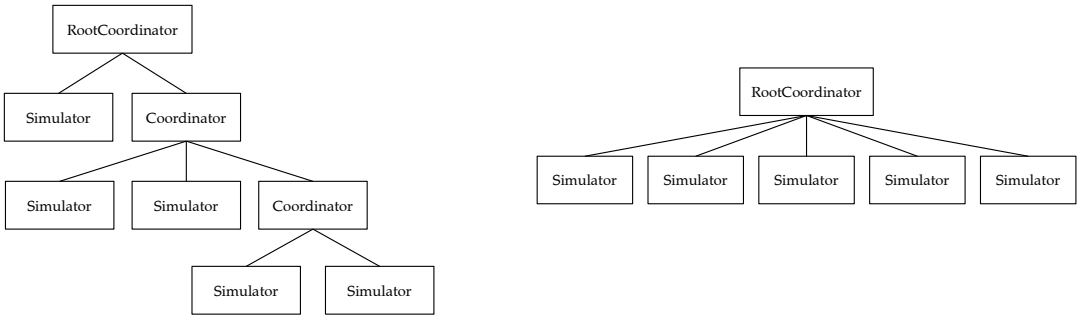


Figure 4.7 Hiérarchie de processeurs avant (a) et après (b) transformation.

Profondeur	Connexion directe	Moy. temps utilisateur (s)	Écart-type rel.
5	✗	1,09150	± 0,65%
	✓	**0,47888**	± 1,91%
10	✗	3,26084	± 0,93%
	✓	**1,11967**	± 1,52%
15	✗	6,55142	± 1,97%
	✓	**1,92219**	± 0,83%
20	✗	10,83112	± 0,77%
	✓	**2,69166**	± 1,74%
25	✗	16,33403	± 1.60%
	✓	**3,61443**	± 1,33%
30	✗	22,76382	± 0.64%
	✓	**4,53505**	± 2,87%

Tableau 4.2 Comparaison des performances d'exécution de *Quartz* avec ou sans connexion directe.

2011) avec et sans connexion directe. DEVStone a été conçu pour évaluer la performance des simulateurs DEVS et permet de générer automatiquement un modèle dont la structure et le comportement varie selon plusieurs paramètres. L'un des paramètres concerne la profondeur de la hiérarchie de modèles. Il est également possible de déterminer différents niveaux de couplages entre les modèles, ainsi que le nombre de modèles atomiques présents à chaque niveau de la hiérarchie (la largeur). L'expérience réalisée initialise le modèle DEVStone avec une largeur de 100 modèles, un couplage de type *HO*, qui correspond à un niveau de couplage élevé générant un nombre important de sorties. Nous faisons varier la profondeur de 5 à 30 et le temps de calcul des fonctions de transitions δ_{int} et δ_{ext} des modèles atomiques sont paramétrés à 0 secondes. Pour chaque profondeur, la simulation est répétée 10 fois, une première fois avec la connexion directe activée et une seconde fois avec cette optimisation désactivée. Le tableau 4.2 montre les résultats d'exécution utilisateur en secondes associés au coefficient de variation sur la profondeur de l'arbre de simulation. Ces résultats montrent une baisse importante du temps d'exécution des simulations lorsque la connexion directe est

activée. Plus la profondeur de la hiérarchie est importante, plus l'accélération est importante. Par exemple, la transformation d'une profondeur de 5 à 1 donne une accélération d'environ 2,28X et la transformation d'une profondeur de 30 à 1 donne une accélération d'environ 5,02X.

Si la réduction de la profondeur de la hiérarchie permet de réduire le chemin qu'un message doit emprunter, cela ne se traduit pas nécessairement par une réduction du temps d'exécution. En effet, cette transformation augmente le nombre de modèles qui doivent être gérés par l'unique coordinateur de la nouvelle hiérarchie. En fonction de la naïveté des algorithmes utilisés pour trier les prochaines dates d'activation des simulateurs, le temps de traitement peut devenir préoccupant, comme nous l'avions conclu dans Franceschini et al. (2014b). C'est cet aspect que nous abordons maintenant.

4.3 *La problématique de l'échéancier*

Au sein d'une simulation à évènements discrets, l'évolution du temps de simulation dépend des dates auxquelles les évènements sont déclenchés. Lorsqu'il y a une période d'inactivité (entre chaque évènement), le temps virtuel « saute » cette période et avance directement jusqu'à la date du prochain évènement. Dans un simulateur à évènements discrets, la responsabilité de la gestion des évènements incombe à l'échéancier, et son implémentation peut avoir un impact considérable sur les performances des simulations. Il est constitué de l'ensemble des événements planifiés dans le temps et doit permettre de récupérer les évènements de manière *ordonnancée* selon les dates d'occurrence des évènements. La structure de donnée qui se prête idéalement à cette problématique est la *file de priorité*.

La gestion de l'échéancier représente une problématique importante dans le domaine de la simulation à évènements discrets. Dans la communauté anglophone, on parle du *pending event set problem* (PES). Durant les 40 dernières années, un effort de recherche important a donné naissance à de nouvelles files de priorité. À notre connaissance, la première étude remonte à 1975 (Vaucher et Duval, 1975), suivie d'autres à mesure que de nouvelles structures ont fait leur apparition (Jones, 1986 ; McCormack et Sargent, 1981 ; Nikolopoulos et MacLeod, 1993). Plus tard, Himmelspach et Uhrmacher (2007b), puis Franceschini et al. (2015) conduisent une étude des échéanciers pour le formalisme PDEVS.

Généralement, l'échéancier est représenté par une file de priorité offrant les opérations de base suivantes : l'*insertion* d'un évènement (*enqueue*) et la *suppression* de l'évènement le plus proche dans le temps (*dequeue*). Au cours de la simulation, l'horloge est avancée jusqu'à la date du prochain évènement planifié dans l'échéancier. Dans le cadre du formalisme PDEVS, le traitement d'un évènement imminent (*dequeue*) peut entraîner l'insertion de nouveaux évènements dans l'échéancier. En effet, dans une simulation de modèles PDEVS, traiter un évènement équivaut à activer un modèle. Celui-ci a alors l'occasion de planifier une prochaine date d'activation. De plus, si il a réalisé des sorties, d'autres modèles auront également une possibilité de réajuster leur future date d'activation. Si l'un de ces nouveaux évènements concerne un modèle dont l'activation est déjà planifiée à travers un évènement

existant dans l'échéancier, deux stratégies existent (Himmelspach et Uhrmacher, 2007b) :

1. L'évènement déjà existant est laissé dans l'échéancier. Il est supprimé de manière normale, lorsque l'horloge de simulation avance jusqu'à sa date (*dequeue*). En conséquence, il est nécessaire de tester la validité de chaque évènement supprimé en comparant la date de l'évènement avec la date d'activation du modèle associé.

2. L'évènement déjà existant est localisé dans l'échéancier puis réajusté (supprimé puis réinséré). Cette approche nécessite une troisième opération de base : la suppression d'un évènement (*delete*).

Bien qu'elle soit plus simple, la première solution entraîne une augmentation de la taille de l'échéancier chaque fois que la date d'activation d'un modèle change. Avec la seconde solution, la taille de l'échéancier est, au pire des cas, égale au nombre de modèles. À l'inverse, la première solution peut entraîner une taille bien plus importante que celle du nombre de modèles. Nous avons opté dans notre outil pour la deuxième solution, privilégiant la mémoire, bien que cela entraîne l'utilisation d'une troisième opération. De plus, les modèles dont la date d'activation est ∞ ne sont pas ajoutés à l'échéancier, ce qui permet de réduire sensiblement la taille de l'échéancier.

Dans Franceschini et al. (2015), nous catégorisons les différentes files de priorités proposées dans la littérature en trois groupes : les files de priorités simples, celles basées sur des arbres et les files de priorités multi-listes. Le tableau 4.3 donne un ordre de complexité en notation $\mathcal{O}$ de Landau des différentes files étudiées. Dans cette étude, nous avons conduit différents tests de performances basés sur un modèle qui permet de capturer tous les aspects pratiques d'une simulation PDEVS (Himmelspach et Uhrmacher, 2007b), pour chaque file et selon différentes distributions statistiques des évènements. Les résultats de cette expérience montrent que les files appartenant à la famille des structures multi-listes obtiennent de meilleures performances et sont relativement stables face aux différentes distributions des évènements. En effet, les files multi-listes offrent un ordre de complexité constant en moyenne, contre un ordre de complexité logarithmique pour les files de priorité basées sur les arbres. Le lecteur intéressé par les conclusions de cette étude peut se référer à l'article. Une partie des tests comparatifs menés dans Franceschini et al. (2015) ont également été effectué pour le cadriciel Quartz (*cf.* annexe C).

Catégorie	**File**	**enqueue**		**dequeue**		**delete**	
		moy.	pire	moy.	pire	moy.	pire
Simple	Minimal list		$\mathcal{O}(1)$		$\mathcal{O}(n)$		$\mathcal{O}(n)$
	Sorted list	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$		$\mathcal{O}(1)$		$\mathcal{O}(\log n)$
Arbre	Binary heap	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$		$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
	Splay tree		$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Multi-liste	Calendar queue	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
	Lazy queue	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
	Ladder queue	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$

Tableau 4.3 Comparaison asymptotique de la complexité en temps des opérations pour le scénario attendu et le pire scénario.

Comme nous l'avons indiqué en présentant l'architecture de la plateforme *Quartz*, nous laissons la possibilité au modélisateur de choisir l'échéancier utilisé par défaut et pour chaque modèle couplé. Afin de fournir des échéanciers performants, nous avons implémenté la *calendar queue* ainsi que la *ladder queue*. Les files de priorité multi-listes, comme leur nom le suggère, reposent sur plusieurs listes pour gérer l'ensemble des éléments. La plupart d'entre elles ont été imaginées spécialement dans le cadre de la problématique de l'échéancier en simulation à évènements discrets. À ce titre, seules les opérations *enqueue* et *dequeue* sont décrites par les auteurs de ces structures. Or, comme nous avons opté pour la solution consistant à avoir un seul évènement pour chaque modèle afin de réduire les besoins en mémoire, nous avons dû concevoir l'opération de suppression pour chacune des deux structures implémentées. Nous présentons dans la suite de cette section chacune des deux files multi-listes plus en détail et décrivons pour chacune d'elles l'opération de suppression.

4.3.1 *Calendar queue*

Proposée par Brown (1988), la *calendar queue* n'est pas la première structure multi-liste à faire son apparition. L'idée de diviser les éléments de la file est due à la *Two-List* (Blackstone Jr et al., 1981), une file de priorité qui segmente ses éléments à forte priorité dans une liste triée de petite taille et ses éléments à plus faible priorité dans une liste non triée. La *calendar queue*, elle, est inspirée de la manière dont l'Homme planifie ses activités à l'aide d'un agenda. Sa structure se compose d'un tableau de *buckets*, où chaque *bucket* est composé d'une liste chaînée ordonnée d'évènements appartenant à un intervalle de temps. Chaque intervalle associé à un *bucket* représente par exemple un *jour* et l'ensemble des *buckets* représente l'intervalle formé par une *année*. La figure 4.8 donne une représentation de cette structure.

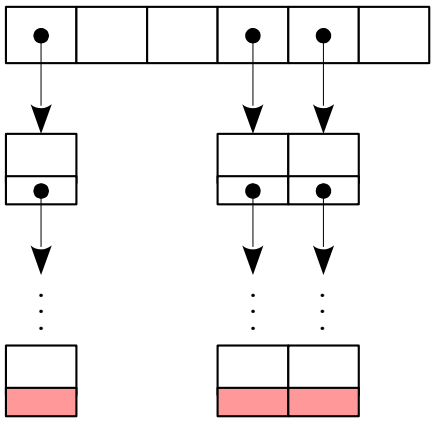


Figure 4.8 Représentation de la structure interne de la file de priorité *calendar queue*.

Lors de l'insertion, un indice indiquant le *bucket* auquel l'évènement est censé appartenir est calculé à partir de la date de l'évènement. Pendant l'opération de *dequeue*, le *bucket* courant est consulté afin d'y rechercher le prochain évènement. Si le jour est vide ou rempli d'évènements prévus pour les années futures, les jours suivants sont consultés jusqu'à trouver le prochain évènement. Lorsque la fin de l'année arrive, on passe à la suivante et on recherche l'évènement à partir du premier *bucket*. Le tableau 4.4 donne un exemple

d'état d'une *calendar queue* retenant un ensemble d'évènements, segmentés en une année de 9 jours. Le jour courant est représenté par le bucket n° 2. Si on applique successivement l'opération de *dequeue*, l'évènement 5.2 est renvoyé, puis 8, 12, etc. Notons que les évènements appartenant à l'année courante sont soulignés, contrairement aux évènements appartenant aux prochaines années.

	Bucket	Intervalle	Éléments
	0	$[2; 3.5[$	$\{15.7, 16.2\}$
	1	$[3.5; 5[$	$\{18.3, 31\}$
☞	**2**	$[\underline{\mathbf{5}}; \underline{\mathbf{6.5}}[$	$\{\underline{\mathbf{5.2}}\}$
	3	$[6.5; 8[$	$\{\emptyset\}$
	4	$[8; 9.5[$	$\{\underline{8}, 22.9\}$
	5	$[9.5; 11[$	$\{\emptyset\}$
	6	$[11; 12.5[$	$\{\underline{12}\}$
	7	$[12.5; 14[$	$\{\underline{12.9}, \underline{13.6}, 27\}$
	8	$[14; 15.5[$	$\{\underline{14.1}, \underline{15}, 28.1\}$

Tableau 4.4 Exemple d'état d'une *calendar queue* retenant un ensemble d'évènements. Ici, l'année est formée de 9 jours (ou *buckets*).

La *calendar queue* tente de préserver une taille raisonnable pour ses *buckets*. Au besoin, c'est-à-dire lorsque la taille de la file dépasse un seuil supérieur ou qu'elle devient inférieure à un seuil inférieur, une opération de *redimensionnement* est déclenchée. Cette opération consiste à doubler ou réduire de moitié le nombre de *buckets* et de ré-ordonner l'ensemble des éléments. Cette opération a une complexité linéaire mais amortie dans le temps et peut être déclenchée depuis n'importe quelle opération de base. Le principal inconvénient de cette file réside dans sa sensibilité aux pics et à l'asymétrie dans la distribution des priorités des évènements. Ceci est principalement dû à la façon dont les intervalles de temps pour chaque *bucket* sont calculés, en échantillonnant un sous-ensemble d'évènements. Pour tenter d'améliorer ses performances, Oh et Ahn (1999) proposent une variante (la *dynamic calendar queue*) dans laquelle une autre méthode d'échantillonnage des évènements est proposée. Tan et Thng (2000) proposent également une variante (la *SNOOPy calendar queue*) basée sur une méthode statistique de calcul des intervalles. Une variante parallèle a aussi été proposée par Santoro et Quaglia (2010).

Comme nous l'avons déjà expliqué, dans le cadre du formalisme PDEVS, l'échéancier nécessite l'opération *delete* qui permet de supprimer n'importe quel évènement donné de la file de priorité. Cependant, l'auteur de la *calendar queue* ne propose pas d'algorithme permettant d'effectuer cette opération. Nous avons défini un algorithme (listing 4.6) permettant de localiser un évènement donné et de le supprimer en temps constant amorti.

De manière similaire à l'opération d'insertion, nous calculons l'indice du *bucket* approprié à partir de la date de l'évènement à supprimer et de la largeur de l'intervalle attribué aux *buckets*. Il s'agit ensuite de parcourir la liste triée jusqu'à trouver l'évènement approprié et de le supprimer de la liste. Si la nouvelle taille de la file est inférieure au seuil déterminé, la file est réduite de moitié et réorganisée.

```
1  calq_entry_t*
2  calq_delete(calq_t* queue, VALUE entry, TIME priority) {
3      calq_entry_t *node;
4      int i;
5      if (queue->size == 0) { return NULL; }
6      // calculate the index of the bucket in which the entry should be
7      i = (int)(priority / queue->bucket_width); // find virtual bucket
8      i = i % queue->bucket_count;               // find actual bucket
9      node = queue->buckets[i]; // grab head of list in appropriate bucket
10     // search node to delete
11     if (node != NULL) {
12         search and delete node holding given entry in the list;
13         --queue->size; // update record of queue size
14         // halve calq size if needed
15         if (queue->size < queue->shrink_thresh) {
16             calq_resize(queue, queue->bucket_count / 2);
17         }
18     }
19     return node;
20 }
```

Listing 4.6 Opération *delete* de la *calendar queue* en pseudocode C

4.3.2 *Ladder queue*

Proposée par Tang et al. (2005), la *ladder queue* voit son nom inspiré de sa structure, rappelant une échelle et ses barreaux. Les éléments sont divisés en trois différents parties :

1. *Bottom*, une liste chaînée ordonnée de taille limitée destinée à retenir les évènements planifiés dans un futur proche.
2. *Ladder*, la couche intermédiaire composée d'un ensemble d'échelons (*rungs*). Chaque échelon est composé d'un tableau de *buckets*, où chacun d'eux est formé par une liste chaînée non ordonnée. Les éléments de cette partie sont destinés à retenir les évènements planifiés dans un futur lointain. Ils ne sont pas triés mais regroupés par intervalles de temps.
3. *Top*, une liste chaînée non ordonnée qui sert de tampon pour les évènements planifiés dans un futur très lointain.

La façon dont la structure divise ses éléments permet de retarder le processus de tri jusqu'à ce qu'il devienne absolument nécessaire, et n'est effectué que sur un sous-ensemble de taille négligeable. Les éléments sont triés uniquement lorsqu'ils sont près à être retirés de la structure, au moment du transfert des éléments d'un *bucket* de l'échelon courant vers la partie basse (*bottom*) de la structure. Ainsi, la plupart des éléments sont soit simplement ajoutés à la partie haute de la structure (*top*), soit ajoutés dans les *buckets* de la couche intermédiaire (*ladder*) sans les trier. L'un des plus grands avantages de la structure réside dans cette couche intermédiaire qui est conçue pour s'adapter automatiquement à la distribution des

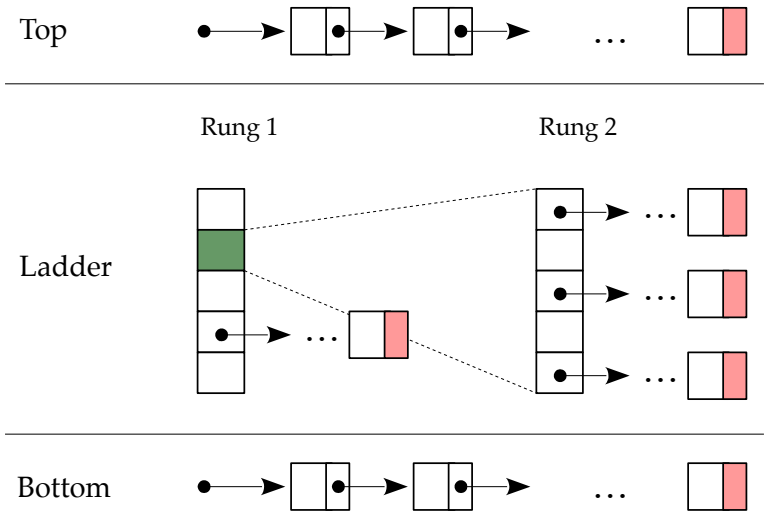


Figure 4.9 Représentation de la structure interne de la file de priorité *ladder queue*.

priorités des éléments. En effet, le nombre d'échelons, appelé *rungs*, varie en fonction de la distribution des priorités, chaque échelon ayant une granularité plus ou moins fine. Cette particularité permet à la structure d'absorber les pics de distribution et de préserver ses performances.

La figure 4.9 donne une représentation permettant de visualiser les trois différentes parties de la structure, où la couche intermédiaire est formée de deux échelons. Considérons le cas suivant : si la *ladder* est censée gérer les évènements inclus dans l'intervalle $[10; 20[$ avec un intervalle de temps de 2, le premier *bucket* du premier échelon contient les évènements compris dans $[10; 12[$, le second les évènements dans $[12; 14[$. Si un pic dans la distribution des priorités implique qu'un nombre trop conséquent d'évènements est ajouté au bucket n° 2, la structure va alors s'étendre en créant un deuxième échelon qui prendra la responsabilité des évènements dans l'intervalle $[12; 14[$, à travers différents buckets. Si 4 buckets sont alloués pour le deuxième échelon, le bucket n° 1 de l'échelon n° 2 aura par exemple la responsabilité des évènements dans $[12; 12.5[$, le second dans $[12.5; 13[$, et ainsi de suite.

Les parties *ladder* et *bottom* sont construites en fonction des évènements se trouvant dans *top*. Ainsi, avant la première opération de *dequeue*, tous les évènements sont ajoutés dans *top*. Lors de l'opération *dequeue*, si la partie *bottom* est vide, une opération de préparation se charge de construire la couche intermédiaire en calculant les intervalles de temps en fonction de tous les évènements dans *top*.

De la même manière que pour la *calendar queue*, les auteurs de la *ladder queue* ne proposent pas d'algorithme pour l'opération *delete*. Nous proposons donc l'algorithme 4.7 pour l'opération de suppression en pseudo code. Dans un premier temps, il s'agit de déterminer dans quelle partie de la structure l'évènement à supprimer se trouve. De manière similaire à l'opération d'insertion, si l'évènement est supérieur ou égal au seuil `top_start`, il a été inséré dans la partie *Top* de la file. Cette liste n'étant pas triée, il est nécessaire

d'effectuer une recherche linéaire jusqu'à trouver l'évènement correspondant. Etant donné qu'avant la première opération de *dequeue*, tous les évènements sont insérés dans la partie haute, nous veillons à préparer la structure de manière similaire à l'opération *dequeue* afin de construire les parties *Ladder* et *Bottom* avant d'effectuer la recherche de l'évènement à supprimer. Ceci nous permet d'éviter une complexité de $\mathcal{O}(n)$ au pire des cas.

Si l'évènement est inférieur à `top_start`, l'évènement se trouve soit dans la partie *Ladder*, soit dans la partie *Bottom*. Les deux cas sont respectivement testés séquentiellement. Ainsi, de manière similaire à l'opération d'insertion, l'indice de l'échelon approprié est calculé, puis l'indice du *bucket* approprié est calculé à son tour. La liste correspondante au *bucket* n'étant pas triée, une recherche linéaire est effectuée pour trouver l'évènement correspondant et le supprimer. Si l'évènement n'a pas été trouvé dans le *bucket*, une recherche est effectuée dans la liste de la partie *bottom*. Notons que cette liste étant triée, si celle-ci est implémentée sous forme de tableau, il est possible d'effectuer une recherche dichotomique en $\mathcal{O}(\log n)$.

```
1  ladq_entry_t* ladq_delete(ladq_t* queue, VALUE entry, TIME priority) {
2      ladq_entry_t *node = NULL;
3      int i = 0;
4      // build the ladder structure and bottom if necessary
5      if (queue->bot_size == 0) { ladq_prepare(queue); }
6      // search in top if the priority exceed top start priority
7      if (priority >= queue->top_start) {
8          node = queue->top;
9          if (node != NULL) {
10             search and delete node holding given entry in top;
11             --queue->top_size; --queue->size; // update queue sizes
12         }
13     } else { /* search in ladder or bottom */
14         // find the appropriate rung in which the entry should be
15         for (i = 0; priority < queue->rungs[i].cur_timestamp && i < queue->
               active_rungs; i++);
16         if (i < queue->active_rungs) {
17             // calculate the index of the appropriate bucket
18             long i = (priority - rung->start_timestamp) / rung->bucket_width;
19             node = rung->buckets[i]; // grab head of list
20             if (node != NULL) {
21                 search and delete node holding given entry in bucket;
22                 // update queue sizes
23                 --rung->bucket_sizes[i]; --rung->size; --queue->size;
24             }
25         }
26         if (node == NULL) { /* fallback search in bottom */
27             node = queue->bottom;
28             if (node != NULL) {
29                 search and delete node holding given entry in bottom;
30                 --queue->bot_size; --queue->size; // update queue size
```

```
31              }
32          }
33      }
34      return node;
35  }
```

Listing 4.7 Opération *delete* de la *ladder queue* en pseudocode C

4.4 *Traitements parallèles*

La simulation parallèle à évènements discrets, en anglais *Parallel Discrete Event Simulation* (PDES), parfois appelée simulation distribuée concerne l'exécution d'une simulation à évènements discrets sur une architecture parallèle. Le domaine de la simulation à évènements discrets a un potentiel important en ce qui concerne la parallélisation des algorithmes, et offre l'occasion de dépasser les limites imposées par la simulation séquentielle, que ce soit en terme de temps d'exécution ou en terme d'espace mémoire. Paradoxalement, comme Fujimoto (1990) le souligne, c'est une problématique difficile à mettre en œuvre.

La PDES concerne plusieurs types d'architectures parallèles : architectures multi-processeurs à mémoire partagée ; clusters de calculs à mémoire distribuée ; machines inter-connectées via Internet ; ou une combinaison de ces possibilités. Il existe plusieurs types d'approches pour tirer parti de ces architectures :

- La forme la plus simple consiste à exécuter sur une architecture parallèle plusieurs occurrences d'une simulation. Cette approche est particulièrement utile pour réaliser une analyse de sensibilité d'un modèle ou pour exécuter un plan d'expérience. Dans les deux cas, un même modèle est exécuté plusieurs fois avec des paramètres d'entrées différents. Aussi, cette approche est utile pour effectuer des réplications de simulations de modèles stochastiques, ou *replicated trials* (Biles et al., 1985 ; Heidelberger, 1986 ; Henderson, 2003), afin de conduire une analyse statistique. Avec cette approche, le gain se situe au niveau de la durée totale de l'expérience ; individuellement, les simulations ne bénéficient pas d'accélération et chaque unité d'exécution doit posséder suffisamment de mémoire pour réaliser une simulation.
- Une autre solution consiste à décomposer certaines fonctionnalités du simulateur et d'assigner chaque tâche à une unité de calcul (Comfort, 1984 ; Davis et al., 1988), comme la génération de nombre pseudo-aléatoires, ou la gestion des évènements par l'échéancier. Cependant, les liens qu'entretiennent chaque composant du simulateur n'offrent pas une grande perspective de parallélisation et peuvent éventuellement entraîner des ralentissements.
- Une troisième approche consiste à exécuter en parallèle les composants d'un système sur une architecture à mémoire distribuée. Des processus logiques évoluent de manière asynchrone, maintiennent leur propre horloge de simulation et sont chargés d'exécuter une partie des composants. Pour maintenir la causalité des évènements, une stratégie de synchronisation est adoptée afin de coordonner les processus logiques. Une stratégie conservatrice (Chandy et Misra, 1979) prévient les erreurs de causalité ; une stratégie

optimiste (Jefferson et Sowizral, 1982) détecte les éventuelles erreurs de causalité et permet de retourner dans un état cohérent.

- Enfin, le formalisme PDEVS offre la possibilité de paralléliser l'algorithme du coordinateur. La gestion du message de collecte et du message interne peut être effectuée en parallèle, ce qui permet de traiter en parallèle les activations simultanés.

Nous avons décidé dans un premier temps de nous orienter vers la première approche de parallélisation, qui est la plus simple à mettre en œuvre. La distribution de simulations séquentielles permet de réduire le temps total d'une expérience numérique pour un effort de développement quasi nul. La norme MPI, pour *Message Passing Interface* (Clarke et al., 1994), permet de faciliter la création de processus sur des architectures parallèles à mémoire distribuée et offre un modèle basé sur la communication par message pour permettre aux différents processus d'échanger des données. Il est possible d'interfacer [11] notre plateforme avec une implémentation de cette norme. Considérons un plan d'expérience représenté par un fichier au format CSV dans lequel chaque ligne représente une condition expérimentale, et où chaque condition est un jeu de paramètres pour le modèle à simuler. Le script représenté par le listing 4.8 est un exemple de l'utilisation de *Quartz* et de MPI permettant d'exécuter une simulation pour chaque condition expérimentale.

```
1  experiment = File.open(ARGV.first, "r")
2  states = [] of SampleModel::State
3  CSV.each_row(experiment) { |row|
4    states << SampleModel::State.new(gamma: row[0].to_f64)
5  }
6
7  MPI.init do |universe|
8    rank = universe.world.rank
9    size = universe.world.size
10   jobs_count = states.size / size
11   start_index = rank * jobs
12   jobs_count += states.size * size if rank == size - 1
13   job = 0
14
15   model = MyModel.new(name: "model")
16   sim = Quartz::Simulation.new(model, duration: 8)
17
18   states[start_index, jobs_count].each do |state|
19     model.initial_state = state
20     sim.restart unless sim.ready?
21     sim.simulate
22     job += 1
23   end
```

11. *Bindings* MPI, développés par nos soins pour le langage Crystal, disponible à l'adresse https://github.com/rumenzu/mpi.cr.

```
24 end
```

Listing 4.8 Exemple de script utilisant MPI pour réaliser une distribution de simulations en fonction d'un plan d'expérience.

Les différentes simulations peuvent ensuite être exécutées en parallèle grâce à l'implémentation de MPI. La commande suivante permet de lancer le plan d'expérience sur 32 processeurs (à condition d'avoir une architecture matérielle adaptée) :

```
mpiexec -n 32 mymodel experiment.csv
```

Il existe également des outils spécialisés dans la distribution de simulations, comme OpenMOLE (Reuillon et al., 2013), dont l'objectif est de faciliter l'exploration de paramètres ou l'analyse de sensibilité d'un modèle. Ces derniers sont conçus pour s'interfacer avec n'importe quel simulateur et peuvent être également utilisés avec Quartz.

Notons que nous avons également exploré la quatrième solution de parallélisation qui a été présentée et qui consiste à exécuter en parallèle les modèles PDEVS dont l'activation est imminente, pour les phases de collecte de sorties et d'activation des transitions. Globalement, l'approche implémentée consiste à partitionner les activations simultanées en fonction du nombre de processeurs physiques disponibles. Cependant, nous avons observé une accélération uniquement pour un nombre important de modèles imminents, et lorsque les calculs réalisés par les fonctions de transitions sont importants. Pour cette raison, nous ne détaillons pas davantage cette approche dans ce manuscrit.

5 *Résumé du chapitre*

Dans ce chapitre, nous avons décrit l'architecture et une partie des fonctionnalités offertes par le cadriciel Quartz que nous avons développé au sein de l'équipe. Basé en grande partie sur la plate-forme DEVS-Ruby (Franceschini et al., 2014c), Quartz permet d'allier l'expressivité de la syntaxe de Ruby aux performances d'exécution. Nous avons proposé un état de l'art sur les différents outils de simulation basés sur le formalisme DEVS afin de mieux situer notre travail.

Les objectifs principaux de cette plateforme étaient de faciliter la modélisation de systèmes multi-agents basés sur la spécification DPDEMAS que nous avons proposée dans le chapitre III. Etant donné que la spécification encourage l'utilisation d'extensions du formalisme PDEVS adaptées aux caractéristiques de chaque système multi-agents, nous avons proposé une architecture modulaire favorisant son extensibilité, en accord avec le principe de séparation explicite de la modélisation et de la simulation. Au delà de permettre la simulation de modèles PDEVS, nous avons également inclus l'extension DSDE, qui permet d'effectuer des simulations de modèles ayant une dynamique structurelle. Nous avons également inclus les extensions CellDEVS (Wainer, 2000) et multiPDEVS (*cf.* annexe B) afin de fournir une implémentation d'environnements cellulaires. CellDEVS permet en effet de représenter des environnements cellulaires et/ou distribués, et l'approche non-modulaire

de multiPDEVS constitue également une extension adaptée à la définition d'environnements cellulaires. L'ensemble de ces extensions nous ont permis de réaliser une bibliothèque annexe permettant de faciliter la définition de SMA basés sur DPDEMAS. Le modélisateur peut ainsi bénéficier d'abstractions logicielles spécifiques au domaine des SMA. Il bénéficie également de composants génériques, dont l'essentiel du comportement est défini, et qui facilitent le passage d'un modèle conceptuel vers un modèle exécutable.

Dans un souci d'évolutivité, nous avons également intégré à l'architecture du cadriciel différents mécanismes permettant de réagir à des évènements. Ainsi, l'observation des états et des ports, la souscription à des notifications, la possibilité de réaliser différents modes d'exécution (étape par étape, aussi vite que possible) et la possibilité de visiter la hiérarchie de modèles permettent d'envisager l'intégration de Quartz dans un environnement complet de M&S, avec des outils de visualisations génériques. Dans l'objectif de réduire les difficultés liées au développement de modèles, nous avons décrit une partie des différentes expressions que nous avons mis en œuvre afin de construire un langage dédié textuel interne. Nous avons également proposé des composants dédiés à la vérification de propriétés concernant le modèle durant son exécution. Cette fonctionnalité, alliée aux possibilités de vérifications statiques que nous avons décrites offrent des perspectives d'évolutions intéressantes.

En gardant à l'esprit le fait que les modèles de systèmes multi-agents puissent nécessiter un nombre important de ressources, nous avons consacré une partie de nos efforts dans l'optimisation d'un certain nombre d'aspects du simulateur, comme la mise à plat de la hiérarchie de modèles, une architecture séquentielle d'exécution prévisible ainsi que l'implémentation de structures de données adaptées à la construction d'un échéancier.

Au cours des précédents chapitres, nous avons proposé une méthode permettant de définir de manière informelle un système multi-agents. Celle-ci a été associée à une méthode permettant de décrire formellement le SMA, basée sur l'utilisation de DPDEMAS. Nous avons enfin développé une plateforme de M&S permettant de faciliter la traduction du modèle conceptuel en modèle exécutable. Dans le prochain chapitre, nous proposons plusieurs cas d'études permettant de mettre en œuvre nos différentes propositions. Ceux-ci seront l'occasion d'avoir une vue d'ensemble de l'approche proposée dans ce manuscrit.

Chapitre V

EXEMPLES DE MODÉLISATION

Sommaire

1 *Introduction*

Nous proposons dans ce dernier chapitre deux exemples de modélisation, afin de mettre en œuvre l'approche que nous avons décrite tout au long de ce mémoire. Ces différents cas d'études sont basés sur des modèles classiques bien connus, qui nous permettent d'illustrer différents aspects d'un SMA et de la spécification DPDEMAS que nous avons définie. Pour chaque exemple, nous employons les méthodes de spécification informelle et formelle proposées dans le chapitre III avant de montrer les résultats de l'implémentation réalisée avec la plateforme Quartz.

Dans le premier exemple, nous définissons un système de type proie/prédateur afin de montrer le caractère dynamique d'un SMA spécifié à l'aide de DPDEMAS. Cet exemple est également l'occasion de montrer les possibilités d'évolution asynchrone des agents avec le paradigme à évènements discrets. Au travers du second exemple, nous proposons de définir et d'implémenter un modèle classique, SugarScape (Epstein et Axtell, 1996). Cet exemple permet de mettre en avant d'autres aspects de la spécification, comme la possibilité de définir d'autres topologies d'environnement, ainsi qu'une dynamique environnementale. Le fait de se baser sur un modèle bien connu est également l'occasion de confronter les

résultats obtenus avec ceux d'une autre implémentation. Ceci nous permettra de souligner l'importance de l'ordonnancement dans la spécification d'un modèle ainsi que la flexibilité de la spécification DPDEMAS à ce sujet.

2 *Modèle proie/prédateur*

Cet exemple a pour objectif d'illustrer la manière dont un modélisateur peut concevoir un système multi-agents basé sur notre spécification à travers un modèle simple. Nous formulons le système d'après les méthodes proposées dans le chapitre III.

Globalement, le système est une version spatialisée issue du système d'équations de Lotka (1925) et Volterra (1926). La variante que nous allons décrire n'a pas vocation à fournir des résultats équivalents au système d'équations fournit par les auteurs. Nous spatialisons le phénomène de prédation ainsi que le phénomène de reproduction, et développons nos propres règles, inspirées par la façon dont les mammifères se reproduisent.

Une première sous-section nous permet de décrire le système de façon informelle avant de définir le modèle conceptuel de manière formelle. Une troisième sous-section présente rapidement les résultats de l'implémentation du modèle.

2.1 *Définition du SMA*

Un paragraphe est associé à chaque étape de la méthode, dans lequel nous décrivons les éléments du SMA.

Représentation du temps

Afin d'illustrer les possibilités d'évolution à évènement discret, nous basons le système sur une évolution asynchrone des agents. L'unité de temps n'a ici pas d'importance, le système modélisé n'étant pas lié à un système réel.

Environnement

Les agents sont plongés dans un environnement physique continu bidimensionnel. La taille de l'environnement peut être définie de façon arbitraire puisque l'unité spatiale ne correspond pas à une échelle réelle. Hormis les agents, l'environnement ne contient pas d'autres ressources particulières. Enfin, les seules variables environnementales concernent la population courante. Une variable permet de donner le nombre d'individus de type « proie » et une autre variable donne le nombre d'individus de type « prédateur ».

Agents

Deux familles d'agents existent au sein de l'environnement physique : les proies et les prédateurs. Bien que les deux familles désignent des espèces différentes, elles partagent les mêmes caractéristiques (statiques ou dynamiques) avec des valeurs différentes. Les propriétés dynamiques comprennent un taux d'énergie, représenté par un entier naturel ;

et une propriété déterminant si l'agent est en cours de gestation. Les propriétés statiques comprennent le genre (mâle ou femelle), le taux de reproduction de l'espèce, le pouvoir calorifique d'un repas type, et la durée de gestation. Un rayon de perception est également déterminé pour chaque espèce. Le genre et la grossesse sont des propriétés observables par les autres agents. Les proies ont une propriété supplémentaire correspondant à la chance d'échapper à une attaque effectuée par un prédateur.

Interaction

Le tableau 5.1 donne la matrice d'interactions, faisant apparaître les différentes actions possibles et les acteurs impliqués. Dans la première colonne cible (∅), nous faisons apparaître les actions impliquant uniquement l'agent source. Ainsi, les deux familles d'agents peuvent se déplacer (*Move*) et mourir (*Death*). Le déplacement n'a pas de pré-condition particulière et a pour effet de modifier la position de l'agent source au sein de l'environnement. Si l'agent source est un prédateur, une unité d'énergie lui est également retranchée. En revanche, la mort est déclenchée si le taux d'énergie de l'agent atteint 0 et a pour effet de faire disparaître l'agent du système.

Le reste des interactions impliquent plusieurs acteurs de même espèce ou d'espèces distinctes. La reproduction (*Mating*) et les naissances (*Birth*) sont des interactions pouvant être émises par les deux familles d'agents et ciblant des agents de la même famille. Afin d'émettre une interaction de reproduction, les conditions suivantes doivent être réunies : l'agent est dans le rayon de perception de l'agent cible, les deux agents ne sont pas du même genre et enfin, la femelle ne doit pas être en cours de gestation. Cette interaction a pour effet d'activer la gestation chez la femelle, et de diviser par deux l'énergie des deux agents. Les naissances sont émises par des agents femelles dont la période de gestation est terminée. Elles ont pour effet d'introduire l'agent cible dans le système et de désactiver la gestation de l'agent source. Enfin, la prédation (*Predation*) est une interaction émise par les agents « prédateurs » et ciblent les agents « proies » lorsque ces derniers sont à portée de perception. Elle a pour effet de faire disparaître l'agent cible du système et d'ajouter le montant du pouvoir calorifique d'un repas type à l'agent source.

Source / Cible	∅	**Proie**	**Prédateur**
Proie	Move, Death	Mating, Birth	
Prédateur	Move, Death	Predation	Mating, Birth

Tableau 5.1 Matrice des interactions proie/prédateur.

Règles environnementales

En plus de respecter les conditions que nous avons établies pour leur déclenchement, pour que les interactions soient valides dans le contexte de l'environnement, elles doivent également respecter les conditions suivantes :

- un déplacement n'est valide que si la position ciblée fait partie du rayon de perception de l'agent ;
- pour chaque prédation, en fonction de la chance de la proie d'échapper à une attaque, un tirage aléatoire détermine la validité de l'interaction ;

- la mort ou la naissance d'un agent est toujours effective si l'agent source fait partie du système.

En ce qui concerne la reproduction, les règles suivantes déterminent si celle-ci a bien lieu :

1. les deux agents impliqués font toujours partie du système ;
2. si il y a un consentement mutuel, la reproduction est validée en fonction d'un tirage aléatoire et du taux de reproduction multiplié par deux ;
3. si l'agent cible souhaite se reproduire avec un autre que l'agent source, la reproduction n'a pas lieu ;
4. si l'agent cible a effectué une interaction de prédation, la reproduction n'a pas lieu ;
5. enfin, dans les autres cas, la reproduction est validée en fonction d'un tirage aléatoire et de la probabilité de reproduction de l'espèce.

Lorsque plusieurs interactions sont réalisées de manière simultanée, leurs validations sont traitées séquentiellement dans l'ordre suivant :

1. Les différentes attaques réalisées par les prédateurs sont prises en compte (*Predation*).
2. La disparition des agents devient ensuite effective (*Death*).
3. Les agents issus des naissances sont créés (*Birth*).
4. Les différentes reproductions sont validées (*Mating*).
5. Enfin, les déplacements sont réalisés (*Move*).

Un certain nombre de conflits peuvent se présenter. Si plusieurs prédateurs souhaitent manger la même proie, un prédateur est privilégié de façon aléatoire. Si plusieurs agents souhaitent se reproduire avec le même agent, un choix est également effectué de manière aléatoire.

Perception

Les agents peuvent observer les autres agents (proies et prédateurs) dont la position est incluse dans leur portée de perception. La portée de perception est déterminée par le rayon de perception qui est une propriété statique définie pour chaque famille d'agents.

Comportement

Le comportement des agents est relativement simple et fait partie de la catégorie des agents réactifs. En fonction de la famille d'agent, le comportement diffère quelque peu :

Proies. Les proies décident de leur prochaine interaction en fonction de leur état courant et des percepts, dans l'ordre suivant :

- si l'énergie de la proie est égale à 0, l'agent meurt ;
- si la proie est une femelle en gestation et que la période de gestation est terminée, celle-ci met bas ;
- si la proie est un mâle ou une femelle qui n'est pas en cours de gestation, la proie recherche une proie de sexe opposé d'après les règles de perception que nous avons définies, afin de se reproduire. Si plusieurs proies sont à portée, la plus proche est sélectionnée. Si la reproduction a lieu, la proie perd la moitié de son énergie ;

- sinon, l'agent se déplace de façon aléatoire.

Prédateurs. Le comportement des prédateurs est assez similaire à celui des proies. Ils décident de leur prochaine interaction selon le même ordre que les proies. En revanche, une nouvelle règle est interposée entre la naissance et la reproduction afin de décider si il est temps de réaliser une interaction de prédation. Le prédateur recherche une proie dans son entourage en respectant les règles de perception définies. Si il y a plusieurs proies potentielles, la plus proche est sélectionnée. Si la prédation a bien lieu, le prédateur incrémente son énergie en fonction du pouvoir calorifique d'un repas type. En ce qui concerne les déplacements, la seule différence qui existe entre la proie et le prédateur est que ce dernier perd une unité d'énergie pour se déplacer.

Pour les deux familles d'agents, afin de refléter le caractère asynchrone de l'évolution du système, les deux planifient leurs actions en fonction d'un tirage aléatoire auquel s'ajoute une unité de temps.

SMA

L'état initial du système est déterminé par celui de l'environnement et des agents. Les états initiaux de ces derniers sont entièrement déterminés par les paramètres d'entrée. Pour l'environnement, trois paramètres d'entrée déterminent l'état initial : la population initiale de proies, la population initiale de prédateurs, puis la taille de l'environnement. En ce qui concerne les agents, le taux initial d'énergie et le genre sont initialisés de manière aléatoire. La position initiale de l'agent au sein de l'environnement physique est également initialisée de manière aléatoire. Les autres propriétés de l'agent sont déterminées à partir des paramètres d'entrée. Le tableau 5.2 donne un récapitulatif des différents paramètres.

Propriété	**Proie**	**Prédateur**
Rayon de perception	$]0, n]$	$]0, m]$
Durée de gestation	$[0, n]$	$[0, m]$
Pouvoir calorifique repas	$[1, n]$	$[1, m]$
Taux de reproduction	$[0, 1]$	$[0, 1]$
Probabilité de s'échapper	$[0, 1]$	$\emptyset$
Énergie	$U(1, 2 * \text{pouvoir calorifique repas})$	
Genre	$U(\text{mâle}, \text{femelle})$	
Gestation	non	non

Tableau 5.2 Récapitulatif des paramètres d'initialisation. $U(a, b)$ indique une valeur dans l'intervalle $[a, b]$ distribuée selon une loi uniforme.

Afin d'observer l'évolution de la population des agents, deux sorties au niveau du SMA correspondant à la population de proies et à la population des prédateurs sont prévues.

2.2 *Spécification formelle*

À partir de la définition du SMA que nous avons donnée dans la section précédente, nous définissons ici le système multi-agents de manière formelle. Nous nous basons pour

cela sur la deuxième méthode que nous avons proposée dans le chapitre III. De la même manière que pour la définition du SMA, nous dédions un paragraphe pour chaque étape de la méthode, dans lesquels nous abordons les différents aspects de la formalisation.

Définition des identifiants

Soit un système multi-agents composé d'agents proies et d'agents prédateurs situés dans un environnement physique :

$$D_{Ag} = Preys \cup Predators$$

où $Preys = \{prey_i \mid 1 \leq i \leq N\}$ et $Predators = \{pred_j \mid 1 \leq j \leq M\}$ avec $N \in \mathbb{N}$ le nombre total d'agents proies et $M \in \mathbb{N}$ le nombre total d'agents prédateurs.

L'ensemble des agents sont situés dans un environnement physique identifié par φ :

$$D_E = \{\varphi\}.$$

Choix du modèle d'environnement

L'environnement étant décrit comme un environnement physique continu bidimensionnel, celui-ci peut tout à fait être représenté par un simple modèle atomique PDEVS. Soit $\varphi \in D_E$ un environnement physique continu 2d défini par le modèle suivant :

$$E_\varphi = (X_\varphi, Y_\varphi, S_\varphi, \delta_{int,\varphi}, \delta_{ext,\varphi}, \delta_{con,\varphi}, \lambda_\varphi, ta_\varphi)$$

où les fonctions de transitions, la fonction de sortie ainsi que la fonction d'avancement du temps, correspondent à celles définies dans la section III.2.3, de la même manière que l'interface d'entrée/sortie correspond à celle qui a été proposée. L'environnement définit un espace $(P_\varphi, dist_\varphi)$ quasimétrique bidimensionnel, où $P_\varphi = \mathbb{R}^2$. La fonction de quasidistance $dist_\varphi$ correspond à la distance euclidienne :

$$dist_\varphi : (x_1, y_1), (x_2, y_2) \mapsto \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Définition des corps

Les propriétés dynamiques observables des agents étant identiques pour les proies et les prédateurs, pour chaque agent $a \in D_{Ag}$, il existe un corps au sein de l'environnement φ tel que :

$$\beta_{a,\varphi} = (p_a, \psi_a, \theta_a)$$

où la position $p_a \in P = \mathbb{R}^2$ est initialisée de manière aléatoire ; $\psi_a \in \wp(P)$ représente la portée de perception/action de l'agent et peut être représentée par une boule[1] ouverte de centre p_a (position de l'agent) et de rayon $r \in \mathbb{R}^+$:

$$\psi_a = \mathcal{B}(p_a, r)$$

avec r la valeur définie par le paramètre d'entrée correspondant au rayon de perception, et qui peut être différent en fonction du type de l'agent ($r = rayon_{prey}$ si $a \in Preys$ ou

$r = rayon_{predator}$ si $a \in Predators$).

La variable θ_a, qui permet de définir des variables supplémentaires au corps de l'agent est définie par :

$$\theta_a = (gender_a, gestate_a)$$

où $gender_a \in \{male, female\}$ correspond au sexe de l'agent, et $gestate_a \in \mathbb{B}$ correspond à l'état de gestation de l'agent, avec la contrainte suivante : $gestate_a = 0$ si $gender_a = male$.

Définition des influences

Pour chaque interaction identifiée dans la matrice d'interaction (*cf.* tableau 5.1), nous définissons un type d'influence. Il existe donc pour un système proie/prédateur, cinq types d'influences : les influences de *déplacement* (Γ_{move}), les influences de *reproduction* (Γ_{mating}), influences de *prédation* ($\Gamma_{predation}$), les influences de naissance (Γ_{birth}) et les influences de mort (Γ_{death}), définies de la manière suivante :

$$\Gamma_{predation} = \{(phase, n, p, a, (a_t))\}$$

où $n \in \{predation\}$, $a \in Predators$, la variable θ est définie par le singleton (a_t) où $a_t \in Preys$ représente l'identifiant de l'agent proie ciblé par l'agent prédateur a. Enfin, la cible de l'influence p est donnée par la position de l'agent proie.

$$\Gamma_{move} = \{(phase, n, p, a, \emptyset)\}$$

où $n \in \{move\}$, $a \in D_{Ag}$ et où p représente la position où l'agent souhaite se déplacer.

$$\Gamma_{mating} = \{(phase, n, p, a, (a_t))\}$$

où $n \in \{mating\}$, $a \in D_{Ag}$, $\theta = (a_t)$ où a_t représente l'identifiant d'un autre agent de la même espèce, avec qui l'agent a souhaite se reproduire. Formellement,

$$a_t \in \begin{cases} Preys \setminus \{a\} & \text{si } a \in Preys \\ Predators \setminus \{a\} & \text{si } a \in Predators \end{cases}$$

Enfin, la cible de l'influence est donnée par la position de l'agent a_t.

$$\Gamma_{birth} = \{(phase, n, p, a, (child, \beta_{child,\varphi}, Ag_{child}))\}$$

où $n \in \{birth\}$, $a \in D_{Ag}$, $child \in D_{Ag}$ est l'identifiant du nouvel agent à faire son entrée dans le système, avec $\beta_{child,\varphi}$ son corps, et Ag_{child} son système cognitif. La cible de l'influence $p \in \mathcal{B}(p_a, r_a)$ correspond à la position de l'agent *child*, initialisée aléatoirement dans la

1. Une boule, en topologie, permet de définir un voisinage dans un espace d'après une fonction de distance. Ainsi, une boule ouverte de centre p et de rayon r dans notre espace quasimétrique correspond à l'ensemble $\{x \in P \mid dist_\varphi(x, p) < r\}$.

portée de perception de l'agent a.

$$\Gamma_{death} = \{(phase, n, p, a, \emptyset)\}$$

$n \in \{death\}$, $a \in D_{Ag}$ est l'agent dont la mort est imminente et la cible p de l'influence est la position de l'agent.

Etat de l'environnement

Comme décrit par la spécification DPDEMAS, l'état S_φ de l'environnement est défini par :

$$S_\varphi = (phase, \sigma, A, (\Xi_\varphi, \theta_\varphi), \Gamma, req^b_\chi)$$

Aucune dynamique environnementale n'ayant été définie pour le système proie/prédateur, $\sigma = \infty$. Aussi, étant donné que tous les agents sont plongés dans l'environnement physique, $A = D_{Ag}$. L'environnement n'étant pas composé de ressources, l'ensemble des entités Ξ_φ est uniquement composé du corps des agents :

$$\Xi_\varphi = \beta_\varphi \cup R_\varphi$$

avec l'ensemble des ressources $R_\varphi = \emptyset$. Afin de définir les variables environnementales correspondant à la population de chaque famille d'agents, nous utilisons la structure θ_φ laissée libre au modélisateur :

$$\theta_\varphi = (pop_{preys}, pop_{predators})$$

où

- $pop_{preys} = \|Preys\|$ est la norme de l'ensemble des identifiants des agents proies ;
- et $pop_{predators} = \|Predators\|$ est la norme de l'ensemble des identifiants des agents prédateurs ;

Calcul de réaction

Nous donnons dans le listing 5.1 ci-dessous la façon dont l'ensemble des influences sont traitées par l'environnement à travers la définition de la fonction de réaction $reac_\varphi$ sous forme de pseudo-code :

```
1  variables:
2    s_φ = (phase_φ, σ_φ, A_φ, Σ_φ, Γ_φ, req^b_χ)
3    Σ_φ = (Ξ_φ, (pop_preys, pop_predators))
4  fonction reac_φ((s_φ,e),Γ'_φ) :
5    morts ← ∅
6    predations ← sélectionner tout γ_i ∈ Γ'_φ où γ_i.n = predation
7    shuffle(predations) // mélanger aléatoirement les influences de prédations
8
9    pour chaque γ_i ∈ predations faire
10     si γ_i.a_t ∈ morts ou random(0,1) > prey_escape_factor alors
11       γ_i.phase ← rejected
```

```
12      sinon
13        morts ← morts ∪γ_i.a_t
14        req^b_χ ← req^b_χ ∪ (del_ag, γ_i.a_t)
15        Ξ_φ ← Ξ_φ \ β_{γ_i.a_t,φ}
16        A_φ ← A_φ \ γ_i.a_t
17        γ_i.phase ← satisfied
18        si γ_i.a_t ∈ Preys alors
19          pop_preys ← pop_preys − 1
20        sinon
21          pop_predators ← pop_predators − 1
22        fin si
23      fin si
24    fin pour
25
26    pour chaque γ_i ∈ Γ'_φ où γ_i.n = death faire
27      si γ_i.a ∉ morts alors
28        req^b_χ ← req^b_χ ∪ (del_ag, γ_i.a)
29        Ξ_φ ← Ξ_φ \ β_{γ_i.a,φ}
30        A_φ ← A_φ \ γ_i.a
31        si γ_i.a ∈ Preys alors
32          pop_preys ← pop_preys − 1
33        sinon
34          pop_predators ← pop_predators − 1
35        fin si
36      fin si
37      γ_i.phase ← satisfied
38    fin pour
39
40    pour chaque γ_i ∈ Γ'_φ où γ_i.n = birth et γ_i.a ∉ morts faire
41      req^b_χ ← req^b_χ ∪ (add_ag, γ_i.child, γ_i.Ag_child)
42      A_φ ← A_φ ∪ γ_i.child
43      Ξ_φ ← Ξ_φ ∪ γ_i.β_{child,φ}
44      si γ_i.a ∈ Preys alors
45        pop_preys ← pop_preys + 1
46      sinon
47        pop_predators ← pop_predators + 1
48      fin si
49      γ_i.phase ← satisfied
50      β_{γ_i.a,φ}.gestate ← false
51    fin pour
52
53    prey_matings ← sélectionner tout γ_i ∈ Γ'_φ où γ_i.n = mating et γ_i.a ∈ Preys et
          γ_i.a ∉ morts
```

```
54  predators_matings ← sélectionner tout γ_i ∈ Γ'_φ où γ_i.n = mating et
        γ_i.a ∈ Predators et γ_i.a ∉ morts
55  shuffle(prey_matings)
56  shuffle(predators_matings)
57  pour chaque γ_i ∈ prey_matings ∪ predators_matings faire
58    valid ← false
59    si γ_i.a ∈ Preys
60      rate ← prey_reproduction_rate
61      match ← sélectionner γ_j ∈ Γ'_φ où γ_j.a = γ_i.a_t et γ_j.phase = pending
62    sinon
63      rate ← prey_reproduction_rate
64      match ← sélectionner γ_j ∈ Γ'_φ où γ_j.a = γ_i.a_t et γ_j.phase = pending
65    fin si
66
67    si match ≠ ∅ alors
68      si match.a_t = γ_i.a // consentement mutuel
69        si random(0,1) < 2 * rate
70          valid ← true
71          match.phase ← satisfied
72        sinon
73          match.phase ← rejected
74        fin si
75      fin si
76    sinon
77      si ∄γ_j ∈ predations où γ_j.a = γ_i.a
78        si random(0,1) < rate
79          valid ← true
80        fin si
81      fin si
82    fin si
83
84    si valid = true alors
85      γ_i.phase ← satisfied
86      si β_{γ_i.a,φ}.gender = female
87        β_{γ_i.a,φ}.gestate ← true
88      sinon
89        β_{γ_i.a_t,φ}.gestate ← true
90      fin si
91    sinon
92      γ_i.phase ← rejected
93    fin si
94  fin pour
95
96  pour chaque γ_i ∈ Γ'_φ où γ_i.n = move et γ_i.a ∉ morts faire
```

```
97        β_{γ_i.a,φ}.p ← γ_i.p
98        γ_i.phase ← satisfied
99      fin pour
100   fin fonction
```

Listing 5.1 Calcul de réaction de l'environnement du modèle proie/prédateur.

Définition des agents

D'après le comportement que nous avons décrit dans la section précédente, les agents peuvent être assimilés à des agents réactifs. À ce titre, une architecture de subsomption est tout à fait adaptée à la représentation de leur système cognitif. Chaque agent, proie ou prédateur, est donc représenté par un modèle PDEVS atomique :

$$Ag_a = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

où $a \in D_{Ag}$ et les ensembles X et Y sont conformes à l'interface d'entrée/sortie définie dans la spécification DPDEMAS pour les agents. Etant donné que les proies et les prédateurs partagent les même propriétés, l'état du système cognitif peut être définie de manière commune :

$$S = \{(energie, bt, \sigma, \gamma)\}$$

où

- $energie \in \mathbb{N}$ est le taux d'énergie de l'agent ;
- $bt \in \mathbb{R}_{+,\infty}$ est le temps restant avant la prochaine naissance ;
- $\sigma \in \mathbb{R}_{+,\infty}$ est la prochaine date d'activation de l'agent ;
- $\gamma \in \Gamma$ est la prochaine influence à envoyer à l'environnement φ.

En ce qui concerne le comportement des agents, le principe est identique. Le système cognitif attend la réception des percepts envoyés par l'environnement, puis réagit en planifiant la prochaine influence à envoyer au modèle d'environnement. Le comportement des fonctions d'avancement du temps, de la fonction de sortie et de la transition interne sont suffisamment génériques pour que leurs définitions soit partagées par les proies et les prédateurs.

Ainsi, la fonction d'avancement du temps renvoie simplement la valeur de σ :

$$ta(s) = \sigma$$

La fonction de sortie dépose sur le port de sortie dédié à l'environnement l'influence qui se trouve dans γ :

$$\lambda(s) = \{(\varphi, \gamma)\}$$

Enfin, la fonction de transition interne permet de basculer dans un état passif afin d'attendre les percepts :

$$\delta_{int}(s) = (s.energie, s.bt, \infty, \emptyset)$$

Toute la logique du comportement est située dans la fonction de transition externe et

diffère en fonction de type de l'agent. Lorsque $a \in Preys$, le comportement associé est donné par le listing 5.2.

```
1  variables:
2    prey_gestation_duration // Durée de gestation des proies
3    rayon_prey  // Rayon de perception des proies
4  fonction δ_ext,a(s,e,x^b) avec a ∈ Preys :
5    body ← chercher β_a,φ dans x^b
6    s.bt ← s.bt - e si body.gestate = true // décrémenter temps restant avant
         accouchement si nécessaire
7    si s.σ ≠ ∞ alors // des percepts ont été reçus avant que l'influence prévue
         ait été envoyée
8      s.σ ← s.σ - e // ne change pas la prochaine date d'activation
9      return // ignorer les nouveaux percepts
10   fin si
11
12   si ∃ dans x^b une influence γ ∈ Γ_mating où γ.a = a et γ.phase = satisfied alors //
         la reproduction a bien eue lieu.
13     s.energie ← s.energie / 2
14     si body.gender = female alors
15       s.bt ← prey_gestation_time
16     fin si
17   fin si
18
19   s.σ ← 1 + random(0,1)
20   candidats ← chercher dans x^b des partenaires prey_i ∈ Preys où β_prey_i,φ.p ∈ body.
         ψ et β_prey_i,φ.gender ≠ body.gender et β_prey_i,φ.gestate = false
21   si s.energie ≤ 0 alors // mort imminente
22     s.σ ← random(0,1)
23     s.γ ← (pending, death, a, body.p) // influence de mort
24   si body.gestate = true et s.bt ≤ 0 // naissance imminente
25     β_prey_N+1,φ ← initialiser corps du nouvel agent
26     Ag_prey_N+1 ← initialiser le système cognitif du nouvel agent
27     pos ← tirer aléatoirement une position ∈ body.ψ
28     s.γ ← (pending, birth, a, pos, (prey_N+1, β_prey_N+1,φ, Ag_prey_N+1)) // influence de
           naissance
29   si body.gestate = false et ||candidats|| > 0 // tentative de reproduction
30     a_t ← sélectionner dans candidats l'agent le plus proche de body.p
31     pos_a_t ← récupérer position de a_t
32     s.γ ← (pending, mating, a, pos_a_t, (a_t)) // influence de reproduction
33   sinon // déplacement
34     pos ← tirer aléatoirement une nouvelle position àune distance rayon_prey de
           body.p
35     s.γ ← (pending, mating, a, pos) // influence de déplacement
36   fin si
```

```
37 fin fonction
```

Listing 5.2 Définition de la transition externe des proies.

Comme nous l'avons décrit dans la section précédente, le comportement des agents de type prédateur est assez similaire à celui des proies. Dans un souci de clarté, nous ne donnons donc pas la définition complète de la fonction de transition externe associée aux agents prédateurs. Nous proposons plutôt de souligner les différences entre les deux variantes de la fonction. Dans un premier temps, les différentes variables relatives aux proies sont remplacées par celles des prédateurs. Pour se reproduire par exemple, les prédateurs recherchent des agents appartenant à l'ensemble *Predators* plutôt que l'ensemble *Preys*. Il convient donc de modifier l'ensemble des éléments de ce type. Deux autres modifications viennent également s'ajouter. La première concerne les lignes 12 à 17 du listing 5.2. En effet, les prédateurs perdent de l'énergie lorsqu'ils se déplacent. Ainsi, il convient de décrémenter le niveau d'énergie lorsqu'un déplacement a eu lieu. De la même manière, le prédateur augmente son niveau d'énergie lorsqu'il a chassé avec succès une proie. Afin de refléter ces changements pour les prédateurs, le listing 5.3 vient s'ajouter à partir de la ligne 17 du précédent listing :

```
1 si ∃ dans x^b une influence γ ∈ Γ_move où γ.a = a et γ.phase = satisfied alors
2   s.energy ← s.energy - 1
3 sinon si ∃ dans x^b une influence γ ∈ Γ_predation où γ.a = a et γ.phase = satisfied
    alors
4   s.energy ← s.energy + predator_food_energy
5 fin si
```

Listing 5.3 Vérification des précédentes influences pour les prédateurs.

La seconde modification concerne la prise de décision de la prochaine influence à réaliser. Juste après la règle de naissance, vient la règle de prédation, définie par le listing 5.4 :

```
1 preys ← chercher dans x^b des proies prey_i ∈ Preys où β_{prey_i,φ}.p ∈ body.ψ
2 si ‖preys‖ > 0 // prédation
3   a_t ← sélectionner la proie la plus proche de body.p
4   pos_{a_t} ← récupérer position de a_t dans x^b
5   s.γ ← (pending, predation, a, pos_{a_t}, a_t) // influence de prédation
6 fin si
```

Listing 5.4 Règle de prédation pour le prédateur.

Définition du SMA

La figure 5.1 donne une vue d'ensemble des différents composants impliqués dans le modèle proie/prédateur. Le SMA est un modèle couplé dynamique DSDE dont le modèle exécutif est donné par la spécification DPDEMAS. L'environnement physique continu φ et les différents systèmes cognitifs des agents sont inter-couplés conformément aux interfaces d'entrées/sorties définies par la spécification. Les sorties au niveau du système multi-agents sont matérialisées par deux ports correspondant à la population de proies et à la

population de prédateurs. Ces ports sont connectés à des sorties spécifiques sur le modèle d'environnement, qui envoie ses sorties dès que la population d'agents évolue. Cet ajout étant trivial, nous ne donnons pas sa définition de manière formelle. Il suffit cependant de déposer la valeur des variables pop_{preys} et $pop_{predators}$ depuis la fonction de sortie de l'environnement pendant la phase *perception* de l'environnement.

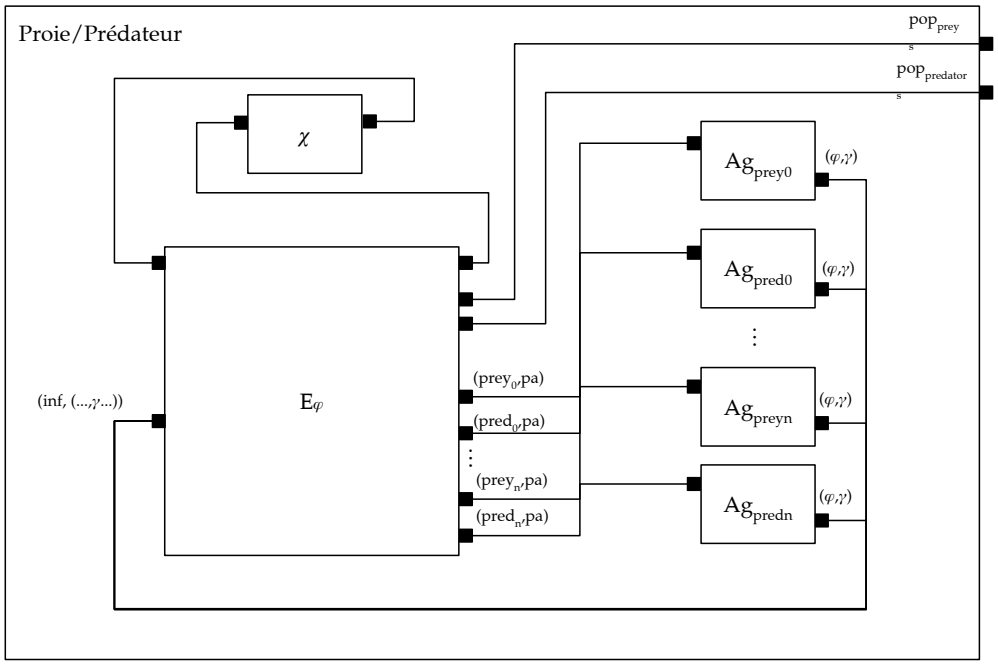
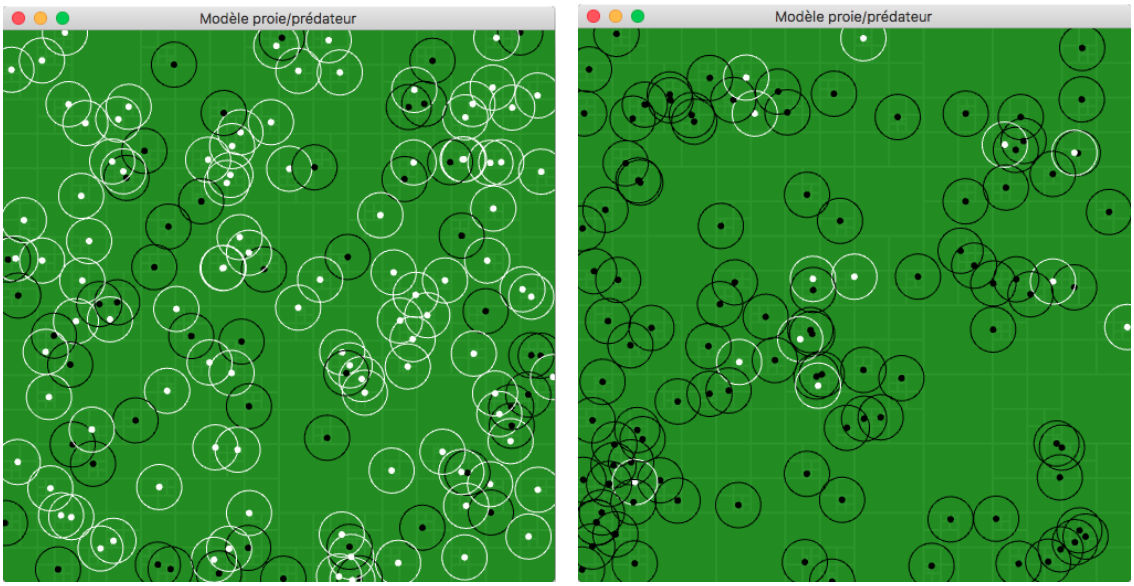


Figure 5.1 Vue d'ensemble du modèle proie/prédateur spécifié à l'aide de DPDEMAS.

2.3 *Exemple d'exécution du modèle*

Nous avons réalisé une implémentation du modèle proie/prédateur que nous venons de définir en utilisant les différentes abstractions fournies par la plateforme Quartz. Les captures d'écran données en figure 5.2 permettent de visualiser l'état de l'environnement à l'état initial (figure 5.2(a)) et pendant son exécution (figure 5.2(b)). Les proies sont représentés par des points blancs et les prédateurs par des points noirs. Pour chacun d'eux, un cercle matérialise la portée de perception associée à l'agent. Cette visualisation de l'environnement permet d'observer l'évolution des agents au cours du temps et a été réalisée en utilisant les mécanismes d'observation de la plateforme Quartz, présentés au cours du chapitre IV.

La figure 5.3 donne les courbes correspondant à l'évolution de la population des proies et des prédateurs au cours du temps de simulation. Ces résultats proviennent d'une seule exécution du modèle, c'est pourquoi nous ne nous y attarderons pas. Nous soulignons cependant que ce modèle est instable, étant donné qu'avec les différents paramètres initiaux que nous avons utilisés, nous avons systématiquement observé l'extinction de l'une des espèces ou une croissance exponentielle des deux espèces.



(a) (b)

Figure 5.2 Captures d'écran de l'état de l'environnement du modèle proie/prédateur.

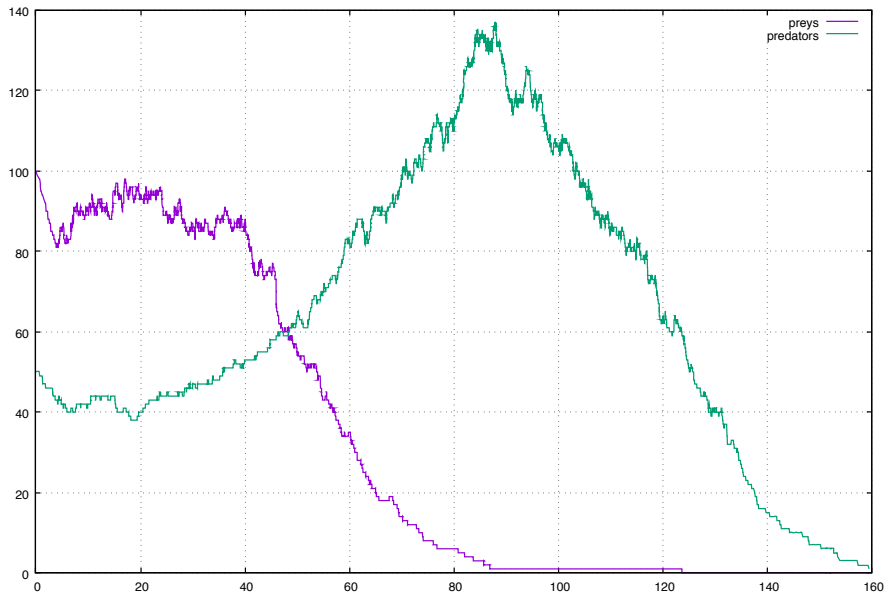


Figure 5.3 Evolution de la population des agents « proies » et des agents « prédateurs » au cours du temps.

Ce modèle nous a permis de donner un exemple concret de la mise en application de notre approche afin de spécifier formellement et d'implémenter un modèle de système multi-agents. La mise en œuvre des différentes méthodes que nous avons proposées nous ont permis de guider la définition des différents aspects du système ainsi que de faciliter la spécification formelle du modèle. Cet exemple nous a permis d'illustrer une partie des possibilités offertes par la spécification DPDEMAS associé à un environnement PDEVS. À travers une dynamique de populations, nous avons pu exploiter la dynamique structurelle du SMA

à travers le mécanisme de requêtes entre l'environnement et le modèle exécutif. Nous avons également montré que l'utilisation du paradigme à évènements discrets offre la possibilité de maîtriser pleinement la façon dont l'ordonnancement des agents s'effectue à travers une évolution asynchrone. De plus, la prise en compte de la simultanéité à travers l'utilisation du formalisme PDEVS et des abstractions inspirées du mécanisme Influence/Réaction montrent qu'il est possible de gérer les actions simultanées des agents, que l'évolution des agents s'effectue à temps discret ou à évènements discrets. Le modélisateur peut ainsi choisir d'utiliser une évolution synchrone lorsque le modèle réalise des abstractions importantes sur le temps (par exemple un déplacement saisonnier des agents), ou une évolution asynchrone qui peut être considérée plus réaliste (Lawson et Park, 2000). Comme nous le verrons avec le prochain exemple, il est important dans les deux cas de maîtriser les conflits qui peuvent se produire.

3 *Modèle SugarScape*

SugarScape est un modèle basé sur le paradigme agent, décrit dans le manuscrit *Growing Artificial Societies* (Epstein et Axtell, 1996). Le modèle est composé d'une population d'agents hétérogène qui se dispute des ressources renouvelables, inégalement réparties dans un environnement bidimensionnel cellulaire (Izquierdo et al., 2009). Différentes règles régissant l'interaction entre les agents et l'environnement sont proposées par Epstein et Axtell (1996). Le modèle gagne alors en complexité graduellement, à travers l'ajout de règles qui permettent aux agents d'enrichir leur comportement (reproduction, commerce, combat), tout en faisant face à de nouvelles situations (pollution, maladie).

SugarScape est un modèle classique au sein de la communauté SMA. Il a été la base de nombreux travaux, visant notamment à étudier la formation de normes à travers la diffusion culturelle (Flentge et al., 2001), ou l'émergence de la coopération entre agents (Buzing et al., 2005). Il a également fait l'objet de reproduction sur de nombreuses plateformes de simulation multi-agents, comme NetLogo (Wilensky, 1999), Repast (North et al., 2013) ou MASON (Bigbee et al., 2007).

Afin de présenter un modèle simple, la variante de SugarScape que nous modélisons dans cette section est basée sur les règles présentées dans le chapitre 2 de l'ouvrage *Growing Artificial Societies*. Globalement, cette variante tient compte uniquement du déplacement des agents et de leur alimentation, du renouvellement de la ressource au sein de l'environnement, ainsi que de la dynamique de la population.

De la même manière que pour le premier exemple, nous suivons les méthodes que nous avons proposées dans le chapitre III afin de donner une définition informelle puis formelle du modèle. Au delà de simplement fournir un second exemple, ce modèle est l'occasion de présenter d'autres aspects de la spécification DPDEMAS, comme la possibilité d'une évolution synchrone des agents, d'une topologie d'environnement différente et de l'introduction de la dynamique environnementale.

3.1 *Définition du SMA*

Comme dans notre premier exemple, pour chaque étape de la méthode, nous associons un paragraphe décrivant les éléments pertinents.

Représentation du temps

L'évolution du temps s'effectue de manière discrète, à pas de temps constants (temps discret). Le système modélisé étant lui même un modèle, l'unité de temps est arbitraire. De la même manière, la durée d'une simulation est variable puisque l'objectif du modèle est de voir apparaître des phénomènes émergents.

Environnement

Les agents sont plongés dans un environnement physique unique bidimensionnel, représenté par une grille de 50x50 cellules. Chaque cellule dispose d'un niveau de sucre, et d'une capacité maximale de sucre, notée c. La capacité maximale de sucre est une valeur entière dans l'intervalle $[0, 4]$, qui est déterminée de manière statique. Le niveau de sucre est une valeur entière dynamique, située dans l'intervalle $[0, c]$. Il peut être consommé par les agents et il est soumis à une dynamique environnementale, définie dans Epstein et Axtell (1996) par la règle G, pour *Growback*. À chaque unité de temps, le sucre « repousse » à un taux de α unités, jusqu'à atteindre la capacité maximale c de la cellule.

Agents

Tous les agents sont du même type et représentent des individus. Chacun d'eux dispose d'un certain nombre de caractéristiques statiques et dynamiques. Les propriétés statiques comprennent la vision, le taux métabolique et l'âge maximal. La vision est une propriété liée à la portée maximale de perception de l'agent, qui peut être plus ou moins importante. Elle représente le nombre maximum de cellules qu'il est possible de percevoir selon la règle de voisinage. C'est une valeur entière comprise dans l'intervalle $[1, 6]$. Le taux métabolique représente le nombre d'unités de sucre que l'agent doit métaboliser à chaque pas de temps pour sa survie. Sa valeur est comprise dans l'intervalle $[1, 4]$. L'âge maximal représente le nombre d'unités de temps maximal durant lequel l'agent peut vivre. En ce qui concerne les variables d'états, chaque agent dispose d'une position au sein de l'environnement, représentée par un point de l'espace géométrique. Les agents ont également la possibilité d'accumuler le surplus de sucre non métabolisé. En conséquence, l'agent dispose d'une variable représentant le stock courant de sucre. L'observation de ces différentes variables par les autres agents n'est pas nécessaire.

Interaction

Comme nous l'avons déjà évoqué, différentes versions du modèle SugarScape existent. Celui-ci est plus ou moins complexe en fonction des possibilités d'interactions intégrées au modèle. Dans notre exemple, nous avons choisi d'intégrer peu d'interactions dans un souci de clarté. Le tableau 5.3 donne la matrice d'interaction et permet de visualiser les acteurs des trois interactions possibles. Nous avons fait apparaître parmi les acteurs l'unique famille d'agents ainsi que la ressource de sucre, celle-ci donnant lieu à une dynamique

environnementale. La seule interaction impliquant deux acteurs est la consommation du sucre par les individus, que nous avons nommée *Eat*. Celle-ci n'est pas liée à une condition particulière puisque tous les individus consomment le sucre disponible à leur position à chaque pas de temps. La « repousse » du sucre (*Growback*) et le déplacement des agents (*Move*) sont de simples actions réalisées sur l'environnement. Ces deux actions ne sont pas soumises à des pré-conditions particulières. La première, *Growback*, étant une dynamique environnementale sans pré-condition, elle a lieu à chaque pas de temps. Nous avons également noté la mort (*Death*) de l'agent comme une action. Notons que la disparition d'un agent peut aussi être considérée comme une conséquence d'une règle environnementale.

Source / Cible	∅	**Sucre**	**Individu**
Sucre	Growback		
Individu	Move, Death	Eat	

Tableau 5.3 Matrice des interactions du modèle SugarScape.

Règles environnementales

Parmi les interactions que nous venons de définir, l'action de déplacement doit réunir un certain nombre de conditions pour être prise en compte. Un déplacement ne peut être effectif que si la cellule visée est libre, et ce déplacement n'est possible que dans la limite de la vision de l'agent, d'après la règle de voisinage que nous décrivons dans le prochain paragraphe. Ainsi, deux agents ne sont pas autorisés à occuper la même cellule au sein de l'environnement.

Epstein et Axtell (1996) précisent que la liste des agents doit être brassée aléatoirement avant d'activer les agents de manière séquentielle. Les auteurs précisent que le modèle peut être exécuté sur une architecture parallèle afin de permettre aux agents de se déplacer de manière simultanée, à condition d'ajouter un mécanisme de gestion de conflit dans le cas où deux agents souhaitent accéder à la même cellule. Face à une telle situation, une règle simple consiste à sélectionner aléatoirement un agent pour lui permettre d'accéder à la nouvelle cellule.

Afin de faire évoluer le modèle, les différentes interactions sont traitées séquentiellement par l'environnement, dans l'ordre suivant :

1. Les agents qui n'ont pas pu métaboliser suffisamment de sucre pour leur survie durant la précédente activation ou qui ont atteint l'âge maximal meurent (*Death*).
2. Les différents déplacements (*Move*) des agents sont pris en compte en suivant les règles que nous avons établies pour les situations conflictuelles (*e.g.* plusieurs agents souhaitent se déplacer vers la même cellule).
3. Les agents consomment le niveau de sucre de la cellule sur laquelle ils sont placés (*Eat*).
4. La dynamique environnementale de régénération de sucre (*Growback*) est appliquée.

Notons que pour chaque mort d'un agent, la règle environnementale de remplacement peut être appliquée. Celle-ci consiste à initialiser un nouvel agent et à l'ajouter à la population

d'agents existante.

Perception

Les agents peuvent observer le niveau de sucre présent sur les cellules qu'ils sont en mesure de percevoir. Un agent peut percevoir les cellules qui l'entourent dans un voisinage de type von Neumann, soit la cellule sur laquelle il est positionné, ainsi que les cellules adjacentes (horizontalement et verticalement). Dans chacune de ces directions, l'agent peut observer le nombre de cellules correspondant à sa propriété de vision. La figure 5.4 illustre la portée de perception des agents.

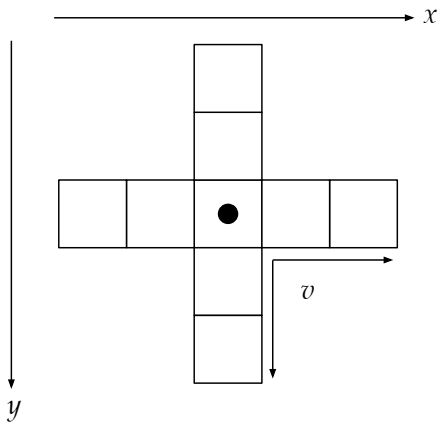


Figure 5.4 Exemple de portée de perception d'un agent de type von Neumann et de portée 2.

Comportement

Le comportement des agents est basé sur la règle de mouvement décrite par Epstein et Axtell (1996). D'après les règles de perception que nous avons décrites, chaque agent cherche à maximiser le niveau de sucre qu'il peut collecter dans les cellules alentours. L'agent identifie les cellules inoccupées (sur lesquelles aucun agent n'est situé) contenant le plus de sucre. Si la valeur la plus élevée de sucre est située sur plusieurs cellules, l'agent sélectionne la cellule la plus proche pour s'y déplacer et y collecter tout le sucre disponible. Si plusieurs cellules correspondent au critère, l'agent en choisi une aléatoirement. Le rayon d'action de l'agent est identique à son rayon de perception. Le sucre collecté est additionné au stock de sucre de l'agent. Afin d'assurer sa survie, l'agent puise aussitôt dans le stock en fonction de son taux métabolique (celui-ci est soustrait au stock). Si le stock est inférieur à 0, l'agent meurt.

La dynamique environnementale, comme nous l'avons évoqué plus haut, concerne uniquement la régénération du sucre. Bien que les auteurs évoquent différentes dynamiques, notamment la régénération immédiate du sucre à la capacité de la cellule, nous utilisons la régénération graduelle, où à chaque unité de temps, le sucre « repousse » à un taux de α unités, jusqu'à atteindre la capacité maximale c de la cellule.

SMA

L'état initial du système multi-agents concerne un certain nombre de propriétés de l'environnement et des agents. Le niveau de sucre initial correspond à la capacité maximale de la cellule. Les différentes capacités des cellules forment une topographie de sucre, qui est décrite par Epstein et Axtell (1996) de la manière suivante : deux « pics » avec une capacité de sucre de 4 sont séparés par une vallée, et sont entourés d'une région désertique, sans sucre. Cette description semble suivre une gaussienne à deux pics. La figure 5.5 montre un exemple de cette distribution de sucre. Les cellules blanches ont une capacité de sucre de 0. Les différents niveaux de sucre sont représentés par une couleur plus ou moins foncée. La couleur la plus foncée a une capacité de 4. L'unité α, servant à la régénération progressive

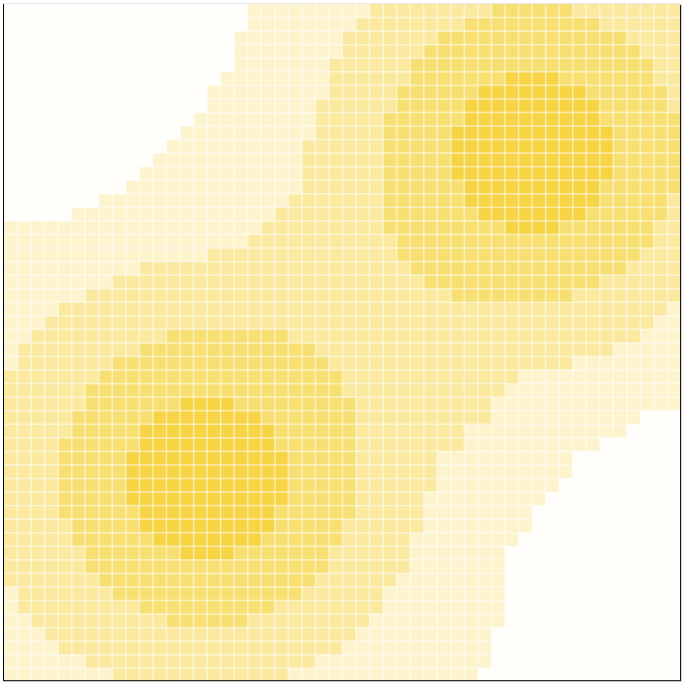


Figure 5.5 Distribution spatiale de la capacité de sucre des cellules dans SugarScape.

de sucre pour la dynamique environnementale, est égale à 1. Le modèle est initialisé avec une population hétérogène de 200 agents. Les variables statiques et dynamiques de chaque agent sont initialisées de manière aléatoire. Ainsi, les agents sont positionnés aléatoirement sur l'ensemble de la grille suivant une distribution uniforme. L'initialisation des autres paramètres de l'agent, c'est-à-dire le stock initial de sucre et les paramètres « génétiques » de l'agent (taux métabolique et vision), sont indiquées dans le tableau 5.4, où $U(a, b)$ indique une valeur dans l'intervalle $[a, b]$ distribuée selon une loi uniforme.

La variante de SugarScape que nous décrivons est utilisée par Epstein et Axtell (1996) pour observer et analyser la distribution du stock de sucre parmi la population d'agents. Les sorties suivantes peuvent être considérées pour chaque cycle de simulation :

- population d'agents ;
- taux métabolique moyen de la population ;
- vision moyenne de la population ;

Description	Initialisation
Stock de sucre des agents à l'état initial	$U(5, 25)$
Taux métabolique initial des agents	$U(1, 4)$
Vision initiale des agents	$U(1, 6)$
Âge maximal	$U(60, 100)$

Tableau 5.4 Paramètres d'initialisation relatifs aux agents pour le modèle SugarScape.

- distribution du stock de sucre à travers la population ;

De la même manière que dans l'ouvrage du modèle, la distribution de sucre à travers la population peut être utilisée pour tracer un histogramme, une courbe de Lorenz (Lorenz, 1905) ou visualiser l'évolution du coefficient de Gini.

3.2 *Spécification formelle*

D'après la définition informelle du SMA que nous venons de donner, nous proposons dans cette section de décrire formellement le modèle.

Identifiants

Soit un système multi-agent composé d'une seule population d'agents D_{Ag} définie par

$$D_{Ag} = \{agent_i \mid 1 \leq i \leq N\}$$

où N est le nombre total d'agents. Ces derniers sont situés dans un unique environnement physique identifié par $grid$:

$$D_E = \{grid\}.$$

Choix du modèle d'environnement

L'environnement étant défini par une grille de cellules, les extensions PDEVS les plus adaptées à ce type de topologie discrète sont CellDEVS ou multiPDEVS (*cf.* annexe A). Ainsi, un modèle basé sur l'une de ces extensions et respectant la définition d'environnement physique donnée en section III.2.3.2 peut être utilisé pour représenter $grid$. Celui-ci définit un espace $(P_{grid}, dist_{grid})$ quasimétrique bidimensionnel, où $P_{grid} = \mathbb{Z}^2$. La fonction de distance utilisée est celle de Manhattan, afin de représenter un voisinage de von Neumann :

$$dist_{grid} : (x_a, y_a), (x_b, y_b) \mapsto |x_b - x_a| + |y_b - y_a|$$

Chaque cellule de l'environnement est associée à une position $p \in P_{grid}$. Les seules entités situées au sein des différentes cellules sont les corps des agents :

$$\Xi_{grid} = \beta_{grid} \cup R_{grid}$$

avec l'ensemble des ressources $R_{grid} = \emptyset$, puisque comme nous le verrons dans la suite de cette formalisation, nous représentons le sucre comme une variable environnementale.

Définition des corps

Pour chaque agent $a \in D_{Ag}$, un corps existe au sein de l'environnement *grid*, défini par :

$$\beta_{a,grid} = (p, \psi, \theta)$$

où la position de l'agent est définie par $p \in P$ avec $P = \mathbb{Z}^2$; aucune variable utile à l'observation des agents n'est définie $\theta = \emptyset$; et enfin, la portée de perception est définie par $\psi = \{p_1, p_2 \in P \mid dist_{grid}(p_1, p_2) \leq v_a\}$ avec $v_a \in [1, 6]$ la vision de l'agent.

Influences

Pour chaque interaction identifiée dans la matrice d'interaction, nous définissons une influence associée. Il existe donc quatre types d'influences : celles de déplacement (Γ_{move}), celles permettant d'amasser une quantité de sucre (Γ_{gather}), les influences de mort et enfin les influences de régénération du sucre.

$$\Gamma_{gather} = \{(phase, n, p, a, (q))\}$$

où $n = gather$, $a \in D_{Ag}$, p est la position de l'agent et $q \in [0, 4]$ la quantité de sucre amassée par l'agent sur la cellule.

$$\Gamma_{move} = \{(phase, n, p, a, \emptyset)\}$$

où $n = move$, $a \in D_{Ag}$ et où p représente la position où l'agent souhaite se rendre.

$$\Gamma_{death} = \{(phase, n, p, a, \emptyset)\}$$

où $n = death$, $a \in D_{Ag}$ et où p représente la position de l'agent.

$$\Gamma_{growback} = \{(phase, n, p, a, \emptyset)\}$$

où $n = growback$, $a = grid$ et où p représente la position de la cellule à régénérer en sucre.

Etat et dynamique environnementale

L'environnement physique dispose de variables relatives à la présence de sucre associées à chaque cellule. Pour définir celles-ci, nous utilisons la structure θ_{grid} laissée libre au modélisateur. Ainsi, $\Sigma_{grid} = (\Xi_{grid}, \theta_{grid})$, avec

$$\theta_{grid} = (sug_{grid}, cap_{grid})$$

où

- $cap_{grid} = \{cap_p \in [0, 4] \mid p \in P\}$ représente l'ensemble des capacités maximales de sucre pour chaque cellule ;
- $sug_{grid} = \{sug_p \in [0, cap_p] \mid p \in P\}$ représente le niveau de sucre d'une cellule.

La dynamique environnementale consiste à régénérer le niveau de sucre des différentes

cellules à chaque pas de temps. Afin d'activer la fonction dédiée à la production d'influences endogènes ($natural_{grid}$), la variable permettant de déterminer la prochaine activation de l'environnement $\sigma_{grid} = 1$. La fonction $natural_{grid}$ est définie de la manière suivante :

$$natural_{grid}(s_{grid}, e) = \bigcup_{p \in P} (pending, growback, p, grid, \emptyset)$$

Calcul de réaction

Nous donnons dans le listing 5.5 la manière dont les influences sont traitées par l'environnement *grid* par le calcul de réaction $reac_{grid}$ sous forme de pseudo-code. Ce calcul de réaction veille à respecter les différentes règles environnementales ainsi que l'ordre dans lequel traiter celles-ci.

```
1  variables:
2    s_grid = (phase_grid, σ_grid, A_grid, Σ_grid, Γ_grid, req^b_χ)
3    Σ_grid = (Ξ_grid, (sug_grid))
4  fonction reac_grid(s_grid, e, Γ'_grid) :
5    deplacements ← regrouper γ_i ∈ Γ'_grid où γ_i.n = move par γ_i.p
6    pour chaque cible, candidats ∈ deplacements faire
7      si ‖ candidats ‖ > 1
8        elue ← sélectionner aléatoirement une influence dans candidats
9        pour chaque influence γ_i ∈ candidats
10         si γ_i = elue
11           β_{γ_i.a,grid}.p ← γ_i.p
12           γ_i.phase ← satisfied
13         sinon
14           γ_i.phase ← rejected
15         fin si
16       fin
17     sinon
18       β_{γ_i.a,grid}.p ← γ_i.p
19       γ_i.phase ← satisfied
20     fin si
21   fin pour
22
23   pour chaque γ_i ∈ Γ'_grid où γ_i.n = gather faire
24     γ_i.q ← sug_{γ_i.p}
25     sug_{γ_i.p} ← 0
26     γ_i.phase ← satisfied
27   fin pour
28
29   pour chaque γ_i ∈ Γ'_grid où γ_i.n = death faire
30     req^b_χ ← req^b_χ ∪ (del_ag, γ_i.a)
31     Ξ_grid ← Ξ_grid \ β_{γ_i.a,grid}
32     A_grid ← A_grid \ γ_i.a
```

```
33      γ_i.phase ← satisfied
34      // replacement rule
35      n_grid ← n_grid + 1
36      req^b_χ ← req^b_χ ∪ (add_ag, agent_n, Ag_agent_n)
37      A_grid ← A_grid ∪ agent_n
38      β_agent_n,grid ← ((rand(0,49),rand(0,49)), agent_n, ψ)
39      Ξ_grid ← Ξ_grid ∪ β_agent_n,grid
40    fin pour
41
42    pour chaque γ_i ∈ Γ'_grid où γ_i.n = growback faire
43      sug_γi.p ← min(sug_γi.p + 1, cap_γi.p)
44    fin pour
45  fin
```

Listing 5.5 Calcul de réaction de l'environnement du modèle SugarScape.

Agents

Le comportement des agents étant relativement simple, nous définissons le système cognitif par un modèle atomique PDEVS :

$$Ag_a = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

où $a \in D_{Ag}$, et l'interface d'entrées/sorties représentée par les ensembles X et Y, est définie telle que dans la spécification DPDEMAS (*cf.* section III.2.2). L'état du système cognitif de l'agent est défini par :

$$S = (m, w, age_{max}, \sigma, inf^b)$$

où

- $m \in [1, 4]$ correspond au taux métabolique de l'agent ;
- $w \in \mathbb{N}$ est le stock de sucre de l'agent ;
- $age_{max} \in \mathbb{N}$ est la durée maximale (en unité de temps) pendant laquelle l'agent peut vivre ;
- $\sigma \in \mathbb{R}_{+,\infty}$ est la prochaine date d'activation
- $inf^b \in \Gamma$ est une variable permettant de définir les prochaines influences à envoyer à l'environnement *grid*.

Le comportement de l'agent s'apparente à une architecture réactive. Le modèle de système cognitif attend la réception des percepts envoyés par l'environnement à chaque pas de temps. Le système cognitif réagit alors en calculant les influences qu'il envoie à l'environnement au prochain pas de temps ($\sigma = 1$). La fonction d'avancement du temps renvoie la valeur de σ :

$$ta(s) = \sigma$$

La fonction de sortie dépose sur le port de sortie dédié à l'environnement les influences qui se trouvent dans inf^b :

$$\lambda(s) = \{(grid, inf^b)\}$$

La fonction de transition interne permet de basculer dans un état passif et d'attendre la réception des percepts :

$$\delta_{int}(s) = (m, w, age_{max}, \infty, \emptyset)$$

L'essentiel du comportement se situe au niveau de la fonction de transition externe. Celle-ci étant moins concise, nous proposons de donner sa définition sous forme de pseudo-code dans le listing 5.6 ci-dessous.

```
1  fonction δ_ext,a(s, e, x^b) :
2    body ← chercher β_a,grid dans x^b
3    si ∃ dans x^b une influence γ ∈ Γ_gather où γ.a = a et γ.phase = satisfied alors
4      s.w ← s.w − s.m + γ.q
5    fin si
6
7    si s.w ≤ 0 alors
8      s.inf^b ← s.inf^b ∪ (pending, death, body.p, a)
9    sinon
10     candidats ← chercher dans x^b les positions p ∈ body.ψ où ∄β_i,grid.p = p
11     sugarmax ← max(...,sug_p,...) ∀p ∈ candidats
12     sweetests ← sélectionner chaque p ∈ candidats où sug_p = sugarmax
13     target ← sélectionner p ∈ sweetests la plus proche de body.p
14
15     si target = body.p
16       s.inf^b ← s.inf^b ∪ (pending, gather, body.p, a)
17     sinon
18       s.inf^b ← s.inf^b ∪ (pending, move, target, a) ∪ (pending, gather, target, a, 0)
19     fin si
20   fin si
21
22   renvoyer s
23 fin
```

Listing 5.6 Définition de la transition externe du système cognitif des agents pour le modèle SugarScape.

SMA

Une vue d'ensemble des différentes entités prenant place au sein du SMA correspondant au modèle SugarScape est donnée en figure 5.6. Conformément à la spécification DPDEMAS, le SMA est un modèle couplé dynamique dont le modèle exécutif (χ) est celui défini par la spécification. Il est composé de l'environnement physique *grid*, lui-même composé d'un composant pour chaque cellule. Les agents sont matérialisés par l'ensemble des systèmes cognitifs.

Afin de récupérer les différentes sorties du modèle SMA, nous utilisons à l'implémentation le mécanisme d'observation d'état du framework *Quartz*. Cependant, afin d'obtenir

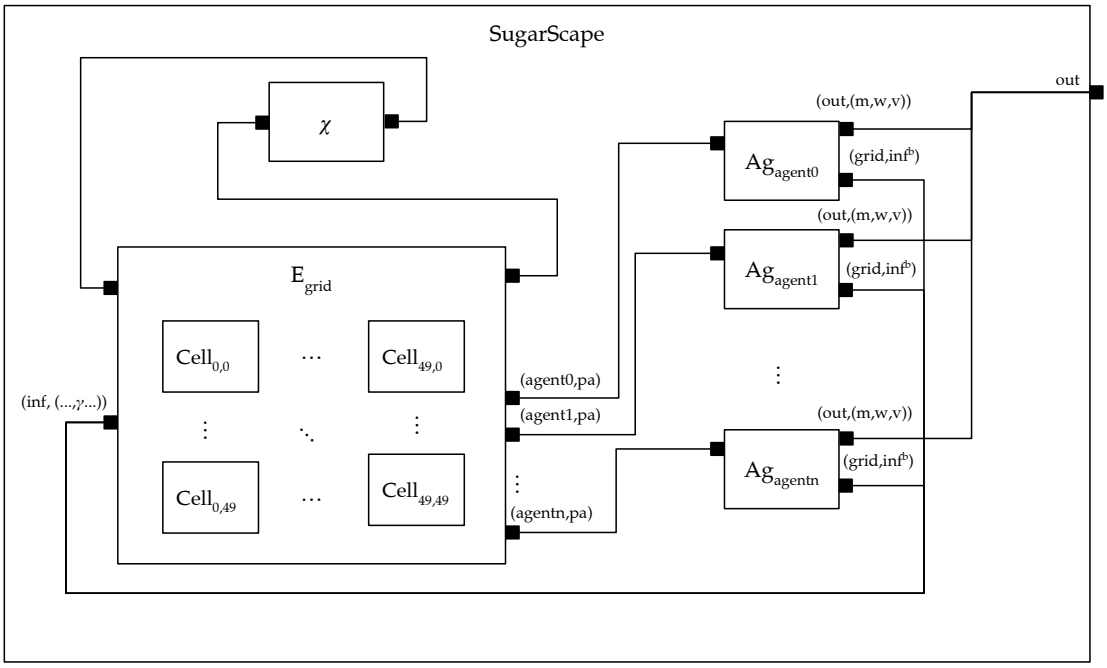


Figure 5.6 Vue d'ensemble du modèle SugarScape spécifié à l'aide de DPDEMAS.

une sortie agrégée au niveau du système couplé, il est possible de définir un port de sortie supplémentaire au modèle de système cognitif, destiné à être lié à un port de sortie du système couplé dynamique. Le modèle de système cognitif peut alors déposer, à chaque activation, les valeurs correspondant au métabolisme (m), à la vision (v) et au stock de sucre de l'agent (w). Cet ajout étant trivial, nous ne le définissons pas de manière formelle. Le nombre de sorties générées par les agents permet de connaître la population d'agents. Cela permet le calcul de la distribution de sucre au sein de la population, la vision moyenne, le métabolisme moyen, l'évolution de la population et enfin, le coefficient de Gini associé à la courbe de Lorenz.

Nous donnons maintenant plus de précisions quant aux états initiaux des agents et de l'environnement. Le nombre total de cellules en abscisse est de $X = 50$, et en ordonnée de $Y = 50$. L'état de chaque agent est initialisé de manière aléatoire. Pour chaque agent $agent_i \in D_{Ag}$, l'état s_0 de son système cognitif ainsi que son corps $\beta agent_i, grid$ sont initialisés de la manière suivante :

$$s_0 = (\text{rand}(1,4), \text{rand}(5,25), \text{rand}(60,100), \infty, \emptyset)$$
$$\beta_{agent_i,grid} = ((\text{rand}(0,X), \text{rand}(0,Y)), \{p_1, p_2 \in P \mid dist_{grid}(p_1,p_2) \leq v_a\})$$

avec $v_a = \text{rand}(1,6)$ et $\text{rand}(a,b)$ une fonction permettant de renvoyer un nombre aléatoire dans $[a,b]$.

La capacité de sucre maximale de chaque cellule peut être décrite à partir d'une gaus-

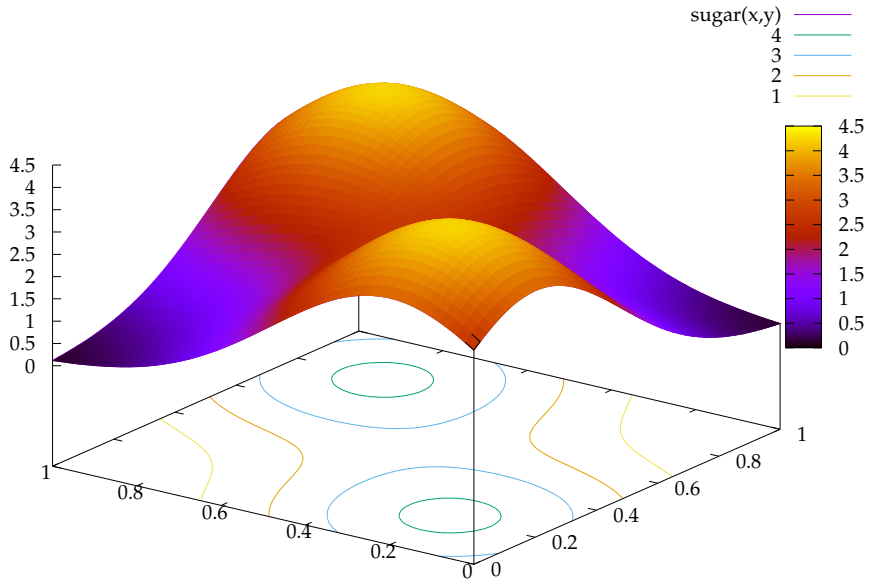


Figure 5.7 Visualisation en trois dimensions de la fonction sugarmap(x, y), permettant de définir la capacité de sucre de chaque cellule.

sienne à deux pics formellement définie par :

$$\text{sugarmap}(x, y) = \text{f}(x - 0.2, y - 0.2) + \text{f}(x - 0.8, y - 0.8)$$

où

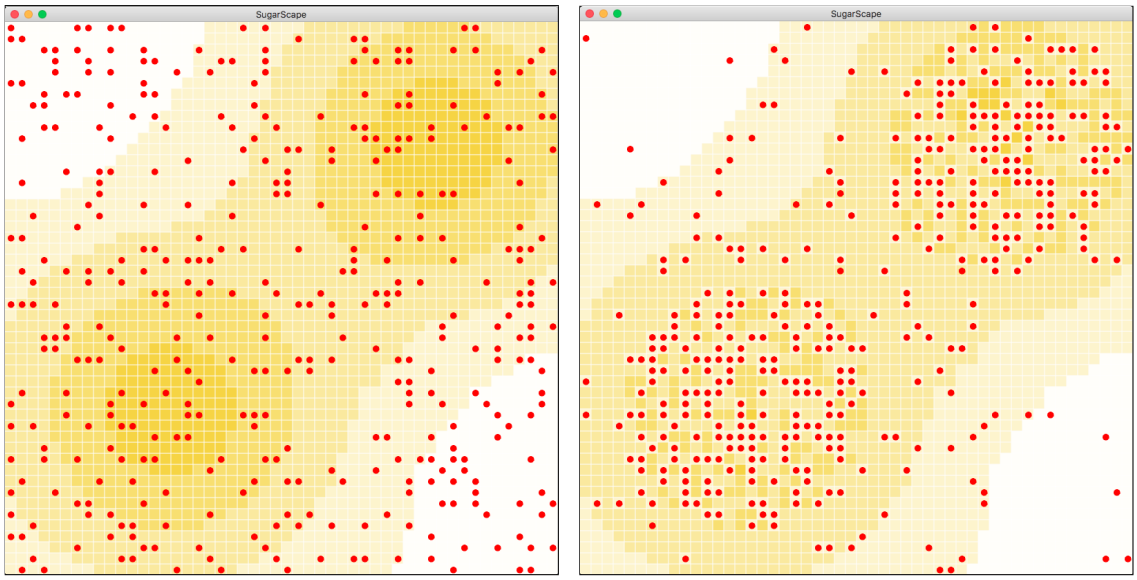
$$\text{f}(x, y) = 4.2 \exp(-(x/0.4)^2 - (y/0.4)^2)$$

Une visualisation de la fonction sugar(x, y) est donnée en figure 5.7. Pour chaque cellule $(x, y) \in \mathbb{Z}^2$ de l'environnement, $cap_{(x,y)} = \lfloor \text{sugarmap}(x/X, y/Y) \rfloor$. À l'état initial, le niveau de sucre $sug_{(x,y)} = cap_{(x,y)}$. Notons que cette fonction a initialement été suggérée par Lawson et Park (2000), pour laquelle nous avons utilisé des valeurs sensiblement différentes.

3.3 *Résultats*

Nous proposons dans cette section de présenter les résultats obtenus à travers l'implémentation du modèle SugarScape, réalisée à l'aide du *framework* Quartz, et dont le code source est donné en annexe D. La figure 5.8 permet de visualiser l'état du modèle SugarScape, tel que nous l'avons défini dans la section précédente. La figure 5.8(a) est une capture d'écran illustrant un état initial possible du système. Le niveau de sucre est représenté par une couleur jaune plus ou moins foncée, et les corps des agents sont représentés par un cercle rouge. La figure 5.8(b), elle, est une capture d'écran de l'état de l'environnement pendant la simulation. Comme on peut s'y attendre, la règle de mouvement des agents entraîne une partie plus importante de la population à se diriger vers les cellules dont le niveau de sucre est le plus élevé.

Les sorties étudiées par Epstein et Axtell (1996) pour cette variante du modèle SugarScape



(a) (b)

Figure 5.8 Visualisation de l'environnement du modèle SugarScape à différents stades d'évolution, sur la plateforme Quartz.

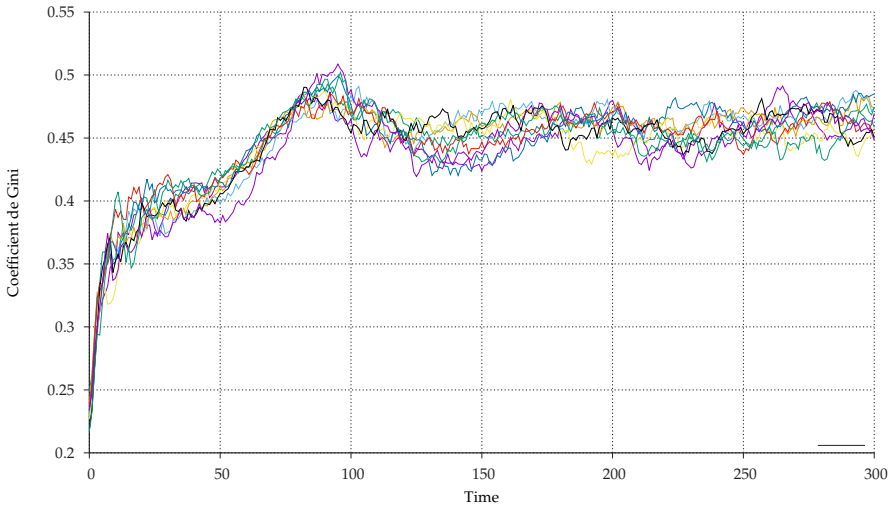
comprennent principalement des indicateurs économiques destinés à analyser la distribution des richesses au sein de la population (*cf.* p.34–38 de l'ouvrage). La plupart des sorties, comme l'histogramme de distribution du sucre ou la courbe de Lorenz, analysent l'état du système à un point précis dans son évolution. Le coefficient de Gini en revanche, permet de montrer l'évolution de la distribution de sucre au cours du temps. Ce coefficient est un indicateur de dispersion dans l'intervalle $[0, 1]$ permettant de calculer la distribution des richesses. Une valeur de 0 correspond à une égalité parfaite (le sucre est uniformément réparti au sein de la population), tandis qu'une valeur de 1 correspond à une inégalité parfaite, où un seul individu détient l'ensemble du stock de sucre.

Afin de donner une tendance de l'évolution de cet indicateur, nous avons réalisé 10 répétitions de simulations sur 300 unités de temps. La figure 5.9(a) permet de visualiser les différentes données générées, où chaque courbe correspond à l'évolution du coefficient de Gini pour une des 10 simulations effectuées. Ces différentes courbes étant difficilement lisibles, nous donnons la tendance de la dynamique via la figure 5.9(b), où est tracée la courbe moyenne ainsi que l'écart-type correspondant à chaque point, afin de visualiser la dispersion des données.

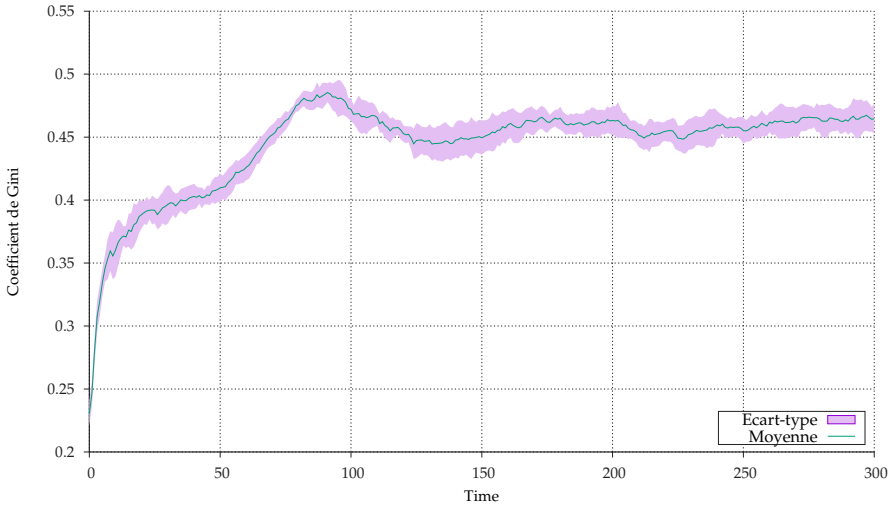
Ces résultats semblent suivre la même tendance que les résultats rapidement décrits dans Epstein et Axtell (1996) :

« *[The Gini coefficient] starts out quite small (about 0.230) and ends up fairly large (0.503).* »

En ce qui nous concerne, le coefficient se stabilise un peu plus bas (~0,465) que le modèle original. Rappelons cependant que pour ce manuscrit, l'objectif de la définition de ce modèle est de fournir un exemple de modélisation permettant de mettre en œuvre la spécification DPDEMAS et les méthodes associées. De plus, cet indicateur semble difficile à reproduire



(a)



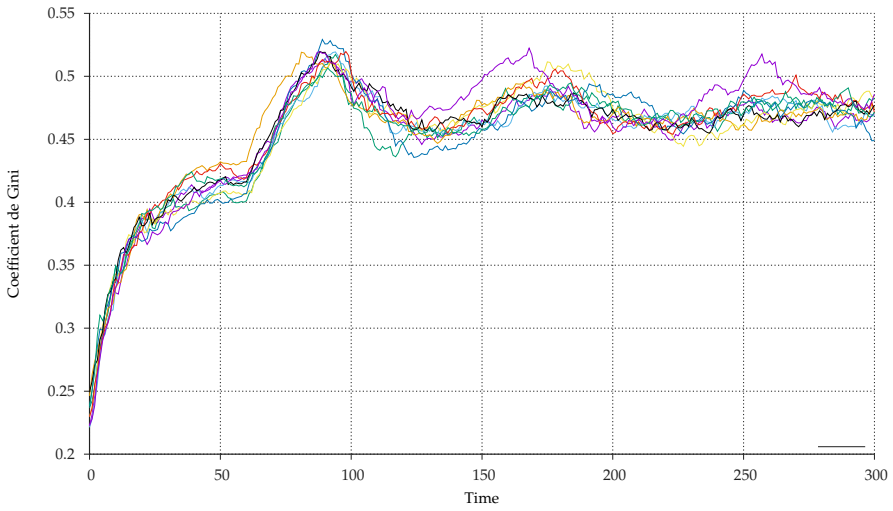
(b)

Figure 5.9 Résultats de simulation montrant l'évolution du coefficient de Gini. En haut (a), le graphe trace les résultats bruts ; en bas (b), le graphe trace la tendance moyenne et l'écart-type.

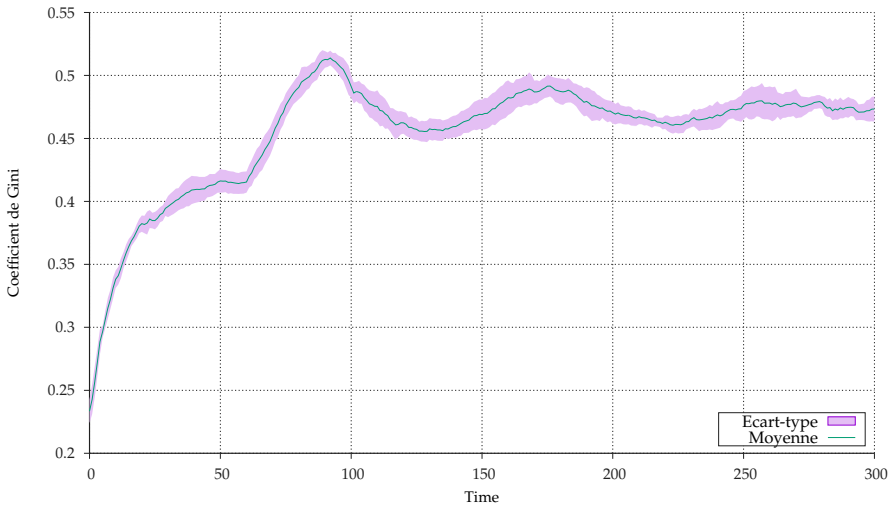
de façon exacte, comme l'expérience de Bigbee et al. (2007) visant à reproduire le modèle SugarScape à l'aide de la plateforme MASON (Luke et al., 2005) le relate. Ces derniers n'ont pas été en mesure de reproduire de façon exacte le coefficient de Gini.

Comme nous l'avons déjà évoqué, le modèle SugarScape a fait l'objet de nombreuses réplications. Afin d'analyser plus précisément les résultats issus de notre modèle, nous avons confronté nos résultats avec des sorties générées par une implémentation du modèle de SugarScape disponible dans la bibliothèque de modèles de la plateforme NetLogo (Wilensky, 1999). Notons que NetLogo met à disposition plusieurs variantes de ce modèle. Nous avons opté pour celle qui se base sur les mêmes règles environnementales que nous avons utilisées pour notre spécification. Cette confrontation nous a semblé intéressante étant donné que NetLogo est basé sur le paradigme à temps discret, avec activation séquentielle des agents

et brassage aléatoire de la liste d'agents (*cf.* chapitre I). Or, comme le soulignent Lawson et Park (2000), des comportements différents peuvent être observés en fonction du mode d'activation. Ainsi, nous avons récupéré les sorties correspondant au coefficient de Gini pour dix répétitions du modèle NetLogo. La figure 5.10 donne les résultats correspondant au modèle NetLogo. À première vue, l'évolution du coefficient de Gini pour l'implémentation



(a)



(b)

Figure 5.10 Evolution au cours du temps du coefficient de Gini pour le modèle SugarScape de NetLogo.

de NetLogo semble proche de notre implémentation. Les courbes moyennes suivent la même tendance. La distribution initiale du sucre (entre 5 et 25) au sein de la population faisant partie des paramètres initiaux, l'indicateur révèle une distribution assez équitable, que ce soit pour notre implémentation ou celle de NetLogo (~0,23). Pour les deux implémentations, des inégalités se créent rapidement (100 premières unités de temps). La première génération d'agents étant graduellement remplacée entre les 60 et 100 premières unités de temps, le coefficient de Gini augmente fortement. En effet, les « nouveaux » agents arrivant dans

le système étant initialisés avec un stock de sucre entre 0 et 25, les inégalités sont plus importantes puisque les individus de la première génération ayant survécus ont amassé un stock important de sucre. Ce pic peut être observé sur les deux jeux de résultats. Une fois la première génération complètement remplacée, les inégalités faiblissent légèrement, et finissent par se stabiliser. Ainsi, après 300 unités de temps, le modèle de NetLogo indique un coefficient de 0,473, tandis que notre implémentation indique un coefficient de 0,464. Afin de comparer les deux moyennes après 300 unités de temps, nous avons utilisé un test t de Student, avec l'hypothèse que les distributions ont des moyennes équivalentes. Nous avons tenu compte de l'homogénéité des variances (homoscédasticité) sur les deux échantillons indépendants (non appariés). Les résultats du test, indiqués dans le tableau 5.5 donnent une probabilité (p-value) de 0,1051, supérieure à la valeur critique des 5%. Nous pouvons donc conclure que les moyennes des deux groupes sont significativement similaires.

Moy. NetLogo	Moy. Quartz	t-value	Degrés de liberté	p-value
0,4733232	0,4644865	1,7068	18	0,1051

Tableau 5.5 Résultats du test t de Student permettant de vérifier l'équivalence des résultats du modèle de NetLogo avec les résultats de notre implémentation.

Notons cependant que cette conclusion n'est valable que pour le coefficient de Gini obtenu après 300 unités de temps. Une analyse des deux courbes moyennes des figures 5.10(b) et 5.9(b) révèle un phénomène oscillatoire du coefficient plus prononcé du côté des résultats de NetLogo. Ce phénomène semble être lié à la durée de vie maximale des agents, puisque les différents pics trouvent leur apogée à la fin de chaque génération d'agents. Rappelons que les agents peuvent mourir de deux façons : de famine (stock de sucre insuffisant pour le métabolisme de l'individu), ou de vieillesse. Nous émettons donc l'hypothèse que le phénomène oscillatoire du côté des résultats de notre implémentation est atténué par un taux de famine plus important. Après avoir ajouté une sortie correspondant au nombre total de pertes d'individus aux deux modèles, nous avons effectivement observé un taux de mortalité plus important sur les résultats de notre implémentation (~600 individus de plus en moyenne).

Cette différence peut s'expliquer par la façon dont les mouvements sont pris en compte entre les deux implémentations. Comme nous l'avons évoqué dans les règles environnementales de la description informelle du modèle, le modèle original brasse la liste des agents aléatoirement après chaque pas de temps. Les agents sont ensuite activés séquentiellement, et chacun d'eux modifie directement l'environnement. Nous avons mis en œuvre l'alternative proposée par Epstein et Axtell (1996) qui consiste à autoriser l'activation parallèle des agents et d'ajouter un mécanisme de gestion de conflit dans le cas où deux agents souhaitent accéder à la même cellule. En donnant ces deux possibilités pour le déplacement des agents, les auteurs ont introduit un artefact[2] dans le modèle.

2. Selon Galán et al. (2009), un artefact est un phénomène pouvant être observé sur les résultats, qui a pour origine des hypothèses jugées (à tort) secondaires par le modélisateur et qui influent les résultats de manière significative.

Dans le cas de l'activation parallèle, les déplacements peuvent mener à des situations conflictuelles, puisque les déplacements ont lieu simultanément. Dans notre variante de SugarScape, chaque agent perçoit le même état de l'environnement au moment de son activation. Il est alors nécessaire de vérifier au moment de traiter les déplacements d'identifier les agents qui visent la même cellule. Dans notre cas, comme nous l'avons décrit dans les sections précédentes, un agent est choisi aléatoirement parmi l'ensemble des candidats. Un seul agent occupe donc la cellule visée, et ceux qui n'ont pas pu se déplacer restent à leur place. La notion de simultanéité des évènements est dans ce cas bien respectée. Dans le cas de l'activation séquentielle, les déplacements sont effectués petit à petit. Cette façon de traiter les évènements simultanés, comme nous l'avions souligné dans le chapitre I, dénature le caractère parallèle des évènements. En effet, un agent qui est activé peut percevoir l'ensemble des modifications qui ont déjà été effectuées par les agents situés en tête de liste. Pour cette raison, dans le cas du modèle SugarScape, les agents peuvent mieux optimiser leurs déplacements. Par exemple, un agent dont la cellule optimale est déjà occupée par un autre agent peut se « rabattre » sur une cellule alternative, peut être moins intéressante en terme de ressource mais éventuellement meilleure que la cellule sur laquelle il se trouve déjà.

Selon nous, le taux de survie plus important pour le modèle NetLogo est principalement dû au fait qu'une activation séquentielle permet aux agents de mieux optimiser leur déplacement lors de l'évolution du système de t à Δt. Afin de permettre aux agents de mieux optimiser leur déplacement avec une activation parallèle, trois solutions peuvent être envisagées :

- Premièrement, il est possible de tirer partie du paradigme à évènements discrets et de laisser la possibilité aux agents qui n'ont pas pu se déplacer sur leur premier choix de réagir plus rapidement afin de cibler une autre cellule.
- Une seconde solution consiste à associer un ensemble de cellules candidates à l'interaction de déplacement, triées par ordre de préférence. La gestion de conflit réalisée par l'environnement peut alors tenir compte de ces choix multiples afin d'éviter de léser les agents.
- La troisième solution est d'adopter une évolution asynchrone des agents de manière similaire à l'exemple du modèle proie/prédateur que nous avons donné dans la première section de ce chapitre et de laisser la règle de conflit telle qu'elle est. Cette solution a été adoptée par Epstein et Axtell (1996).

Afin de vérifier notre hypothèse, nous avons mis en œuvre la première solution en réalisant une variante du modèle. Dans un souci de clarté, nous omettons sa définition, mais l'implémentation de cette variante est donnée en annexe D. Globalement, les modifications concernent le comportement de l'agent. Lorsqu'il reçoit des percepts de l'environnement, celui-ci vérifie si sa tentative de déplacement a échoué, auquel cas il cherche une autre cellule libre à sa portée. S'il en perçoit une qui est plus intéressante que sa position, il établit sa prochaine date d'activation à une valeur ϵ afin d'envoyer une nouvelle influence de déplacement. Dans le cas contraire, il reste à sa position courante (par choix).

Après simulation, il apparaît que cette modification apporte effectivement une baisse du

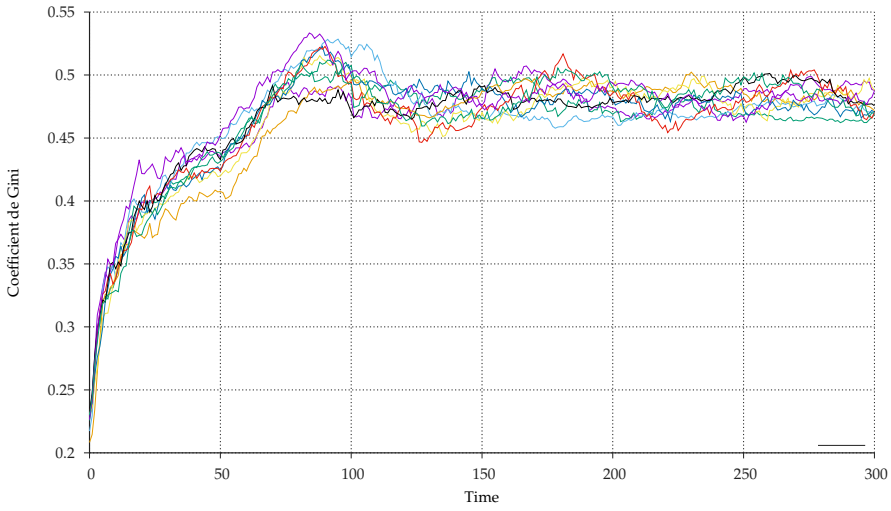
taux de famine, avec toutefois une variance importante. L'indicateur de Gini montre que cette baisse de mortalité a tendance à entraîner une augmentation des inégalités (moins de nouveaux agents arrivent dans le système). Après 300 unités de temps, cette variante du modèle indique un coefficient de Gini de 0,475. Le tableau 5.6 donne les résultats d'un nouveau test T de Student comparant les résultats du modèle de NetLogo avec les résultats produits par notre seconde implémentation. Avec une p-value de 0,5656, cet échantillon est statistiquement beaucoup plus proche des résultats générés par le modèle de NetLogo que la première version du modèle.

Moy. NetLogo	Moy. Quartz 2	t-value	Degrés de liberté	p-value
0,4733232	0,4758203	-0,58527	18	0,5656

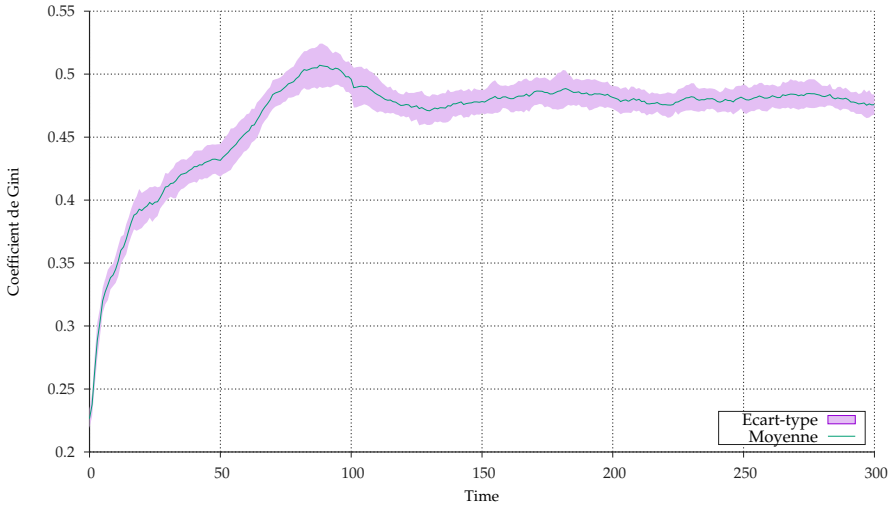
Tableau 5.6 Résultats du test T de Student entre les résultats du modèle de NetLogo et les résultats de la variante de notre implémentation.

En ce qui concerne le phénomène d'oscillation, la figure 5.11 montre l'évolution du coefficient au cours du temps pour la nouvelle implémentation. Une comparaison de la moyenne des courbes entre la première implémentation (figure 5.9(b)) et la seconde (figure 5.11(b)) permet de s'apercevoir que l'amplitude du premier pic de la seconde implémentation se rapproche de celle de NetLogo (figure 5.10(b)). En revanche, après l'extinction de la première génération d'agents, le phénomène oscillatoire a toujours tendance à se stabiliser beaucoup plus vite que le modèle de NetLogo. Cet aplatissement peut s'expliquer par la variance importante du nombre de morts par famine d'une simulation à l'autre dans la deuxième version du modèle. Cependant, notre objectif ici n'étant pas d'aboutir à une reproduction du modèle, nous n'allons pas plus loin dans l'analyse.

Le fait de confronter les résultats de notre modèle avec la version de NetLogo nous a permis de mettre en avant l'importance de l'ordonnancement des agents dans la dynamique du système. Lawson et Park (2000) l'ont d'ailleurs déjà souligné. Ces derniers ont également confronté une implémentation du modèle SugarScape avec les résultats présentés dans Epstein et Axtell (1996). En se basant sur une autre variante du modèle mettant l'accent sur la dynamique de la population, les auteurs notent d'importantes variations entre les résultats du modèle original (synchrone) et de leur modèle (asynchrone). Ils observent un phénomène oscillatoire important sur la version synchrone qui est aplati sur leurs propres résultats. À plusieurs reprises, ils concluent que le phénomène d'oscillations est une conséquence d'un artefact introduit par l'évolution synchrone du modèle. Cette autre expérience révèle une nouvelle fois l'importance de l'ordonnancement des agents. Comme nous l'avons montré dans cette section, l'utilisation de notre spécification permet d'avoir une flexibilité suffisante pour s'adapter à différentes situations d'évolutions des agents, tout en permettant de gérer explicitement les situations de conflit. Nous pensons que ces possibilités facilitent la reproduction de modèles existants tout en permettant la conception de nouveaux modèles. La spécification d'un modèle permet de communiquer de façon non ambiguë les aspects liés à l'ordonnancement et à la gestion des conflits.



(a)



(b)

Figure 5.11 Evolution au cours du temps du coefficient de Gini pour la variante du modèle SugarScape, implémentée avec Quartz.

4 *Conclusion*

Ce chapitre nous a permis de donner une vue d'ensemble de l'approche de modélisation et de simulation que nous proposons pour le paradigme agent. Les deux exemples présentés ont permis de mettre en application les méthodes que nous avons proposées et du caractère générique de la spécification DPDEMAS. Nous avons également pu mettre en œuvre ces concepts à travers un framework de modélisation et de simulation permettant la conception et la simulation de modèles DPDEMAS. Le premier modèle décrit un système proie/prédateur simple et permet de mettre en application la dynamique structurelle du système à travers une dynamique de population. Le second exemple définit un modèle classique, SugarScape, et permet de souligner d'autres aspects de DPDEMAS, comme le processus de

dynamique environnemental. Ces deux exemples nous ont également permis de montrer le caractère générique de la spécification, puisque les deux exemples sont basés sur des environnements distincts, où l'évolution des agents diffère. En confrontant les résultats du modèle SugarScape au modèle équivalent implémenté en NetLogo, nous avons confirmé le fait que l'ordonnancement peut introduire des artefacts dans le modèle. Enfin, les deux exemples nous ont permis de montrer que l'évolution synchrone ou asynchrone des agents est possible en adoptant une gestion des évènements simultanés explicite.

Les deux modèles que nous avons présentés ne nous ont en revanche pas permis d'illustrer toutes les possibilités liées à l'utilisation de la spécification DPDEMAS. Par exemple, la définition d'un modèle basé sur un environnement social nous aurait permis de mettre en application les concepts de groupes, de rôles, ou encore de communication directe. Illustrer l'approche multi-environnement en proposant un modèle composé d'un environnement physique et d'un environnement social aurait donné un exemple intéressant.

Bien que ce chapitre ait été l'occasion d'illustrer notre approche, il est important d'envisager sa validation. Cependant, la validation d'une approche couvrant une partie du processus de M&S et dont une partie des objectifs est de faciliter le partage des modèles dans le but de favoriser leur reproductibilité constitue un exercice difficile. Afin de déterminer si notre approche favorise d'une certaine façon la reproductibilité d'un modèle, nous proposons un plan d'expérience impliquant différents acteurs. Ce plan d'expérience consiste dans un premier temps à fournir différents niveaux de spécification d'un modèle à partir d'un système source et d'objectifs de simulation. Un premier niveau de spécification correspond à la définition informelle du SMA, établie suivant la méthode que nous avons proposée dans le chapitre III. Un second niveau de spécification correspond à la formalisation du modèle conceptuel, en suivant la seconde méthode que nous avons proposée et qui s'appuie sur la spécification DPDEMAS. La seconde étape de l'expérience consiste à impliquer au moins trois modélisateurs différents. Chacun d'eux sera chargé à partir d'un niveau de spécification, de fournir un modèle exécutable basé sur DPDEMAS. La figure 5.12 illustre le plan d'expérience que nous proposons. La traduction du modèle conceptuel en modèle exécutable par l'utilisateur doit être réalisée en utilisant une plateforme de M&S basée sur PDEVS.L'idée est de confronter les résultats fournis par l'exécution de chaque modèle à partir de paramètres identiques, afin de déterminer si les modélisateurs ont été en mesure de fournir une implémentation valide.

Il serait intéressant de réaliser cette expérience sur un modèle original dans un premier temps, et de réitérer celle-ci en se basant sur un modèle de la littérature. Différentes tentatives de reproduction de modèles SMA ont donné lieu à des publications dans lesquelles ont été relatées les difficultés rencontrées (Edmonds et Hales, 2003 ; Rouchier, 2003). Ces modèles représentent des cas d'étude intéressants étant donné que les difficultés concernant leur reproductibilité sont explicitement mentionnées. Par exemple, le modèle d'ethnocentrisme proposé par Axelrod et Hammond (2003) constitue une application intéressante puisqu'il a été reproduit avec difficulté par Wilensky et Rand (2007). Ces derniers ont détaillé leur expérience, et ont indiqué avoir réalisé cinq versions de leur modèle pour arriver à des résultats quantitativement similaires d'un point de vue statistique. Ils avaient pourtant accès

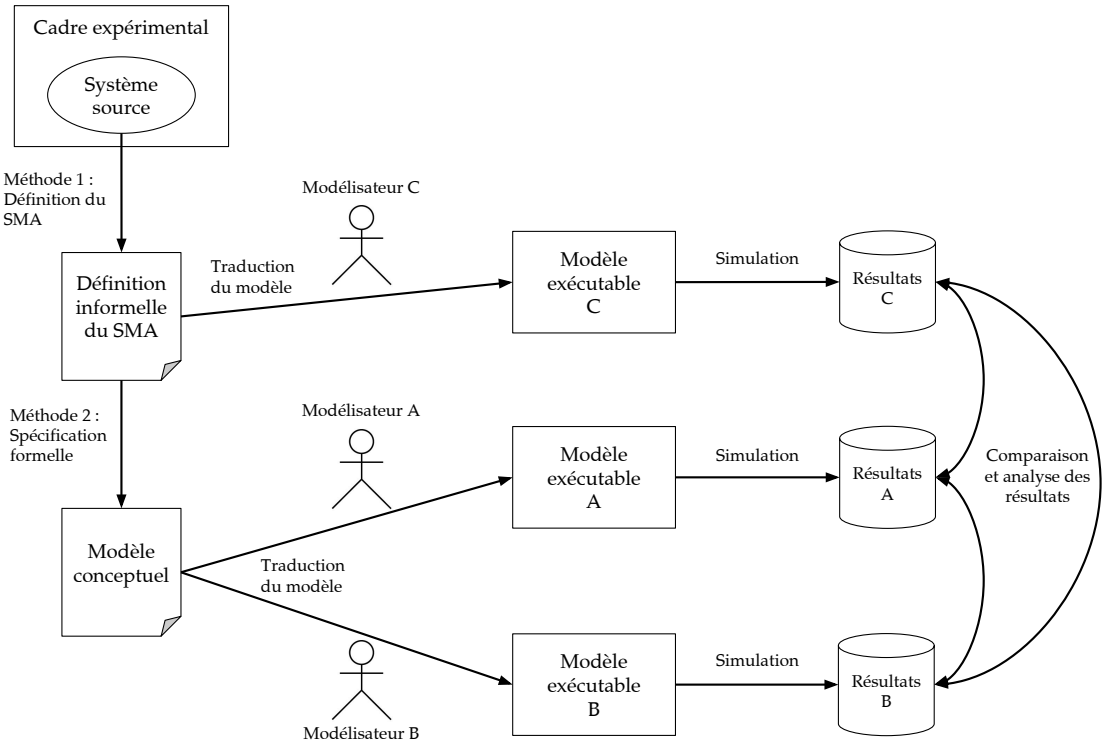


Figure 5.12 Perspective d'un plan d'expérience visant à valider l'approche de M&S proposée, afin de déterminer si elle permet de favoriser la reproductibilité des expériences numériques.

au code source du modèle original, et ont échangé plusieurs fois avec les auteurs du modèle original. Cinq personnes au total ont été impliquées pour arriver à la version finale. À deux reprises, des ambiguïtés au niveau de l'ordonnancement des agents étaient en cause.

CONCLUSION GÉNÉRALE

Un résumé de nos contributions est présenté dans ce chapitre, afin de synthétiser les travaux que nous avons menés. De manière globale, ceux-ci se sont intéressés à la reproductibilité des expériences numériques dans le cadre des systèmes complexes environnementaux, et plus particulièrement dans le cadre de la modélisation de systèmes multi-agents. Pour tenter d'en faciliter la reproductibilité, nous nous sommes dirigés vers une approche formelle, afin de permettre la définition non ambigüe de modèles SMA. Une étude du paradigme agent nous a permis de nous positionner vis-à-vis des concepts qui nous ont semblés les plus adaptés pour répondre à notre objectif. Cette orientation nous a permis de sélectionner un formalisme particulier, à partir duquel nous avons établi notre spécification formelle des SMA. Nous avons associé à cette dernière des méthodes, permettant de guider en partie le modélisateur dans le processus de conception d'un modèle. Pour répondre à notre deuxième objectif, un outil de M&S mettant en œuvre les concepts liés à notre spécification a été proposé, afin de faciliter la modélisation de systèmes multi-agents.

Synthèse

Nous avons dans un premier temps posé les bases de la M&S en étudiant les concepts associés aux systèmes complexes et à leur représentation. Une étude approfondie du paradigme agent, qui est au cœur de notre problématique, nous a permis de dégager les concepts qui nous ont semblé les plus adaptés à la spécification non ambigüe des modèles. Une analyse des facteurs de non reproductibilité nous a permis de mettre en avant le manque de spécification des modèles SMA au sein de la communauté, de l'intérêt de l'utilisation d'approches formelles, ainsi que de la nécessité de la gestion des actions simultanées. Nous avons vu que le principe Influence/Réaction (Ferber et Müller, 1996 ; Michel, 2007a) permet une telle gestion des actions et donc, d'écarter les biais liés à l'implémentation et à l'exécution d'un modèle.

Nous nous sommes alors attachés à étudier les travaux formels déjà existants, et les formalismes adaptés à la représentation de modèles basés sur le paradigme agent. Notre choix s'est porté sur la théorie développée par Zeigler et al. (2000), et plus particulièrement sur le formalisme PDEVS qui permet la description comportementale et structurelle des systèmes, tout en offrant une gestion de la simultanéité de manière similaire au principe Influence/Réaction.

En combinant notre vision du paradigme agent à l'outil formel que nous avons retenu, nous avons proposé une spécification formelle, baptisée DPDEMAS pour *Dynamic Parallel Discrete Event Multi-Agent Specification*. Celle-ci se veut suffisamment générique pour faciliter les interactions entre agents et pour éviter au modélisateur de définir le comportement de certaines entités. Associée à cette spécification, nous avons proposé deux méthodes couvrant une partie du processus de M&S. La première permet de donner une description informelle du SMA ; la seconde s'appuie sur la première pour guider la définition du modèle formel. La spécification, ainsi que ces méthodes, permettent de répondre à notre objectif n° 1 visant à permettre la définition non ambigüe de modèles SMA, afin d'en faciliter le partage et l'implémentation.

Pour faciliter la traduction de la spécification du modèle en modèle exécutable et afin de répondre à notre second objectif qui était de fournir un outil de M&S permettant de faciliter la définition de modèles SMA, nous avons présenté nos travaux d'implémentation, concrétisés à travers la plateforme Quartz. Nous avons conçu son architecture de manière modulaire et extensible afin d'en faciliter l'évolution et l'intégration dans un environnement intégré de modélisation. L'outil implémente les concepts définis par la spécification DPDEMAS, et offre au modélisateur des abstractions logicielles spécifiques au paradigme agent, afin de faciliter le passage du modèle conceptuel au modèle exécutable.

Nous avons finalement proposé de mettre en application l'ensemble de l'approche à travers deux exemples : un premier modèle décrit un système de proie / prédateur ; un second modèle décrit le modèle classique SugarScape. Chacun des exemples a permis d'illustrer des aspects différents de la spécification, et a permis de mettre en application les méthodes que nous avons proposées. Une confrontation des résultats du modèle SugarScape avec une implémentation issue d'une plateforme SMA nous a confirmé l'importance de l'ordonnancement comme biais possible lors de de la définition de modèles SMA. Nous avons conclu par la définition d'un plan d'expérience permettant de quantifier l'apport de notre approche, afin de voir dans quelle mesure celle-ci permet de faciliter la reproductibilité des expériences numériques de modèles SMA.

Perspectives

Tout au long de ce manuscrit, nous avons relevé de nombreuses perspectives de travail visant à améliorer ou compléter notre approche. Nous tâchons ici de développer chacune d'entre elles.

Quantifier l'apport de la méthode

Puisque nous proposons une approche couvrant une partie du processus de modélisation et de simulation, il est indispensable dans le court terme de déterminer dans quelle mesure celle-ci permet de favoriser la reproductibilité des expériences numériques. Nous avons entamé une réflexion sur ce sujet à travers les conclusions du chapitre V, lorsque nous mentionnons un plan d'expérience dont l'objectif est de déterminer si à partir d'une même spécification formelle DPDEMAS, différents modélisateurs sont capables d'obtenir des

résultats équivalents. Nous comptons démarrer cette expérimentation très prochainement, afin de quantifier l'apport de notre méthode.

Définir une méthodologie

À moyen terme, nous souhaitons compléter les méthodes proposées pour fournir une méthodologie adaptée au processus de M&S pour les systèmes multi-agents dans sa globalité. Nous pensons que cette perspective est indispensable en vue de l'adoption de notre approche. Une telle méthodologie permettrait d'identifier les différents acteurs (experts, modélisateurs et informaticiens) impliqués dans la conception d'un modèle pour les guider dans chacune des étapes où ils interviennent.

Dans le but d'accélérer le processus de conception d'un modèle, il serait également intéressant de centraliser les contributions issues des différents acteurs au sein d'un environnement intégré de modélisation. Un tel outil permettrait de réunir l'ensemble des informations relatives à un modèle à différents stades de conception et de faciliter les échanges entre les acteurs. L'intérêt d'une telle centralisation est double. Elle facilite la validation du modèle, et permet d'atteindre plus facilement le niveau de reproductibilité « recherche ouverte » définit par Stodden et al. (2013).

Approche IDM

Un environnement intégré de modélisation est idéal pour mettre en œuvre les concepts liés à l'Ingénierie Dirigée par les Modèles. Nous souhaitons à moyen terme étudier cette approche, qui permettrait d'intégrer le métamodèle proposé à la spécification DPDEMAS indépendamment de toute plateforme, tout en offrant la possibilité d'y associer des contraintes sémantiques sur les SMA (*e.g.* à travers l'utilisation d'OCL). Le principal avantage de l'IDM est qu'il permet d'automatiser plusieurs étapes du processus de M&S. Entre les relations structurelles définies entre les concepts du métamodèle et la définition de contraintes sémantiques, il est possible d'appliquer des vérifications statiques sur les modèles de manière automatisée. Des règles de transformation permettent également de passer automatiquement d'un modèle indépendant d'une plateforme à un modèle dépendant autorisant l'exécution du modèle, réduisant ainsi les biais potentiels associés à la traduction d'un modèle. Cette étape nécessite simplement la définition du métamodèle associé à chaque plateforme cible. Nous pensons que l'utilisation des outils associés à l'IDM permettrait de compléter notre approche visant à faciliter la reproductibilité des modèles et de réduire les biais potentiels associés à chaque méthode. L'approche couvrirait alors la quasi-totalité du processus de M&S, en ajoutant la vérification automatique des modèles et la traduction automatique du modèle conceptuel vers le modèle d'exécution. Son intégration est envisagée à long terme.

Définition d'un DSL

Afin de proposer une approche permettant de définir de manière non-ambigüe des modèles SMA, nous nous sommes orientés vers une approche formelle, visant à définir précisément la structure et le comportement de chaque entité d'un SMA. Dans le cadre de cette thèse nous attribuons aux méthodes formelles la capacité de clarifier la définition d'un modèle. Cependant, au vu de la complexité du paradigme agent et de la longueur des

définitions formelles proposées dans les chapitres III et V, il est légitime de se demander si une approche formelle dans ce cadre permet véritablement d'éclaircir la définition d'un modèle. Le niveau d'expertise nécessaire pour appréhender l'approche proposée, est semble-t-il, beaucoup trop élevé pour permettre son adoption, et donc de réellement favoriser la reproductibilité. Pour pallier à cette problématique, nous sommes convaincus qu'il est nécessaire de concevoir un langage de modélisation spécifique permettant de faciliter la définition des aspects structurels et comportementaux des SMA. Il est possible de concevoir un langage associé aux concepts utilisés dans le cadre de cette thèse, de l'intégrer à notre environnement de modélisation à moyen terme et de réutiliser ce langage pour l'intégrer à une approche IDM à plus long terme.

Application de l'approche à des systèmes environnementaux

On pourra regretter que cette thèse soit restée dans le domaine du théorique, c'est pourquoi l'une de nos perspectives à court terme est d'accompagner l'adoption de l'approche au sein des différentes équipes du laboratoire, qui pourrait trouver à travers l'utilisation des SMA, un paradigme de modélisation capable de répondre aux problématiques étudiées. Ainsi, nous envisageons de travailler sur des domaines d'application tels que la propagation de feux de forêts associés à des stratégies de lutte incendie, ainsi que sur la gestion halieutique, pour la mise en place de politiques de développement durable.

Annexe A

EXTENSIONS DEVS POUR LA DESCRIPTION DE TOPOLOGIES DISCRÈTES

Cette annexe a pour objectif de présenter succinctement deux extensions du formalisme DEVS adaptées à la définition de topologies discrètes : Cell-DEVS et multiDEVS.

1 *Cell-DEVS*

L'extension Cell-DEVS, proposée par Wainer et Giambiasi (2001), permet de simplifier la définition d'automates cellulaires à l'aide du formalisme PDEVS, et forme à ce titre, une extension adaptée pour la définition d'environnements cellulaires pour les SMA. En effet, PDEVS n'est pas particulièrement adapté à la définition d'un nombre important de modèles interconnectés. Il est pour cela nécessaire de spécifier l'ensemble des couplages. Cell-DEVS, en plus de faciliter la définition et la construction de modèles cellulaires, offre une interface permettant de définir le comportement des cellules avec une sémantique propre au domaine des automates cellulaires. Pour cela, de nouveaux éléments viennent s'ajouter à la définition d'un modèle atomique et d'un modèle couplé.

Cell-DEVS considère une cellule comme un modèle atomique, avec néanmoins certaines modifications. La structure du modèle atomique représentant une cellule est définie par la structure suivante :

$$TDC = \langle X, Y, I, S, N, delay, d, \delta_{int}, \delta_{ext}, \delta_{con}, \tau, \lambda, ta \rangle$$

où X, Y, S, δ_{int}, δ_{ext}, δ_{con}, λ et ta sont définis de la même manière qu'un modèle atomique DEVS. Les nouveaux éléments introduits par Cell-DEVS sont les suivants :

- I représente la définition de l'interface de la cellule avec son voisinage, et établit les connexions avec les cellules voisines à travers un port d'entrée et un port de sortie dédié à chaque voisine.
- N est l'ensemble des évènements d'entrée venant des cellules voisines, plus précisément, il est constitué des états binaires représentant l'activité des cellules voisines.

- $\tau : N \to N$ est la fonction de calcul du nouvel état de la cellule en fonction des états des cellules voisines et son résultat indique si la cellule est active ou non.
- $d \in \mathbb{R}_0^+$ est une variable qui représente la durée du délai à attendre pour que le nouvel état de la cellule soit effectif.
- *delay* représente le type de délai à utiliser pour la cellule, qui peut prendre trois valeurs différentes :
  - le délai de type *transport* permet de représenter le temps de propagation d'un signal et signifie que l'état de la cellule met un certain temps pour se propager jusqu'aux cellules voisines ;
  - le délai de type *inertial* permet de modéliser des systèmes qui ont besoin de recevoir une certaine quantité d'énergie pour changer d'état ;
  - le délai de type *rise-fall* permet de modéliser le délai d'attente nécessaire pour qu'un signal passe d'un état haut à un état bas.

En fonction du type de délai de la cellule, l'ensemble des variables d'états S de la cellule peut varier. Toutes les cellules partagent au moins la variable $s \in \{0, 1, \emptyset\}$, un booléen qui représente l'activité de la cellule ; la $phase \in \{active, passive\}$ de la cellule ; $\sigma \in \mathbb{R}_{0,\infty}^+$ qui représente la date à laquelle la cellule s'active, utilisée par la fonction ta ; et éventuellement d'autres variables définies par le modélisateur. Il faut noter qu'en fonction du type de délai utilisé, le nouvel état de la cellule peut être envoyé de manière instantanée aux cellules voisines ou après la durée du délai d'attente. Rappelons que les sorties des modèles PDEVS sont envoyées de manière instantanée après la fonction de transition interne. Si cela convient à un type de délai inertiel, un type de délai de transport nécessite en revanche l'utilisation d'une file d'attente permettant de stocker l'état de la cellule avant de le propager après la durée du délai de transport.

Etant donné que les automates cellulaires sont constitués de cellules identiques qui diffèrent uniquement par leur position dans l'espace, et qu'il est fastidieux de définir l'ensemble des couplages pour chaque cellules, Cell-DEVS propose une modification de la structure du modèle couplé afin de faciliter la définition de l'espace (nombre de dimensions et nombre de cellules) et du voisinage de chaque cellule. La structure du modèle couplé est définie de la manière suivante :

$$GCC = \langle X_{list}, Y_{list}, I, X, Y, n, \{t_1, \ldots, t_n\}, N, C, B, Z \rangle$$

où X_{list} et Y_{list} représentent l'ensemble des cellules qui possèdent des ports d'entrée ou de sortie vers des modèles extérieurs au réseau de cellules. Les cellules appartenant à X_{list} possèdent donc des couplages d'entrées externes et les cellules appartenant à Y_{list} possèdent des couplages de sorties externes. Ces deux ensembles sont utilisés par I, qui définit l'interface du modèle couplé. Les ensembles X et Y ont la même sémantique qu'avec un modèle couplé DEVS classique. Afin de faciliter la définition de l'espace de cellules, n permet de spécifier la dimension de l'espace ; t_i représente le nombre de cellules présentes dans la $i^{ème}$ dimension où $\forall 0 < i \leq n$. Afin de faciliter la définition du voisinage pour chaque cellule, l'ensemble N permet de définir le motif de voisinage en spécifiant des coordonnées relatives (*e.g.* $\{(-1, 0), (0, -1), (0, 0), (1, 0), (0, 1)\}$ pour un voisinage de Von Neumann dans

un espace à deux dimensions). C est l'ensemble des cellules et B est l'ensemble des cellules situées aux bordures de l'espace.

Cell-DEVS propose donc deux définitions de modèles. Le modèle atomique permet de représenter la dynamique de la cellule et le modèle couplé permet de définir la structure de l'automate cellulaire avec la sémantique du domaine.

2 *Multi-component DEVS*

Dans leur ouvrage, Zeigler et al. (2000) font remarquer que certains systèmes sont plus faciles à modéliser en identifiant des entités simples, en les modélisant indépendamment avant de modéliser la manière dont elles interagissent plutôt que de décrire leur comportement directement. Cette remarque nous renvoie directement aux différentes approches de conception descendantes et ascendantes (*cf* section 2.3).

Dans un système modélisé avec le formalisme DEVS, l'interaction entre composants s'effectue en connectant les interfaces d'entrée et de sortie de manière modulaire (*cf.* figure 2.2). Un modèle atomique n'a pas d'accès direct aux variables d'état des autres modèles. Dans le cas contraire, il y aurait violation de modularité. Le formalisme Multi-component DEVS, basé sur le formalisme DEVS et proposé par Zeigler et al. (2000), permet de définir des systèmes couplés non modulaires, formés par un ensemble de composants. Chacun d'eux possède ses propres états, ses propres fonctions de transition et peut influencer ou être influencé par d'autres composants. Un composant peut être influencé par une entrée du système et peut contribuer à la sortie du système. Contrairement à un système couplé, les entités qui le composent ne possèdent pas d'interfaces d'entrée/sortie.

Un système multi-composant est définit par la structure :

$$multiDEVS = \langle X, Y, D, \{M_d\}, Select \rangle$$

où X et Y sont les ensembles d'entrées et de sorties et D est l'ensemble des identifiants des composants. La fonction *Select* joue le rôle d'arbitre en cas d'évènements simultanés en sélectionnant un composant :

$$Select : 2^D \rightarrow D$$

Pour chaque $d \in D$, un composant M_d est définit par :

$$M_d = \langle S_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \lambda_d, ta_d \rangle$$

où S_d est l'ensemble des états de d, $I_d \subseteq D$ est l'ensemble des composants qui *influencent* d. $E_d \subseteq D$ est l'ensemble de composants *influencés* par d. Le couple (s, e) forme un état total où e représente le temps écoulé dans l'état courant depuis la dernière transition. Q_d est l'ensemble des états totaux d'un composant d :

$$Q_d = \{(s, e_d) \mid s \in S_d, e_d \in \mathbb{R}, 0 \leq e \leq ta_d(s)\}$$

Comme un modèle DEVS, chaque composant $d \in D$ du système multi-composant planifie individuellement via sa fonction d'avancement du temps ta_d la date du prochain évènement interne qu'il recevra en fonction des états totaux q_i des influenceurs I_d :

$$ta_d : \underset{i \in I_d}{\times} Q_i \rightarrow \mathbb{R}^+_{0,\infty}$$

Lorsque le temps écoulé $e = ta(s)$, le composant calcule son futur état à travers sa fonction de transition d'état interne $\delta_{int,d}$:

$$\delta_{int,d} : \underset{i \in I_d}{\times} Q_i \rightarrow \underset{j \in E_d}{\times} Q_j$$

$\delta_{int,d}$ dépend de l'ensemble des états totaux q_i des influenceurs I_d et entraîne la modification des états totaux (s_j, e_j) des influencés E_d. Le composant peut également à ce moment là contribuer à la sortie du système via la fonction de sortie :

$$\lambda_d : \underset{i \in I_d}{\times} Q_i \rightarrow Y$$

Chaque évènement externe propagé vers le système multi-composant peut être traité par n'importe quel composant via la fonction de transition d'état externe :

$$\delta_{ext,d} : \underset{i \in I_d}{\times} Q_i \times X \rightarrow \underset{j \in E_d}{\times} Q_j$$

Cependant, la définition de δ_{ext} peut être omise si un composant n'a pas utilité de recevoir d'évènements externes. De la même manière, si un composant n'a pas besoin de propager de sorties vers d'autres modèles, la fonction λ_d peut être omise. Le lecteur intéressé par la définition des simulateurs associés à un système multi-composant et son équivalent DEVS peut se référer à Zeigler et al. (2000).

L'utilisation d'un système multi-composant facilite donc une approche de conception ascendante. Il est par exemple beaucoup plus adapté qu'un système couplé pour la définition d'entités spatialement explicites telles que des automates cellulaires ou des environnements cellulaires pour les SMA. Zeigler et al. (2000) donnent d'ailleurs un exemple de définition du jeu de la vie spécifié avec Multi-component DEVS. Cependant, cette approche est définie uniquement pour le formalisme DEVS classique, dont la sémantique n'est pas compatible celle de PDEVS. Pour pallier à cela, nous présentons dans l'annexe B de présenter la définition du formalisme multiPDEVS, qui vise à intégrer les solutions apportées par PDEVS pour l'approche multi-composant.

Annexe B

FORMALISME MULTIPDEVS

Cette annexe propose la définition du formalisme multiPDEVS, une variante du formalisme multi-composant (Zeigler et al., 2000) compatible avec la sémantique du formalisme PDEVS. Nous donnons sa définition, et une preuve d'équivalence avec le formalisme PDEVS. Le travail présenté ici a été motivé par notre objectif n° 2 cherchant à proposer un outil opérationnel pour la représentation d'environnements cellulaires SMA. L'approche multi-composant semble être une réponse possible. Ce travail fait actuellement l'objet d'une soumission à un comité de lecture pour publication sous le titre : « *multiPDEVS : A Parallel Multicomponent System Specification Formalism* », co-signé par : Damien Foures, Romain Franceschini, Paul-Antoine Bisgambiglia et Bernard P. Zeigler.

1 *multiPDEVS*

La structure d'un multicomposant DEVS parallèle est :

$$multiPDEVS = (X, Y, D, \{M_d\})$$

où X, Y, et D sont définis comme dans l'approche multi-composant classique. De manière équivalente à PDEVS, la fonction *Select* a disparu de la définition de multiPDEVS. Elle devient inutile puisque l'ensemble des composants imminents sont gérés simultanément, au lieu d'être sérialisés.

Pour chaque $d \in D$, un composant M_d est défini par la structure :

$$M_d = (S_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \delta_{con,d}, \delta_{reac,d}, \lambda_d, ta_d)$$

où S_d est l'ensemble des états séquentiels de d, $Q_d = \{(s, e_d) \mid s \in S_d, e_d \in \mathbb{R}_0^+\}$ est l'ensemble des états totaux avec e le temps passé dans l'état depuis la dernière transition, $I_d \subseteq D$ est l'ensemble des composants influencent et $E_d \subseteq D$ est l'ensemble des composants influencés. Les composants établissent la date de leur prochain événement interne en fonction des états totaux de leurs influenceurs en utilisant la fonction d'avancement du temps :

$$ta_d : \underset{i \in I_d}{\times} Q_i \rightarrow \mathbb{R}_0^+ \cup \{\infty\}.$$

Lorsque $e_d = ta_d(s)$, le composant peut générer un ensemble de sorties pour l'interface de sortie du multiPDEVS via la fonction de sortie (qui peut rester non définie) :

$$\lambda_d : \underset{i \in I_d}{\times} Q_i \rightarrow Y^b.$$

Ensuite, l'événement interne arrive, sa fonction de transition interne est activée. Il prend l'ensemble des états totaux des influenceurs et propose un ensemble d'états suggérés pour l'ensemble des influencés :

$$\delta_{int,d} : \underset{i \in I_d}{\times} Q_i \rightarrow \underset{j \in E_d}{\times} S_j$$

Si un ou plusieurs nouveaux états sont produits par les transitions d'état de composants pouvant influencer d, alors la fonction de transition de d est activée afin de laisser le modélisateur définir explicitement le comportement de collision d'état :

$$\delta_{reac,d} : K_d^b \times Q_d \rightarrow S_d$$

où

$$K_d = \{(s_d, c) \mid s_d \in S_d\}.$$

Le couple (s_d, c) est l'état suggéré par le composant c pour le composant d.

Cette fonction récupère un bag d'états proposés par les composants comprenant d comme influencé. Ce bag d'état est associé à l'état total actuel du composant d afin de produire un nouvel état total valide. Nous proposons d'utiliser Q_d à la place d'un produit vectoriel de tous les états proposés par les influenceurs ($\underset{i \in I_d}{\times} Q_i$) afin d'assurer la cohérence d'états. Si nous décidions d'autoriser une telle démarche, nous devrions nous assurer que l'ensemble des lectures dans $I_d - \{d\}$ sont effectuées dans l'état courant et non pas sur les nouveaux états produits par leur propre fonction de réaction δ_{reac}. Ce cas de figure peut apparaître si les états proposés ont été produit au même temps de simulation pour des composants incluent dans $I_d - \{d\}$.

Lorsque des événements externes se produisent au niveau multiPDEVS, les composants associés au multiPDEVS peuvent les recevoir s'ils définissent leur fonction de transition externe correspondante :

$$\delta_{ext,d} : \underset{i \in I_d}{\times} Q_i \times X^b \rightarrow \underset{j \in E_d}{\times} S_j.$$

Comme dans PDEVS (contrairement à DEVS et multiDEVS), $\delta_{ext,d}$ reçoit un bag d'entrées qui contient tous les événements externes simultanés au lieu d'un seul événement d'entrée.

Si un événement interne arrive au même moment qu'un ou plusieurs événements externes, une collision doit être gérée. La fonction de confluence, initialement introduite dans PDEVS, permet de gérer explicitement le comportement de collision :

$$\delta_{con,d} : \underset{i \in I_d}{\times} Q_i \times X^b \rightarrow \underset{j \in E_d}{\times} S_j.$$

Notons que de manière similaire à la fonction de transition interne, $\delta_{ext,d}$ et $\delta_{con,d}$ génèrent un ensemble d'états proposés (S) pour les influencés sans mettre à jour directement leurs états.

Un multiPDEVS se comporte de la façon suivante : Comme dans PDEVS, l'activation de composants imminents se fait simultanément en utilisant une approche en deux phases. Dans une première phase, les sorties de tous les composants qui définissent une fonction λ sont collectées. Nous divisons la deuxième phase en deux micro-étapes : Dans la première micro-étape, les transitions d'état appropriées sont effectuées et leurs sorties (les bags d'états) sont temporairement enregistrées pour la deuxième micro-étape. Pour les composants qui ont défini la fonction δ_{ext}, les transitions externes sont activées lorsque des événements externes se produisent sur l'interface d'entrée du multiPDEVS. Pour tous les composants imminents, lorsqu'il existe également des entrées disponibles au niveau multiPDEVS, les fonctions de confluences sont activées pour les composants qui ont défini δ_{ext}, sinon leur δ_{int} est activé. S'il n'y a pas d'entrées, les transitions internes sont activées pour tous les composants imminents. La deuxième micro-étape consiste à activer la fonction de réaction pour tous les composants qui ont un bag d'états généré au cours de la première micro-étape.

2 *Construction d'équivalence avec PDEVS*

Nous fournissons ici une construction d'équivalence entre multiPDEVS et PDEVS en donnant la résultante de multiPDEVS par un modèle atomique PDEVS. D'après la définition d'un multiPDEVS donnée, nous associons le modèle atomique PDEVS suivant :

$$PDEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

où $S = \underset{d \in D}{\times} Q_d$.

La fonction d'avancement du temps est définie par

$$ta(s) = min\{\sigma_d \mid d \in D\}$$

avec $\sigma_d = ta_d(\dots, (s_i - tl_i), \dots) - e_d$ le temps restant jusqu'à la date du prochain évènement planifié par d.

L'ensemble des composants imminents est défini par :

$$IMM(s) = \{d_\sharp \mid d_\sharp \in D, \sigma_{d_\sharp} = ta(s)\}.$$

Pour chaque $d \in D$, nous définissons $S_d = \{s_d \mid d \in E_{d_\sharp}\}$ l'ensemble d'évènements suggérés produits par $d_\sharp$ et $K_d = \{(s_d, d_\sharp) \mid d_\sharp \in IMM(s), d \in E_{d_\sharp}\}$, l'ensemble des états suggérés et des composants qui les ont générés pour chaque composant influencé d.

Etant donné $d_\sharp \in IMM(s)$, la fonction de transition interne du multiPDEVS est définie par :

$$\delta_{int}(q_1, q_2, \ldots, q_n) = (q'_1, q'_2, \ldots, q'_n)$$

avec

$$(s'_j, e'_j) = \begin{cases} (s_j, e_j + ta(s)) & \text{si } k_j^b = \varnothing \\ (\delta_{reac,j}(k_j^b, (s_j, e_j + ta(s))), 0) & \text{sinon} \end{cases}$$

où

$$k_j^b = \{(\delta_{int,d_\sharp}(\ldots, (s_i, e_i + ta(s)), \ldots) \bullet j, d_\sharp) \mid d_\sharp \in IMM(s), j \in E_{d_\sharp}, i \in I_j\}$$

La fonction de transition interne $\delta_{int,d_\sharp}$ de chaque composant imminent $d_\sharp$ est exécutée, où $(\ldots, (s_i, e_i + e), \ldots)$ est le vecteur d'états des composants influenceurs $i \in I_{d_\sharp}$, qui produit un ensemble d'états suggérés pour chaque composant influencés $j \in E_{d_\sharp}$. Afin de remplir le sac entrant d'états suggérés k_j^b pour chaque composant influencé $j \in E_{d_\sharp}$, chaque état suggéré produit pour j est transformé en un tuple composé de l'état et de l'identifiant du composant qui l'a généré. Pour tous les composants influencés par les composants imminents, la fonction de transition de réaction $\delta_{reac,j}(k_j^b, (s_j, e_j))$ est activée. Pour les composants dont le sac d'états suggérés est vide ($k_j^b = \varnothing$), le temps écoulé est mis à jour.

La sortie du système est définie de la manière suivante :

$$\lambda(s) = \{\lambda_{d_\sharp}(\ldots, (s_i, t - tl_i), \ldots) \mid d_\sharp \in IMM(s), i \in I_{d_\sharp}\}$$

À l'arrivée d'un sac d'évènements externes x^b, la fonction de transition externe δ_{ext} est définie par le produit croisé des fonctions de transition externes de tous les composants $d \in D$. La fonction globale de transition externe est définie par :

$$\delta_{ext}((q_1, q_2, \ldots, q_n), e, x^b) = (q'_1, q'_2, \ldots, q'_n)$$

avec

$$(s'_j, e'_j) = \begin{cases} (s_j, e_j + e) & \text{si } k_j^b = \varnothing \\ (\delta_{reac,j}(k_j^b, (s_j, e_j + e)), 0) & \text{sinon} \end{cases}$$

où

$$k_j^b = \{(\delta_{ext,d}((\ldots, (s_i, e_i + e), \ldots), x^b) \bullet j, d) \mid d \in D, i \in I_d, j \in E_d, \exists \delta_{ext,d}\}$$

Avec un mécanisme similaire à la fonction de transition interne globale, le sac d'entrée

d'états suggérés (k_j^b) de chaque composant influencé est construit d'après le résultat des fonctions $\delta_{ext,d}$. Comme précédemment, les autres composants dont le sac d'états suggérés est vide ($k_j^b = \varnothing$), le temps écoulé est mis à jour.

Enfin, la fonction de transition de confluence δ_{con} est définie par :

$$\delta_{con}((q_1, q_2, \ldots, q_n), x^b) = (q'_1, q'_2, \ldots, q'_n)$$

avec

$$(s'_j, e'_j) = \begin{cases} (s_j, e_j + ta(s)) & \text{si } k_j^b = \varnothing \\ (\delta_{reac,j}(k_j^b, (s_j, e_j + ta(s))), 0) & \text{sinon} \end{cases}$$

où

$$k_j^b = \begin{cases} \{(\delta_{int,d}(\ldots, (s_i, e_i + ta(s)), \ldots) \bullet j, d) \\ \qquad \mid d \in IMM(s), i \in I_d, j \in E_d\} & \text{si } \nexists \delta_{ext,d}, d \in IMM(s) \\ \{(\delta_{con,d}((\ldots, (s_i, e_i + ta(s)), \ldots), x^b) \bullet j, d) \\ \qquad \mid d \in IMM(s), i \in I_d, j \in E_d\} & \text{si } \exists \delta_{ext,d}, d \in IMM(s) \\ \{(\delta_{ext,d}((\ldots, (s_i, e_i + ta(s)), \ldots), x^b) \bullet j, d) \\ \qquad \mid d \notin IMM(s), i \in I_d, j \in E_d\} & \text{if } \exists \delta_{ext,d}, d \notin IMM(s) \end{cases}$$

Pour la fonction globale de confluence, le sac d'états suggérés est construit à partir de trois contributions : à partir des composants imminents dont la fonction $\delta_{ext,d}$ n'est pas définie (la fonction $\delta_{int,d}$ est activée), à partie des composants imminents dont la fonction $\delta_{ext,d}$ est définie (la fonction $\delta_{con,d}$ est activée) et enfin, à partir des composants non-imminents dont la fonction $\delta_{ext,d}$ est définie (la fonction $\delta_{ext,d}$) est activée).

Annexe C

EVALUATION DE PERFORMANCES

1 *Comparaison des performances des simulateurs Quartz et aDEVS*

Cette section a pour objectif de comparer les performances de notre cadriciel Quartz avec les performances d'adevs (Nutaro, 1999), ce dernier étant considéré par la littérature comme l'un des simulateurs PDEVS les plus performants (Franceschini et al., 2014a ; Van Tendeloo et Vangheluwe, 2016). Pour ce comparatif nous avons utilisé le benchmark DEVStone (Wainer et al., 2011), qui a été spécifiquement conçu pour permettre de comparer les performances de simulateurs PDEVS. DEVStone permet de générer une hiérarchie de modèles de façon automatique avec une structure et un comportement variable.

L'environnement utilisé pour exécuter ce comparatif de performances repose sur un processeur Intel(R) Core(TM) i5-3360M @ 2.80Ghz (3MB L2), 16GB (2 x DDR3 - 1600 Mhz) de RAM, un disque dur Toshiba MK5061GS, et sur le système d'exploitation Ubuntu 16.04. Le logiciel utilisé repose sur le cadriciel Quartz (commit a663dfe), compilé avec Crystal dans sa version 0.24.0 (*release mode*), ainsi que sur le simulateur adevs, dans sa version 2.8.1, compilé avec gcc 4.8.2 en mode -O3.

Pour les deux plateformes, nous avons mesuré le temps horloge moyen d'exécution, basé sur 10 répétitions du test. Les paramètres utilisés pour générer la hiérarchie de modèles DEVStone correspond a une profondeur fixe de 3 (nombre de modèles couplés imbriqués), le type de couplage prédéfini HI et un temps de calcul des fonctions δ parametré à 0 secondes. Le paramètre que nous faisons varier est celui de la *largeur*, et correspond au nombre de composants dans chaque niveau de hiérarchie.

La figure 3.1 montre les résultats du comparatif dans des conditions idéales pour l'échéancier des simulateurs. La fonction d'avancement du temps des modèles DEVStone renvoie une valeur aléatoire selon une distribution uniforme. Dans ces conditions, les deux simulateurs sont sur le même ordre de performances.

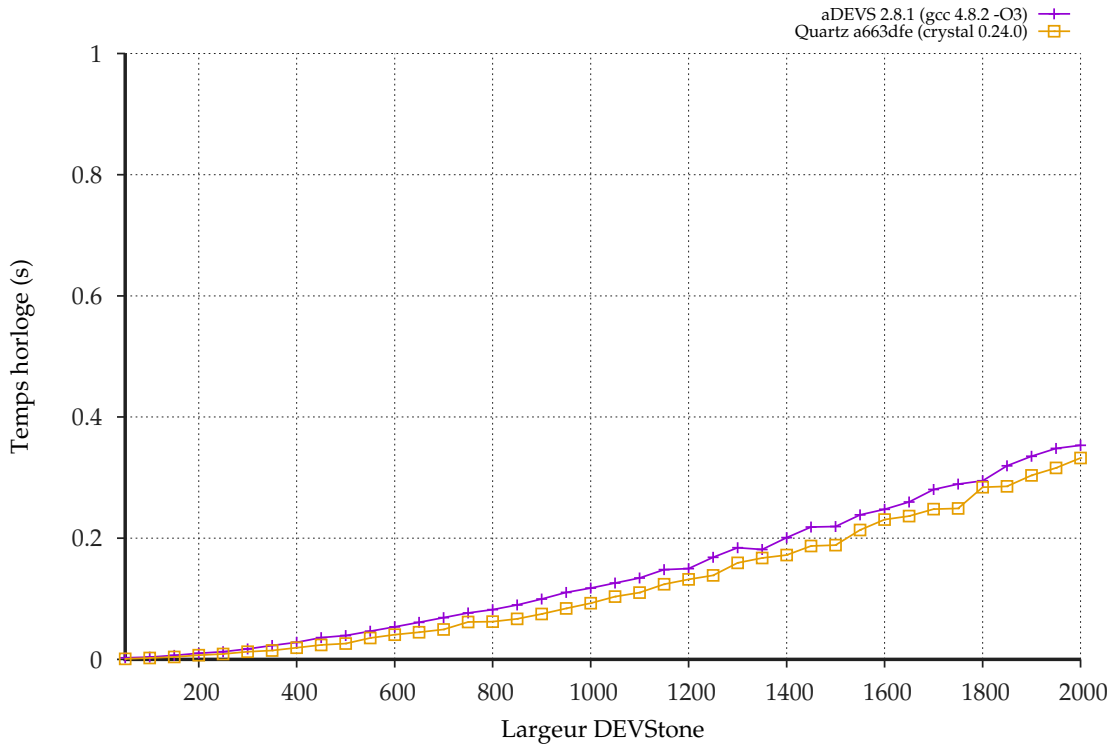


Figure 3.1 Évolution du temps horloge d'exécution en secondes du benchmark DEVStone en fonction de la largeur, pour une distribution d'évènements uniforme.

La figure 3.2 donne les résultats d'un second test réalisé dans des conditions plus stressantes pour l'échéancier. Cette fois, la fonction d'avancement du temps des modèles DEVStone est fixée à 1, ce qui a pour conséquence de générer un nombre important d'évènements simultanés. Ici, l'échéancier utilisé par défaut dans Quartz (basé sur la file de priorité *calendar queue*), semble mieux gérer des situations de collisions par rapport à l'échéancier d'adevs. Dans ces conditions, notre outil obtient sensiblement de meilleures performances.

2 *Comparaison des performances de différents échéanciers*

Nous donnons dans cette section les résultats d'un comparatif de performances de différents types d'échéanciers implémentés dans notre cadriciel Quartz (*cf* section IV.4.3). Un travail similaire pour l'implémentation de DEVS-Ruby a fait l'objet d'une publication dans Franceschini et al. (2015). Nous détaillons dans un premier temps la méthode que nous avons utilisée pour réaliser ce comparatif avant de présenter les résultats.

Afin de capturer au mieux les aspects pratiques d'une simulation à évènements discrets, nous avons utilisé une variante du *hold model* (Vaucher et Duval, 1975) proposée par Himmelspach et Uhrmacher (2007b), qui permet de simuler les différentes étapes réalisées lors d'une simulation PDEVS.

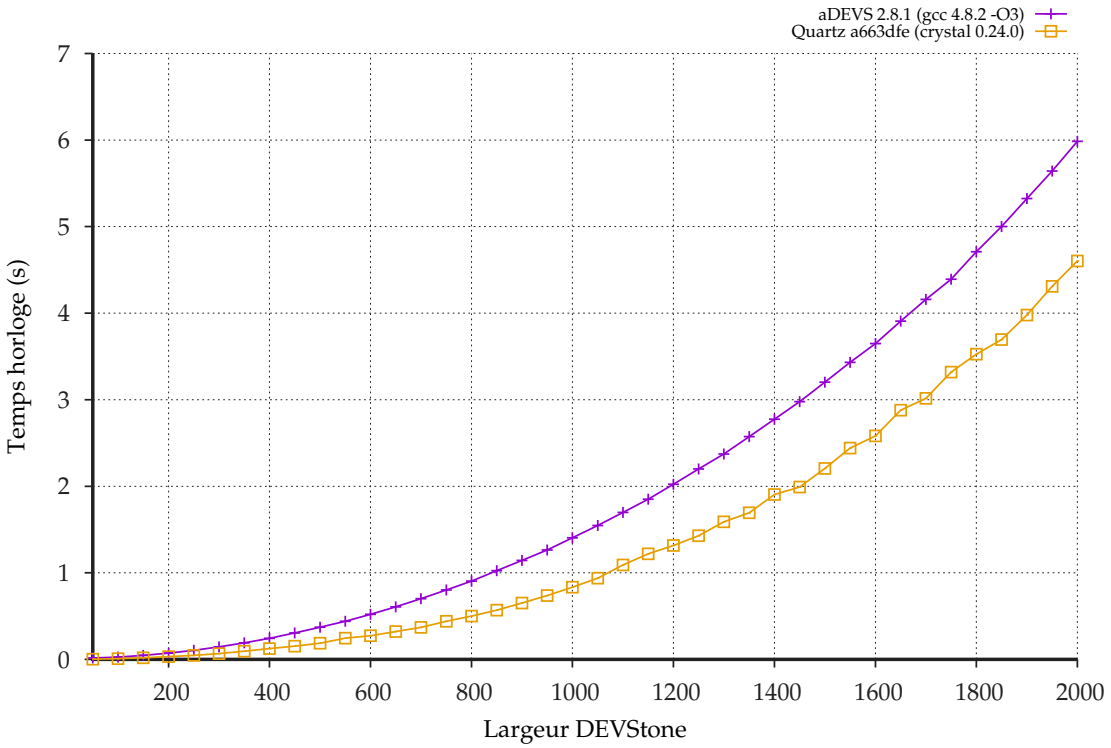


Figure 3.2 Évolution du temps horloge d'exécution en secondes du benchmark DEVStone en fonction de la largeur, pour une distribution d'évènements uniforme.

La distribution des évènements peut plus ou moins affecter les performances de l'échéancier (Franceschini et al., 2015). Le tableau 3.1 donne les différentes expressions que nous avons utilisées pour générer la date d'activation associée à chaque évènement, ainsi que pour générer les prochaines dates d'activation lors de chaque étape. Les différences entre deux distributions peuvent être plus ou moins bien gérées par l'échéancier. En effet, une asymétrie ou un pic dans la distribution des évènements augmente le nombre d'évènements simultanés en file d'attente, ce qui représente une condition non-optimale pour un échéancier.

Distribution	**Expression**
Constant	1
Uniform (0, 1)	$rand$
Uniform (0.9, 1.1)	$0.9 + 0.2 * rand$
Exponential	$-ln(rand)$
Triangular (0, 1.5)	$1.5 * sqrt(rand)$
Neg Triangular (0, 1000)	$1000 * (1 - sqrt(1 - rand)$
Bimodal	$0.95238 * rand + rand < 0.1 \, ? \, 9.5238 : 0$
Camel (2, 0.8, 0.2)	cf. (Rönngren et al., 1991)

Tableau 3.1 Expressions utilisées pour la distribution d'évènements.

L'environnement utilisé pour exécuter ce comparatif de performances repose sur un processeur Intel(R) Core(TM) i5-3210M @ 2.50Ghz, 8GB (2 x DDR3 - 1600 Mhz) de RAM, un

disque dur APPLE SSD SM128E, et sur le système d'exploitation macOS 10.13.1. Le logiciel utilisé repose sur le cadriciel Quartz (commit a663dfe), compilé avec Crystal dans sa version 0.23.1 (*release mode*).

Nous avons exécuté le test suggéré par (Himmelspach et Uhrmacher, 2007b) pour quatre échéancier, basés sur les files de priorités suivantes, décrites dans la section IV.4.3 : la *calendar queue*, la *ladder queue*, le tas binaire et la liste triée. Nous avons soumis chaque échéancier aux différentes distributions listées dans le tableau 3.1, et mesuré le temps horloge d'exécution moyen, basé sur 10 répétitions.

La figure 3.3 montre montre les résultats d'un premier test réalisé avec un faible nombre d'évènements (10000) et sur 100000 cycles de simulation. L'échelle logarithmique en ordonnée représente le temps d'exécution, l'abscisse permet de visualiser les performances des différents échéanciers pour chaque type de distribution. L'échéancier basé sur la *sorted list*, suggéré dans les simulateurs abstraits (Zeigler et al., 2000), offre les performances les moins intéressantes. L'échéancier basé sur le tas binaire offre à l'inverse des performances très intéressantes. Ceci est dû en partie par une optimisation de la fonction de suppression d'un élément de la liste. Si en théorie, celle-ci entraîne une recherche linéaire dans le pire des cas, nous utilisons une structure annexe permettant de réaliser cette recherche en temps constant amorti. Parmi les structures multi-listes, la *calendar queue* est ici la plus performante, bien qu'elle semble plus sensible aux pics de distributions que la *ladder queue*.

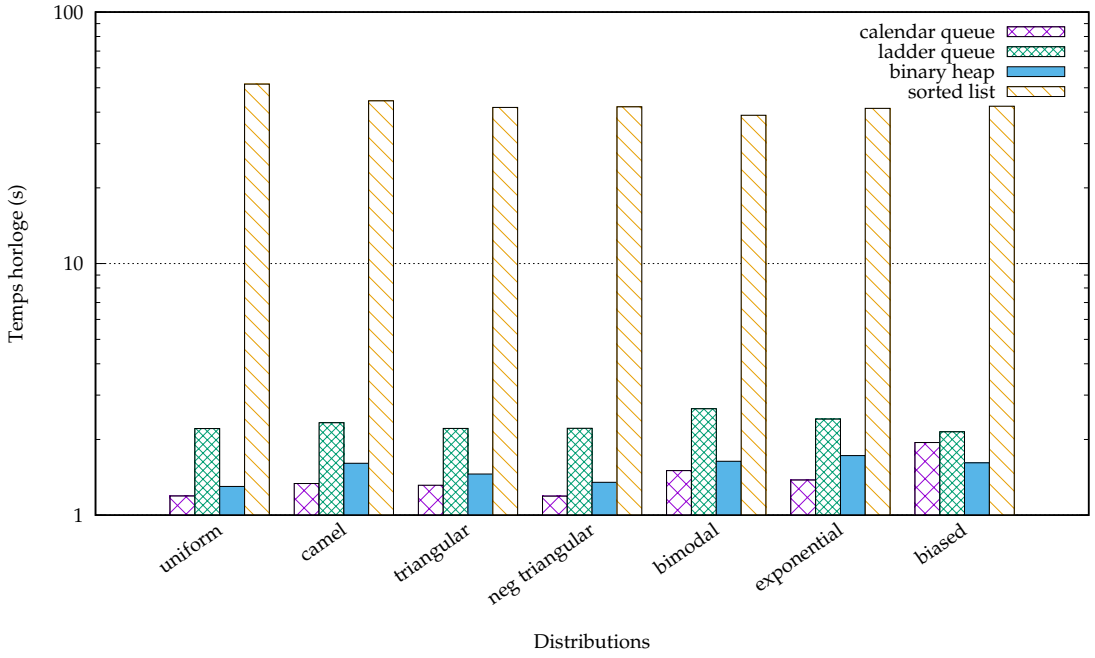


Figure 3.3 Performances d'exécution des échéanciers selon différentes distributions d'évènements et sur 10000 évènements, avec le benchmark PDEVS.

La figure 3.4 donne les résultats d'un second test réalisé avec un nombre plus élevé d'évènements (500000), toujours sur 100000 cycles de simulation. Dans ce cas, la *ladder queue* est la plus performante dans la majorité des distributions, tout en préservant sa capacité à

absorber les pics de distributions.

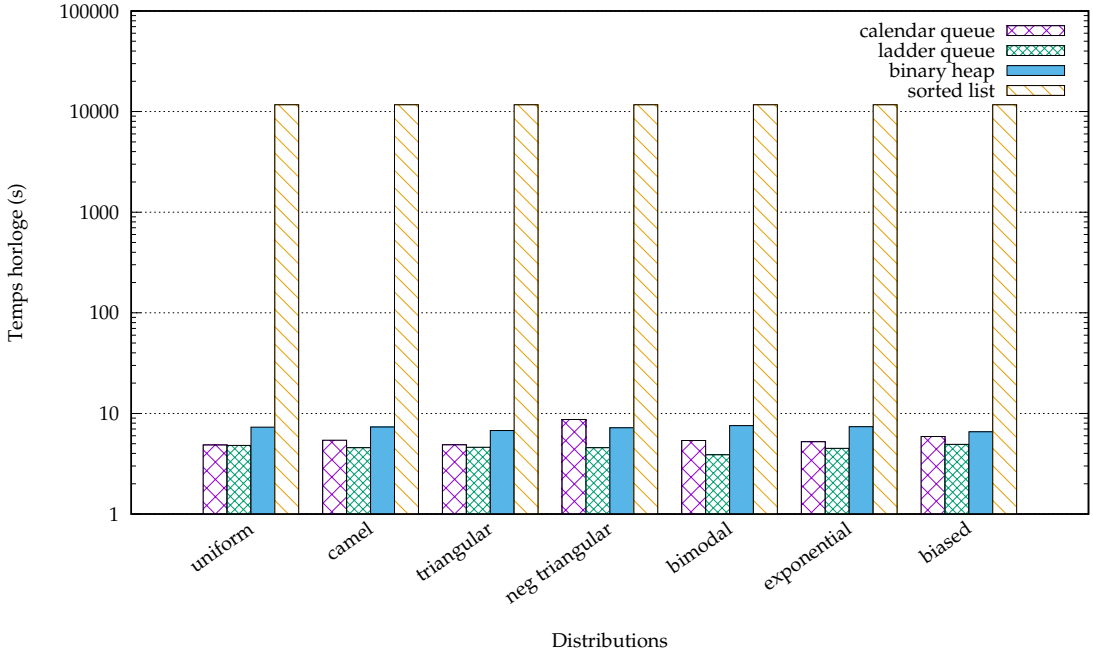


Figure 3.4 Performances d'exécution des échéanciers selon différentes distributions d'évènements et sur 500000 évènements, avec le benchmark PDEVS.

Annexe D

IMPLÉMENTATION DE SUGARSCAPE

Dans cette annexe, une traduction du modèle SugarScape tel que nous l'avons décrit dans le chapitre V (section 3) est donnée en langage Crystal. Plus particulièrement, le code ci-dessous est la traduction de la seconde version du modèle. L'implémentation utilise l'API du cadriciel Quartz.

Construction du modèle

```
require "../src/mas"
require "./sugarscape/env"
require "./sugarscape/influences"
require "./sugarscape/agent"
require "./sugarscape/plot"
require "./sugarscape/sfml_renderer"

WIDTH  = 50
HEIGHT = 50

class SugarScape < Quartz::DSDE::CoupledModel
  def initialize(name, pop, depth = 7)
    bounds = Quartz::MAS::Rect.new(0, 0, WIDTH, HEIGHT)
    env = Env.new("env", bounds, depth)
    env.add_field(:sugar)
    super(name, env)

    initial_sugar = Array(Array(Int8)).new
    env.initial_state = Env::State.new(
      population: pop,
      sugar_max: initial_sugar
    )

    file = File.new("examples/sugarscape/sugar-map.txt", "r")
    y = 0
```

```
26      file.each_line do |line|
27        x = 0
28        initial_sugar << [] of Int8
29        row = line.split(/[ ]+/).map(&.to_i).map do |value|
30          target = Quartz::MAS::Point2f.new(x.to_f, y.to_f)
31          env.field(:sugar)[target] = value.to_f
32          initial_sugar.last << value.to_i8
33          x += 1
34        end
35        y += 1
36      end
37      file.close
38
39      occupied_cells = Set(Quartz::MAS::Point2f).new
40      pop.times do |i|
41        pos = bounds.sample_point.floor
42        while occupied_cells.includes?(pos)
43          pos = bounds.sample_point.floor
44        end
45        occupied_cells << pos
46
47        vision = rand(1..6)
48        scope = Quartz::MAS::CrossScope.new(vision, 0.99)
49        name = "agent_#{MyAgent.number}"
50        body = MyBody.new(name, pos, scope)
51        agent = MyAgent.new(name)
52        MyAgent.number += 1
53
54        env.add_output_port(name)
55        agent.add_input_port(env.name)
56        agent.add_output_port(env.name)
57        self << agent
58
59        env.add(body.as(Quartz::MAS::Entity))
60
61        attach name, to: env.name, between: env, and: agent
62        attach env.name, to: :influences, between: agent, and: env
63      end
64    end
65  end
66
67  INIT_POP = 400
68  model = SugarScape.new(:sugarscape, INIT_POP)
69  sim = Quartz::Simulation.new(model, duration: 301.0)
70
```

```
pop = PopulationPlotter.new
wealth = WealthPlotter.new
lg = LorenzGiniPlotter.new
model["env"].add_observer(pop)
model["env"].add_observer(wealth)
model["env"].add_observer(lg)

renderer = SFMLRenderer(SugarScape, Env).new(sim, model)
renderer.work
```

Définition du corps et du système cognitif de l'agent

```
class MyBody < Quartz::MAS::Body
  # No specific observables properties
end

class MyAgent < Quartz::AtomicModel
  class_property number = 0

  @sigma = Quartz::INFINITY

  # the metabolic rate, which represents the units of sugar the agent burns per
      time-step.
  state_var metabolism : Int32

  # Agents also have the capacity to accumulate sugar wealth.
  # An agent's sugar wealth is incremented at the end of each
  # time-step by the sugar collected and decremented by the
  # agent's metabolic rate.
  state_var sugar : Int32

  state_var max_age : Int32
  state_var age : Int32

  # The next action
  state_var next_influences : Array(Quartz::MAS::Influence)

  state_initialize do
    @metabolism = rand(1..4)
    @sugar = rand(5..25)
    @max_age = rand(50..100)
    @age = 0
    @next_influences = Array(Quartz::MAS::Influence).new
  end
```

```
32
33  private def has_move?(sensor) : Bool
34    !sensor.my_influences(Quartz::MAS::Move.name).empty?
35  end
36
37  private def move_rejected?(sensor) : Bool
38    moves = sensor.my_influences(Quartz::MAS::Move.name)
39    !moves.empty? && moves.first.rejected?
40  end
41
42  private def gathered_sugar(sensor) : Int8?
43    sugars = sensor.my_influences(GatherSugar.name)
44    if !sugars.empty?
45      return sugars.first.as(GatherSugar).gathered
46    end
47    nil
48  end
49
50  private def choose_patch(sensor) : Quartz::MAS::Point2f?
51    # consider moving to unoccupied patches in our vision, as well as staying
        at the current patch
52    move_candidates = sensor.field_values_nearby(:sugar).select do |p, v|
53      # check if this cell is occupied
54      sensor.entities_at(p).empty?
55    end
56
57    if !move_candidates.empty?
58      # select patches with the most sugar
59      sugarmax = move_candidates.max_by(&.[1])[1]
60      sweetests = move_candidates.select &.[1].==(sugarmax)
61
62      # if sugar levels around are worth a move
63      if sugarmax > sensor.field_value(:sugar)
64        # move to one of the patches that is closest
65        my_pos = sensor.myself.position.as(Quartz::MAS::Point2f)
66        mindist = sweetests.min_of &.[0].distance(my_pos)
67        closests = sweetests.select(&.[0].distance(my_pos).==(mindist))
68        return closests.sample[0] unless closests.empty?
69      end
70    end
71
72    nil
73  end
74
75  private def sweetests_patches(sensor) : Array(Quartz::MAS::Point2f)?
```

```
76      # consider moving to unoccupied patches in our vision, as well as staying
            at the current patch
77      move_candidates = sensor.field_values_nearby(:sugar).select do |p, v|
78        sensor.entities_at(p).empty?
79      end
80
81      if !move_candidates.empty?
82        # select patches with the most sugar
83        sugarmax = move_candidates.max_by(&.[1])[1]
84        sweetests = move_candidates.select &.[1].==(sugarmax)
85        my_pos = sensor.myself.position.as(Quartz::MAS::Point2f)
86
87        return sweetests.map(&.[0]).shuffle!.sort_by(&.distance(my_pos))
88      end
89
90      [sensor.myself.position.as(Quartz::MAS::Point2f)]
91    end
92
93    @rounds : UInt8 = 0u8
94    EPSILON    = 0.125
95    MAX_ROUNDS =   1u8
96
97    def external_transition(bag)
98      sensor = bag[input_port("env")].first.raw.as(Quartz::MAS::Sensor)
99      @sigma = 1
100
101     # agent already chose a cell, but may want to change his mind.
102     # actually, this is mandatory because a cell that was free during the agent
103     # first choice, may now be occupied by another agent during a conflict.
104     if !@next_influences.empty?
105       influence = @next_influences.first
106       if influence.is_a?(Quartz::MAS::Move)
107         if target = choose_patch(sensor)
108           if target != influence.target
109             @next_influences[0] = Quartz::MAS::Move.new(sensor.myself, target)
110           end
111         end
112       end
113       @sigma = @sigma - @elapsed
114       return
115     end
116
117     # A conflict occurred
118     if move_rejected?(sensor) && @rounds < MAX_ROUNDS
119       if target = choose_patch(sensor)
```

```
120         @next_influences << Quartz::MAS::Move.new(sensor.myself, target) <<
                GatherSugar.new(sensor.myself)
121         @sigma = EPSILON
122         @rounds += 1u8
123         return
124       else
125         @next_influences << GatherSugar.new(sensor.myself)
126         @sigma = EPSILON
127         @rounds += 1u8
128         return
129       end
130     elsif @rounds > 0u8
131       @sigma = 1 - @rounds * EPSILON
132       @rounds = 0u8
133     else
134       @sigma = 1
135     end
136
137     if amount = gathered_sugar(sensor)
138       @sugar = (@sugar - @metabolism + amount)
139     end
140
141     @age += 1
142
143     if @sugar <= 0 || @age > @max_age
144       @next_influences << Death.new(sensor.myself, @sugar <= 0)
145     else
146       if target = choose_patch(sensor)
147         # move to the new patch and gather the sugar
148         @next_influences << Quartz::MAS::Move.new(sensor.myself, target)
149       end
150
151       @next_influences << GatherSugar.new(sensor.myself)
152     end
153   end
154
155   def internal_transition
156     @next_influences.clear
157     @sigma = Quartz::INFINITY
158   end
159
160   def output
161     post @next_influences.map &.as(Quartz::Type), on: "env"
162   end
163 end
```

Définition des influences

```
class ConstantGrowback < Quartz::MAS::Influence
  def commit(env)
    field = env.field(:sugar)
    # gradually grow back all of the sugar for each patch
    env.sugar_max.each_with_index do |row, y|
      row.each_with_index do |max, x|
        next if max == 0i8
        point = Quartz::MAS::Point2f.new(x.to_f, y.to_f)
        current_value = field[point]
        field[point] = Math.min(max.to_f, current_value + 1)
      end
    end
  end

  def target
    @influencer.position
  end
end

class GatherSugar < Quartz::MAS::Influence
  getter gathered : Int8 = 0i8

  def commit(env)
    point = @influencer.position.as(Quartz::MAS::Point2f)
    field = env.field(:sugar)
    @gathered = field[point].to_i8
    field[point] = 0.0
  end

  def target
    @influencer.position
  end
end

class Move < Quartz::MAS::Influence
  MAX_TRIES = 10
  getter index : Int32 = 0
  @targets : Array(Quartz::MAS::Point2f)

  def initialize(influencer, @targets : Array(Quartz::MAS::Point2f))
    super(influencer)
```

```
42    end
43
44    def target
45      @targets[@index]
46    end
47
48    def invalidate_target!
49      if @targets.size == @index + 1 || @index == MAX_TRIES
50        reject!
51      else
52        @index += 1
53      end
54    end
55
56    protected def commit(env : Quartz::MAS::Topology)
57      env.update_position(@influencer.as(Quartz::MAS::Entity), self.target)
58    end
59  end
60
61  class Death < Quartz::MAS::Influence
62    include Quartz::MAS::Death
63
64    getter starving : Bool = false
65
66    def initialize(influencer, @starving)
67      super(influencer)
68    end
69
70    protected def commit(env)
71      env.remove_agent(@influencer)
72      env.as(Env).starvation += 1 if @starving
73    end
74
75    def target
76      @influencer.position
77    end
78  end
```

Définition du calcul de réaction de l'environnement

```
1   class Env < Quartz::MAS::ContinuousEnvExecutive
2     @sigma = 0
3     @phase = Phase::Perception
4     REPLACEMENT = true
```

```
5
6   state_var sugar_max : Array(Array(Int8)) = Array(Array(Int8)).new
7   state_var population : Int32 = 0
8   state_var starvation : Int32 = 0
9   setter starvation
10
11  def add(e)
12    super(e)
13    if e.is_a?(Quartz::MAS::Body)
14      @population += 1
15    end
16  end
17
18  def delete(e)
19    super(e)
20    if e.is_a?(Quartz::MAS::Body)
21      @population -= 1
22    end
23  end
24
25  def output
26    if @sigma == 0 && @phase.perception?
27      puts "initial perception"
28
29      @spatial_index.each do |_, entity|
30        if entity.is_a?(Quartz::MAS::Body)
31          post(sensor_for(entity), entity.agent)
32        end
33      end
34    else
35      super
36    end
37  end
38
39  def natural
40    {ConstantGrowback.new(@body)}
41  end
42
43  def reaction(influences)
44    if 0 < @elapsed < 1
45      @sigma = (@sigma - @elapsed)
46    else
47      @sigma = 1
48    end
49
```

```
50    noeaters = Set(Quartz::Name).new
51    # Two agents are not allowed to occupy the same cell in the grid.
52    moves = influences[Quartz::MAS::Move.name].group_by &.target.as(Quartz::MAS
          ::Point2f)
53    points = moves.keys.shuffle!
54    points.each do |target|
55      candidates = moves[target]
56
57      # if the cell is occupied
58      if (content = @spatial_index[target]?) && !content.empty?
59        # this is no good, so reject all candidates
60        candidates.each &.reject!
61        next
62      end
63
64      # if several agents lust for the same patch, randomly select one
65      if candidates.size > 1
66        elected = candidates.sample
67        candidates.each { |influence|
68          if influence == elected
69            influence.execute(self)
70          else
71            noeaters << influence.influencer.agent
72            influence.reject!
73          end
74        }
75      else
76        candidates.first.execute(self)
77      end
78    end
79
80    influences[GatherSugar.name].each do |influence|
81      if noeaters.includes? influence.influencer.agent
82        influence.reject!
83      else
84        influence.execute(self)
85      end
86    end
87
88    influences[Death.name].each do |influence|
89      influence.execute(self)
90
91      # Agent replacement rule R:
92      # Whenever an agent dies it is replaced by a new agent of age 0 placed on
            a randomly chosen unoccupied cell, having random attributes v, m and
```

```
          max-age, and random initial wealth w0.
      if REPLACEMENT
        vision = rand(1..6)

        pos = bounds.sample_point.floor
        while (content = @spatial_index[pos]?) && !content.empty?
          pos = bounds.sample_point.floor
        end

        scope = Quartz::MAS::CrossScope.new(vision, 0.99)
        name = "agent_#{MyAgent.number += 1}"
        body = MyBody.new(name, pos, scope)
        agent = MyAgent.new(name)
        MyAgent.number += 1
        self.add_agent(body, agent)
      end
    end

    influences[ConstantGrowback.name].each do |influence|
      influence.execute(self)
    end
  end

  def internal_transition
    if @sigma == 0 && @phase.perception?
      @sigma = 1
    end

    super

    if @population == 0
      puts "population = 0. aborting"
      @sigma = Quartz::INFINITY
    end
  end
end
```

Interface graphique et observateurs

Interface graphique

```
require "crsfml"

class SFMLRenderer(M, E)
```

```
4   include Quartz::Hooks::Notifiable
5   include Quartz::Observer
6
7   getter sim : Quartz::Simulation
8   getter model : M
9
10  @window : SF::RenderWindow?
11  @render_states : SF::RenderStates?
12
13  def initialize(@sim, @model)
14    Quartz::Hooks.notifier.subscribe(Quartz::Hooks::PRE_INIT, self)
15    Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_SIMULATION, self)
16    @model["env"].as(E).add_observer(self)
17    @sim.initialize_simulation
18  end
19
20  QT_COLOR   = SF.color(44, 149, 44)
21  BLACK      = SF.color(0, 0, 0)
22  WHITE      = SF.color(255, 255, 255)
23  GREEN      = SF.color(34, 139, 34, 255)
24  YELLOW     = SF.color(244, 202, 22)
25  GRID_COLOR = WHITE
26
27  private def draw_grid(renderer, states)
28    HEIGHT.times do |i|
29      line = [
30        SF::Vertex.new(SF.vector2(0, (i + 1)*SCALEY), GRID_COLOR),
31        SF::Vertex.new(SF.vector2(WIDTH*SCALEX, (i + 1)*SCALEY), GRID_COLOR),
32      ]
33      renderer.draw(line, SF::Lines)
34    end
35    WIDTH.times do |i|
36      line = [
37        SF::Vertex.new(SF.vector2((i + 1)*SCALEX, 0), GRID_COLOR),
38        SF::Vertex.new(SF.vector2((i + 1)*SCALEX, HEIGHT*SCALEY), GRID_COLOR),
39      ]
40      renderer.draw(line, SF::Lines)
41    end
42  end
43
44  private def draw_sugar(renderer, states, field)
45    field.matrix.each_by_col do |value, y, x|
46      alpha = value * 50 + 5
47      color = SF.color(244, 202, 22, alpha.to_i)
48
```

```
49        rs = SF::RectangleShape.new({SCALEX, SCALEY})
50        rs.position = {x*SCALEX, y*SCALEY}
51        rs.fill_color = color
52        renderer.draw(rs, states)
53      end
54    end
55
56    PIXELS = WIDTH * HEIGHT
57
58    def update(model)
59      if model.is_a?(E)
60        env = model.as(E)
61        return if env.phase != E::Phase::Natural && env.time > 0
62        qt = env.spatial_index
63        render_states = @render_states.not_nil!
64        @window.try do |window|
65          if window.open?
66            window.clear WHITE
67            draw_sugar(window, render_states, env.field(:sugar))
68            draw_grid(window, render_states)
69            qt.each do |p, e|
70              case e
71              when Quartz::MAS::Body
72                circ = SF::CircleShape.new(0.3*SCALEX)
73                circ.fill_color = SF::Color::Red
74                circ.position = {p.x * SCALEX, p.y * SCALEY}
75                circ.move({0.25*SCALEX, 0.25*SCALEY})
76                window.draw(circ, render_states)
77              end
78            end
79          end
80        end
81      end
82    end
83
84    SCALEX = 16
85    SCALEY = 16
86
87    PIXELS_X = WIDTH * SCALEX
88    PIXELS_Y = HEIGHT * SCALEY
89
90    def notify(hook)
91      case hook
92      when Quartz::Hooks::PRE_INIT
93        settings = SF::ContextSettings.new(depth: 24, antialiasing: 2)
```

```
94        video = SF::VideoMode.new(PIXELS_X.to_i, PIXELS_Y.to_i)
95        @window = SF::RenderWindow.new(video, "SugarScape", settings: settings)
96        @window.not_nil!.framerate_limit = FRAMERATE
97        transform = SF::Transform::Identity
98        @render_states = SF::RenderStates.new(transform: transform)
99        @window.not_nil!.display
100     when Quartz::Hooks::POST_SIMULATION
101       finish
102     end
103   end
104
105   def finish
106     @sim.abort
107     exit
108   end
109
110   def next
111     if @sim.step.nil?
112       finish
113     end
114   end
115
116   DELAY_MAX = 300
117   DELAY_MIN =   0
118   FRAMERATE =  60
119
120   def work
121     pause = true
122     framerate = FRAMERATE
123
124     @window.try do |window|
125       while window.open?
126         while event = window.poll_event
127           if event.is_a? SF::Event::Closed
128             finish
129           elsif event.is_a? SF::Event::KeyPressed
130             case event.code
131             when SF::Keyboard::Escape, SF::Keyboard::Q
132               finish
133             when SF::Keyboard::N
134               self.next if pause
135             when SF::Keyboard::P, SF::Keyboard::Space
136               pause = !pause
137               window.framerate_limit = pause ? 4 : framerate
138             when SF::Keyboard::Up
```

```
139               unless framerate == 200
140                 framerate += 1
141                 window.framerate_limit = framerate
142                 puts "set framerate limit to: #{framerate}fps"
143               end
144             when SF::Keyboard::Down
145               unless framerate == 1
146                 framerate -= 1
147                 window.framerate_limit = framerate
148                 puts "set framerate limit to: #{framerate}fps"
149               end
150             end
151           end
152         end
153
154         if pause
155           window.display
156         else
157           self.next
158           window.display
159         end
160       end
161     end
162   end
163 end
```

Observateurs

```
1  class PopulationPlotter
2    include Quartz::Hooks::Notifiable
3    include Quartz::Observer
4
5    @file : File?
6    property filename : String
7
8    SPACES = 30
9
10   def initialize(@filename = "sugarscape.dat")
11     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::PRE_INIT, self)
12     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_SIMULATION, self)
13     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_ABORT, self)
14   end
15
16   def close
17     @file.try { |f| f.flush; f.close }
```

```
18    end
19
20    def notify(hook)
21      case hook
22      when Quartz::Hooks::PRE_INIT
23        @file = File.new(@filename, "w+")
24        @file.try &.printf("%-#{SPACES}s %-#{SPACES}s %-#{SPACES}s %-#{SPACES}s
              %-#{SPACES}s\n", "time", "population", "starvation", "avg vision", "
              avg metabolism")
25      when Quartz::Hooks::POST_SIMULATION, Quartz::Hooks::POST_ABORT
26        @file.try &.close
27        @file = nil
28      end
29    end
30
31    def update(model)
32      if model.is_a?(Env)
33        env = model.as(Env)
34        return unless env.phase.natural?
35        return unless env.time == env.time.floor
36
37        pop = env.population
38        starvation = env.starvation
39        avg_vision = env.spatial_index.each.map(&.[1].as(MyBody).perception_scope
              .as(Quartz::MAS::CrossScope).dist).sum / pop.to_f
40
41        avg_metabolism = env.network.as(SugarScape).each_child.select(&.is_a?(
              MyAgent)).map(&.as(MyAgent).metabolism).sum / pop.to_f
42
43        @file.try do |file|
44          file.printf("%-#{SPACES}s %-#{SPACES}s %-#{SPACES}s %-#{SPACES}s %-#{
                SPACES}s\n", env.time.to_i, pop, starvation, avg_vision,
                avg_metabolism)
45          file.flush
46        end
47      end
48    end
49  end
50
51  class WealthPlotter
52    include Quartz::Hooks::Notifiable
53    include Quartz::Observer
54
55    @file : File?
56    property filename : String
```

```
57
58  SPACES              = 10
59  HISTOGRAM_INTERVALS = 10
60
61  def close
62    @file.try { |f| f.flush; f.close }
63  end
64
65  def initialize(@filename = "sugarscape_wealth.dat")
66    Quartz::Hooks.notifier.subscribe(Quartz::Hooks::PRE_INIT, self)
67    Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_SIMULATION, self)
68    Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_ABORT, self)
69  end
70
71  def notify(hook)
72    case hook
73    when Quartz::Hooks::PRE_INIT
74      @file = File.new(@filename, "w+").tap do |file|
75        file.printf "%-#{SPACES}s %-#{SPACES}s", "time", "sugarmax"
76        HISTOGRAM_INTERVALS.times do |i|
77          file.printf "%-#{SPACES}s", "tier#{i}"
78        end
79        file.print "\n"
80      end
81    when Quartz::Hooks::POST_SIMULATION, Quartz::Hooks::POST_ABORT
82      @file.try &.close
83      @file = nil
84    end
85  end
86
87  def update(model)
88    if model.is_a?(Env)
89      env = model.as(Env)
90
91      return unless env.phase.natural?
92      return unless env.time == env.time.floor
93      return unless env.time > 290
94
95      @file.try do |file|
96        agents = env.network.as(SugarScape).each_child.select(&.is_a?(MyAgent))
97        sugars = agents.map(&.as(MyAgent).sugar).to_a
98        max = sugars.max
99
100       file.printf "%-#{SPACES}s %-#{SPACES}s", env.time.to_i, max
101
```

```
102         interval = max.to_f / HISTOGRAM_INTERVALS
103         groups = Array(Int32).new(HISTOGRAM_INTERVALS, 0)
104
105         sugars.each { |sugar|
106           i = Math.min(9, (sugar / interval).to_i)
107           groups[i] = groups[i] + 1
108         }
109         groups.each { |pop| file.printf "%-#{SPACES}s", pop }
110
111         file.print "\n"
112         file.flush
113       end
114     end
115   end
116 end
117
118 class LorenzGiniPlotter
119   include Quartz::Hooks::Notifiable
120   include Quartz::Observer
121
122   @file : File?
123   @lorenz : File?
124
125   property gini_filename : String
126   property lorenz_filename : String
127
128   SPACES = 10
129
130   def initialize(@gini_filename = "sugarscape_gini.dat", @lorenz_filename = "
        sugarscape_lorenz.dat")
131     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::PRE_INIT, self)
132     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_SIMULATION, self)
133     Quartz::Hooks.notifier.subscribe(Quartz::Hooks::POST_ABORT, self)
134   end
135
136   def close
137     @file.try { |f| f.flush; f.close }
138     @lorenz.try { |f| f.flush; f.close }
139   end
140
141   def notify(hook)
142     case hook
143     when Quartz::Hooks::PRE_INIT
144       @file = File.new(@gini_filename, "w+").tap do |file|
145         file.printf("%-#{SPACES}s %-#{SPACES}s\n", "time", "gini")
```

```
146       end
147       @lorenz = File.new(@lorenz_filename, "w+").tap do |file|
148         file.printf("%-#{SPACES}s %-#{SPACES}s\n", "time", "lorenz")
149       end
150     when Quartz::Hooks::POST_SIMULATION, Quartz::Hooks::POST_ABORT
151       @file.try &.close
152       @lorenz.try &.close
153       @file = nil
154       @lorenz = nil
155     end
156   end
157
158   def update(model)
159     if model.is_a?(Env)
160       env = model.as(Env)
161
162       return unless env.phase.natural?
163       return unless env.time == env.time.floor
164
165       @file.try do |file|
166         agents = env.network.as(SugarScape).each_child.select(&.is_a?(MyAgent))
167         sorted_wealth = agents.map(&.as(MyAgent).sugar).to_a.sort
168         total_wealth = sorted_wealth.sum
169         wealth_sum_so_far = 0
170         gini_index_reserve = 0.0
171         lorenz_points = Array(Float64).new(initial_capacity: env.population)
172
173         sorted_wealth.each_with_index do |wealth, index|
174           wealth_sum_so_far += wealth
175           lorenz_points << (wealth_sum_so_far.to_f / total_wealth.to_f) * 100.0
176           gini_index_reserve += (index / env.population.to_f) - (
                wealth_sum_so_far / total_wealth.to_f)
177         end
178
179         gini_index = (gini_index_reserve / sorted_wealth.size) * 2
180         file.printf("%-#{SPACES}s %-#{SPACES}s %-#{SPACES}s\n", env.time, "0.0"
              , gini_index)
181         file.flush
182
183         if env.time == 300
184           @lorenz.try do |lorenzfile|
185             lorenz_points.each do |point|
186               lorenzfile.printf "%-#{SPACES}s\n", point
187             end
188             lorenzfile.flush
```

```
189             end
190           end
191         end
192       end
193     end
194 end
```

BIBLIOGRAPHIE

Alonso, F. et al. (2005). « SONIA: A Methodology for Natural Agent Development ». In : *Engineering Societies in the Agents World V*. Sous la dir. de M.-P. Gleizes, A. Omicini et F. Zambonelli. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 245–260 (cf. p. 34).

Atlan, H. (1979). *Entre le cristal et la fumée : essai sur l'organisation du vivant*. Seuil (cf. p. 8).

Austin, J. L. (1975). *How to Do Things with Words*. 2nd edition. Harvard University Press (cf. p. 29).

Axelrod, R. et R. A. Hammond (2003). « The Evolution of Ethnocentric Behavior ». In : *Midwest Political Science Convention*. Chicago, IL, USA (cf. p. 197).

Baati, L., C. Frydman et N. Giambiasi (2007). « LSIS DME M&S environment extended by dynamic hierarchical structure DEVS modeling approach ». In : *Proceedings of the 2007 Spring Simulation Multiconference - Volume 2*. Norfolk, Virginia, USA, p. 227–234 (cf. p. 66).

Bae, J. W. et I.-C. Moon (2016). « LDEF Formalism for Agent-Based Model Development ». In : *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46.6, p. 793–808 (cf. p. 69, 70, 75, 77).

Bae, J. W., G. Lee et I.-C. Moon (2012). « Formal specification supporting incremental and flexible agent-based modeling ». In : *2012 Winter Simulation Conference*. Berlin, Germany : IEEE, 414:1–414:12 (cf. p. 68).

Bae, J. W. et al. (2016). « Efficient Flattening Algorithm for Hierarchical and Dynamic Structure Discrete Event Models ». In : *ACM Transactions on Modeling and Computer Simulation* 26.4, p. 1–25 (cf. p. 150).

Bajracharya, K. et R. Duboz (2013). « Comparison of Three Agent-based Platforms on the Basis of a Simple Epidemiological Model (WIP) ». In : *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*. San Diego, CA, USA : Society for Computer Simulation International, 7:1–7:6 (cf. p. 38).

Balci, O. (1988). « The implementation of four conceptual frameworks for simulation modeling in high-level languages ». In : *Proceedings of the 20th Conference on Winter Simulation*. San Diego, CA : ACM Press, p. 287–295 (cf. p. 40).

Barros, F. J. (1995). « Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modeling and Simulation ». In : *Proceedings of the 27th Conference on Winter Simulation*. Arlington, Virginia, USA : IEEE Computer Society, p. 781–785 (cf. p. 63–65, 72).

BARROS, F. J. (1996a). « Modeling and Simulation of Dynamic Structure Discrete Event Systems: A General Systems Theory Approach ». Thèse de doct. University of Coimbra (cf. p. 63–65).

— (1996b). « The Dynamic Structure Discrete Event System Specification Formalism ». In : *Transactions on Modeling and Computer Simulation* 13.1, p. 35–46 (cf. p. 63–65).

— (1997). « Modeling Formalisms for Dynamic Structure Systems ». In : *ACM Transactions on Modeling and Computer Simulation* 7.4, p. 501–515 (cf. p. 65).

— (1998). « Abstract simulators for the DSDE formalism ». In : *Simulation Conference Proceedings, 1998. Winter*. IEEE, 407–412 vol.1 (cf. p. 63, 65).

— (2005). « A Formal Representation of Hybrid Mobile Components ». In : *SIMULATION* 81.5, p. 381–393 (cf. p. 64, 66).

BAUER, B., J.-P. MÜLLER et J. J. ODELL (2001). « AGENT UML: A FORMALISM FOR SPECIFYING MULTIAGENT SOFTWARE SYSTEMS ». In : *International Journal of Software Engineering and Knowledge Engineering* 11.03, p. 207–230 (cf. p. 70).

BELLIFEMINE, F., A. POGGI et G. RIMASSA (1999). « JADE - A FIPA-compliant agent framework ». In : *Proceedings of the fourth International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology* (cf. p. 30).

BENTAHAR, J., B. MOULIN et B. CHAIB-DRAA (2004). « Commitment and Argument Network: A New Formalism for Agent Communication ». In : *Advances in Agent Communication*. Sous la dir. de F. DIGNUM. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 146–165 (cf. p. 70).

BERG, M. de et al. (2008). *Computational Geometry*. 3rd. Algorithms and Applications. Springer-Verlag (cf. p. 145).

BERGERO, F. et E. KOFMAN (2011). « PowerDEVS: a tool for hybrid system modeling and real-time ». In : *SIMULATION* 87.1-2, p. 113–132 (cf. p. 126).

BERNON, C., M.-P. GLEIZES et G. PICARD (2009). « Méthodes orientées agent et multi-agent ». In : *Technologies des systèmes multi-agents et applications industrielles*. Sous la dir. d'A. EL FALLAH SEGHROUCHNI et J.-P. BRIOT. Hermès Lavoisier (cf. p. 34).

BÉZIVIN, J. (2005). « On the unification power of models ». In : *Software & Systems Modeling* 4.2, p. 171–188 (cf. p. 127).

BIGBEE, A., C. CIOFFI-REVILLA et S. LUKE (2007). « Replication of Sugarscape Using MASON ». In : *Agent-Based Approaches in Economic and Social Complex Systems IV*. Tokyo : Springer Japan, p. 183–190 (cf. p. 178, 191).

BILES, W. E., C. M. DANIELS et T. J. O'DONNELL (1985). « Statistical considerations in simulation on a network of microcomputers ». In : *Proceedings of the 17th conference on winter simulation*. ACM Press, p. 388–393 (cf. p. 159).

BLACKSTONE JR, J. H., G. L. HOGG et D. T. PHILLIPS (1981). « A Two-list Synchronization Procedure for Discrete Event Simulation ». In : *Communications of the ACM* 24.12, p. 825–829 (cf. p. 154).

BLAZY, S. (2008). *Sémantiques formelles*. Habilitation à diriger les recherches. Université d'Evry Val d'Essonne (cf. p. 50).

BOELLA, G., L. van der TORRE et H. VERHAGEN (2006). « Introduction to normative multiagent systems ». In : *Computational & Mathematical Organization Theory* 12.2-3, p. 71–79 (cf. p. 30).

BONAVENTURA, M., G. A. WAINER et R. CASTRO (2013). « Graphical modeling and simulation of discrete-event systems with CD++Builder ». In : *SIMULATION* 89.1, p. 4–27 (cf. p. 126).

BOND, A. H. et L. GASSER, éds. (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann (cf. p. 15).

BOUSQUET, F. et C. LE PAGE (2004). « Multi-agent simulations and ecosystem management: a review ». In : *Ecological Modelling* 176.3–4, p. 313–332 (cf. p. 2).

BOUSQUET, F., C. CAMBIER et P. MORAND (1994). « Distributed artificial intelligence and object-oriented modelling of a fishery ». In : *Mathematical and Computer Modelling* 20.8, p. 97–107 (cf. p. 2).

BRAZIER, F. M. T., C. M. JONKER et J. TREUR (2000). « Compositional design and reuse of a generic agent model ». In : *Applied Artificial Intelligence* 14.5, p. 491–538 (cf. p. 70).

BROOKS, R. A. (1983). « Solving the find-path problem by good representation of free space ». In : *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.2, p. 190–197 (cf. p. 16, 21, 73).

BROWN, R. (1988). « Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem ». In : *Communications of the ACM* 31.10, p. 1220–1227 (cf. p. 154).

BUSH, G., S. CRANEFIELD et M. PURVIS (2001). « The Styx agent methodology ». In : *Information Science Discussion Papers Series No. 2001/02* (cf. p. 34).

BUZING, P. C., A. E. EIBEN et M. C. SCHUT (2005). « Emerging communication and cooperation in evolving agent societies ». In : *Journal of Artificial Societies and Social Simulation* 8 (cf. p. 178).

CADAVID, J. J., B. COMBEMALE et B. BAUDRY (2012). *Ten years of Meta-Object Facility: an Analysis of Metamodeling Practices*. Rapp. tech. 7882. Institut national de recherche en informatique et en automatique (cf. p. 33).

CAMPOS, A. M. C. et al. (2004). « MASIM: A methodology for the development of agent-based simulations ». In : *Proceedings of 16th European Simulation Symposium*. Budapest, Hungary (cf. p. 33, 34, 48).

CAPOCCHI, L. et al. (2011). « DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems ». In : *2011 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, p. 170–175 (cf. p. 126).

CASSANDRAS, C. G. et S. LAFORTUNE (2008). *Introduction to Discrete Event Systems, 2nd edition*. Springer (cf. p. 52).

CERNUZZI, L., M. COSSENTINO et F. ZAMBONELLI (2005). « Process models for agent-based development ». In : *Engineering Applications of Artificial Intelligence* 18.2, p. 205–222 (cf. p. 33, 118).

CHANDY, K. M. et J. MISRA (1979). « Distributed Simulation: A Case Study in Design and Verification of Distributed Programs ». In : *IEEE Transactions on Software Engineering* SE-5.5, p. 440–452 (cf. p. 159).

CHATFIELD, D. C., J. C. HAYYA et T. P. HARRISON (2007). « A multi-formalism architecture for agent-based, order-centric supply chain simulation ». In : *Simulation Modelling Practice and Theory* 15.2, p. 153–174 (cf. p. 70).

CHEN, B. et H. VANGHELUWE (2010). « Symbolic Flattening of DEVS Models ». In : *Proceedings of the 2010 Summer Computer Simulation Conference*. San Diego, CA, USA : Society for Computer Simulation International, p. 209–218 (cf. p. 150).

CHOW, A. C. H. (1996). « Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator ». In : *TRANSACTIONS of the Society for Computer Simulation International* 13, p. 55–67 (cf. p. 66).

CHOW, A. C. H. et B. P. ZEIGLER (1994). « Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism ». In : *Proceedings of the 26th Conference on Winter Simulation*. San Diego, CA, USA : Society for Computer Simulation International, p. 716–722 (cf. p. 53, 149).

CHOW, A. C. H., B. P. ZEIGLER et D. H. KIM (1994). « Abstract simulator for the parallel DEVS formalism ». In : *Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems, 1994. Distributed Interactive Simulation Environments*. IEEE, p. 157–163 (cf. p. 131, 148, 150).

CLARKE, L., I. GLENDINNING et R. HEMPEL (1994). « The MPI Message Passing Interface Standard ». In : *Programming Environments for Massively Parallel Distributed Systems*. Basel : Birkhäuser Basel, p. 213–218 (cf. p. 160).

COMBEMALE, B. (2008). « Approche de métamodélisation pour la simulation et la vérification de modèle ». Thèse de doct. Institut national polytechnique de Toulouse (cf. p. 33).

COMFORT, J. C. (1984). « The simulation of a master-slave event set processor ». In : *SIMULATION* 42.3, p. 117–124 (cf. p. 159).

CONRAD, S., G. SAAKE et C. TÜRKER (2002). « Towards an Agent-Oriented Framework for Specification of Information Systems ». In : *Formal Models of Agents*. Sous la dir. de J.-J. C. MEYER et P.-Y. SCHOBBENS. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 57–73 (cf. p. 70).

COQUILLARD, P. et D. R. C. HILL (1997). *Modélisation et simulation d'écosystèmes*. Des modèles déterministes aux simulations à évènements discrets. Dunod (cf. p. 2).

CORBARA, B. et al. (1993). « Simulating the Sociogenesis Process in Ant Colonies with MANTA ». In : *Toward a Practice of Autonomous Systems Proceedings of the First European Conference on Artificial Life*, p. 224–235 (cf. p. 2, 16, 29).

COSSENTINO, M. et C POTTS (2001). *PASSI: a Process for Specifying and Implementing Multi-Agent Systems Using UML*. Rapp. tech. University of Palermo (cf. p. 34, 49).

COSSENTINO, M. et al. (2009). « ASPECS: an agent-oriented software process for engineering complex systems ». In : *Autonomous Agents and Multi-Agent Systems* 20.2, p. 260–304 (cf. p. 34).

DALLE, O. (2012). « On reproducibility and traceability of simulations ». In : *2012 Winter Simulation Conference*. IEEE, p. 1–12 (cf. p. 36, 38, 40).

DAO, V. T. et al. (2016). « La reproductibilité des simulations stochastiques parallèles et distribuées utilisant le calcul à haute performance ». In : *Journées DEVS Francophones 2016 : Théorie et applications, workshop RED*, p. 109–117 (cf. p. 36, 129).

DÁVILA, J. et K. TUCCI (2000). « Towards a logic-based, multi-agent simulation theory ». In : *IEEE International Conference on Modeling, Simulation and Neural Networks*, p. 37–51 (cf. p. 71).

DÁVILA, J. et M. UZCÁTEGUI (2000). « GALATEA: A multi-agent, simulation platform ». In : *International Conference on Modeling, Simulation and Neural Networks*. Mérida, Venezuela, p. 187–198 (cf. p. 71, 77).

DAVIS, C. K., S. V. SHEPPARD et W. M. LIVELY (1988). « Automatic development of parallel simulation models in ADA ». In : *Proceedings of the 20th Conference on Winter Simulation*. San Diego, California, USA : ACM Press, p. 339–343 (cf. p. 159).

DE LARA, J. et H. VANGHELUWE (2002). « AToM3: A Tool for Multi-formalism and Meta-modelling ». In : *Fundamental Approaches to Software Engineering*. Sous la dir. de R.-D. KUTSCHE et H. WEBER. Springer Berlin Heidelberg, p. 174–188 (cf. p. 128).

DE ROSNAY, J. (1975). *Le Macroscope*. Paris : Seuil (cf. p. 7).

DEFFUANT, G. et al. (2002). « How can extremism prevail? A study based on the relative agreement interaction model ». In : *Journal of Artificial Societies and Social Simulation* 5.4 (cf. p. 2).

DEMAZEAU, Y. (1995). « From Interactions To Collective Behaviour In Agent-Based Systems ». In : *Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, p. 117–132 (cf. p. 20).

DEMAZEAU, Y. et A. C. ROCHA COSTA (1996). « Populations and organizations in open multi-agent systems ». In : *Proceedings of the 1st National Symposium on Parallel and Distributed AI*. Hyderabad, India (cf. p. 30).

DEVS STANDARDIZATION GROUP (2001). *DEVS Standardization Group*. URL : http://cell-devs.sce.carleton.ca/devsgroup/ (cf. p. 127).

Dinu, R., T. Stratulat et J. Ferber (2012). « A Formal Model of Agent Interaction Based on MASQ ». In : *AMPLE'2012: 2nd International Workshop on Agent-based Modeling for PoLicy Engineering*. Montpellier, France (cf. p. 26, 30, 68, 88, 91).

Domingo, C. (1988). « GLIDER, a network oriented simulation language for continuous and discrete event simulation ». In : *International Conference on Mathematical Models*, p. 11–14 (cf. p. 71).

Domingo, C. et M Hernández (1985). *Ideas básicas del lenguaje GLIDER*. Rapp. tech. Mérida, Venezuela : Instituto de Estadıstica Aplicada y Computación, Universidad de Los Andes (cf. p. 71).

Drogoul, A. (1993). « De la Simulation Multi-Agent à la Résolution Collective de Problèmes ». Thèse de doct. Paris : Université Paris VI (cf. p. 2, 16, 21, 29).

Duboz, R. (2004). « Intégration de modèles hétérogènes pour la modélisation et la simulation de systèmes complexes ». Thèse de doct. Université du Littoral - Côte d'Opale (cf. p. 15, 127).

Duboz, R., E. Ramat et N. Giambiasi (2002). « Utilisation du formalisme DEVS pour la spécification de systèmes d'agents réactifs ». In : *Dixièmes journées francophones sur les systèmes multi-agents*. Lille, France, p. 99–102 (cf. p. 87).

Duboz, R. et al. (2005). « Specification of Dynamic Structure Discret event Multiagent Systems ». In : *Agent-directed simulation, part of the 2006 Spring Simulation Multiconference*. Huntsville, AL, USA, SCS, p. 23–30 (cf. p. 72, 76, 77, 79, 88, 101, 127).

Duboz, R., B. Bonté et G. Quesnel (2012). « Vers une spécification des modèles de simulation de systèmes complexes ». In : *Stud. Inform. Univ.* 10.1, p. 7–37 (cf. p. 2, 3, 6, 35, 38, 41, 48, 53, 76, 79, 113).

Edmonds, B. et D. Hales (2003). « Replication, Replication and Replication: Some Hard Lessons from Model Alignment ». In : *Journal of Artificial Societies and Social Simulation* 6 (cf. p. 38, 197).

Elammari, M. et R. Elsaeti (2011). « Multi-Agent System Meta-Models ». In : *Proceedings of the International Arab Conference on Information Technology* (cf. p. 34).

Elammari, M. et W Lalonde (1999). « An agent-oriented methodology: High-Level and Intermediate Models ». In : *Proceedings of Agent-Oriented Information Systems*. Heidelberg, Germany (cf. p. 34).

Epstein, J. M. et R. L. Axtell (1996). *Growing Artificial Societies*. Social Science from the Bottom Up. The MIT Press (cf. p. 163, 178–182, 189, 190, 193–195).

Erman, L. D. et al. (1980). « The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty ». In : *ACM Computing Surveys* 12.2, p. 213–253 (cf. p. 1, 28).

Ferber, J. (1995). *Les systèmes multi-agents. Vers une intelligence collective.* Paris : InterEditions (cf. p. 1, 8, 13, 15, 17, 19, 25, 28, 70).

FERBER, J. et O. GUTKNECHT (1997). *Aalaadin: A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems* (cf. p. 31).

— (1998). « A meta-model for the analysis and design of organizations in multi-agent systems ». In : *International Conference on Multi Agent Systems, 1998. Proceedings*. IEEE, p. 128–135 (cf. p. 30, 34).

FERBER, J. et J.-P. MÜLLER (1996). « Influences and Reaction : a Model of Situated Multiagent Systems ». In : *Proceedings of the Second International Conference on Multiagent Systems* (cf. p. 42, 70, 71, 73, 105, 199).

FERBER, J., F. MICHEL et O. GUTKNECHT (2003). « Agent/Group/Roles: Simulating with Organizations ». In : *ABS'03: Agent Based Simulation*. Montpellier (France) (cf. p. 31, 32, 103).

FERBER, J., O. GUTKNECHT et F. MICHEL (2004). « From Agents to Organizations: An Organizational View of Multi-agent Systems ». In : *Agent-Oriented Software Engineering IV*. Sous la dir. de P. GIORGINI, J. P. MÜLLER et J. J. ODELL. Springer Berlin Heidelberg, p. 214–230 (cf. p. 30).

FERBER, J., F. MICHEL et J.-A. BÁEZ-BARRANCO (2005). « AGRE: Integrating Environments with Organizations ». In : *Environments for Multi-Agent Systems*. Sous la dir. de D. WEYNS, H. V. D. PARUNAK et F. MICHEL. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 48–56 (cf. p. 26, 31, 34, 68, 88, 91, 101).

FINKEL, R. A. et J. L. BENTLEY (1974). « Quad trees a data structure for retrieval on composite keys ». In : *Acta Informatica* 4.1, p. 1–9 (cf. p. 146).

FISHER, M. et M. J. WOOLDRIDGE (1997). « On the Formal Specification and Verification of Multi-Agent Systems ». In : *International Journal of Cooperative Information Systems* 06.01, p. 37–65 (cf. p. 70).

FISHWICK, P. A. (1995). *Simulation Model Design and Execution*. Building Digital Worlds. Prentice Hall (cf. p. 10, 11).

FLENTGE, F., D. POLANI et T. UTHMANN (2001). « Modelling the Emergence of Possession Norms using Memes ». In : *Journal of Artificial Societies and Social Simulation* 4 (cf. p. 178).

FOERSTER, H. von (2003). « On Self-Organizing Systems and Their Environments ». In : *Understanding Understanding*. New York, NY : Springer New York, p. 1–19 (cf. p. 16).

FORRESTER, J. W. (1980). *Principes des systèmes*. Presses Universitaires de Lyon (cf. p. 7).

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (2002a). *FIPA ACL Message Structure Specification*. FIPA TC Communication (cf. p. 30).

— (2002b). *FIPA Contract Net Interaction Protocol Specification*. FIPA TC Communication (cf. p. 30).

FOURES, D. (2015). « Validation de modèles de simulation ». Thèse de doct. Toulouse, France (cf. p. 128).

FOWLER, M. (2010). *Domain-Specific Languages*. Addison-Wesley (cf. p. 128, 137, 138).

FRANCESCHINI, R. et al. (2014a). « A survey of modelling and simulation software frameworks using Discrete Event System Specification ». In : *2014 Imperial College Computing Student Workshop*. London, UK : Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, p. 40–49 (cf. p. 126, 150, 213).

FRANCESCHINI, R., P.-A. BISGAMBIGLIA et P. BISGAMBIGLIA (2014b). « Decentralized Approach for Efficient Simulation of Devs Models ». In : *Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 336–343 (cf. p. 152).

FRANCESCHINI, R. et al. (2014c). « DEVS-Ruby: a Domain Specific Language for DEVS Modeling and Simulation (WIP) ». In : *DEVS 14: Proceedings of the Symposium on Theory of M&S*. SCS International, p. 393–398 (cf. p. 126, 128, 129, 137, 161).

FRANCESCHINI, R., P.-A. BISGAMBIGLIA et P. BISGAMBIGLIA (2015). « A comparative study of pending event set implementations for PDEVS simulation ». In : *DEVS 15: Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Alexandria, VA, United States, p. 836–843 (cf. p. 152, 153, 214, 215).

— (2017). « Approche formelle pour la modélisation et la simulation de systèmes multi-agents ». In : *Vingt-cinquièmes Journées Francophones sur les Systèmes Multi-Agents*. Caen, France, p. 193–202 (cf. p. 80).

FUJIMOTO, R. M. (1990). « Parallel Discrete Event Simulation ». In : *Communications of the ACM* 33.10, p. 30–53 (cf. p. 159).

GALÁN, J. M. et al. (2009). « Errors and Artefacts in Agent-Based Modelling ». In : *Journal of Artificial Societies and Social Simulation* 12 (cf. p. 193).

GAMMA, E. et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Sofware*. Addison Wesley (cf. p. 131).

GARREDU, S. et al. (2011). « A methodology to specify DEVS domain specific profiles and create profile-based models ». In : *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*. IEEE, p. 353–359 (cf. p. 128).

— (2012). « Enriching a DEVS meta-model with OCL constraints ». In : *EMSS'12*. Vienne, Austria (cf. p. 128).

GASSER, L. (2001). « Perspectives on Organizations in Multi-agent Systems ». In : *Multi-Agent Systems and Applications*. Sous la dir. de M. LUCK et al. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 1–16 (cf. p. 30).

GELERNTER, D. et N. CARRIERO (1992). « Coordination languages and their significance ». In : *Communications of the ACM* 35.2, p. 96 (cf. p. 29).

GHEZZI, C., M. JAZAYERI et D. MANDRIOLI (2002). *Fundamentals of Software Engineering*. 2nd. Upper Saddle River, NJ, USA : Prentice Hall PTR (cf. p. 33).

GLEIZES, M.-P. et al. (2009). « Méthodes de dévéloppement de systèmes multi-agents ». In : *Génie Logiciel* 86, p. 66–80 (cf. p. 20, 33, 34).

GOLDBERG, D. (1991). « What every computer scientist should know about floating-point arithmetic ». In : *ACM Computing Surveys* 23.1, p. 5–48 (cf. p. 135).

GOLDSTEIN, R., S. BRESLAV et A. KHAN (2017). « Practical aspects of the DesignDEVS simulation environment ». In : *SIMULATION* (cf. p. 137).

GOMEZ-SANZ, J. J. et J. PAVÓN (2002). « Meta-modelling in Agent Oriented Software Engineering ». In : *Advances in Artificial Intelligence — IBERAMIA 2002*. Sous la dir. de M. C. MONARD et J. S. SICHMAN. Springer Berlin Heidelberg, p. 606–615 (cf. p. 20).

GRASSÉ, P.-P. (1959). « La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes Natalensis et Cubitermes sp. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs ». In : *Insectes Sociaux* 6.1, p. 41–80 (cf. p. 29).

GRIGNARD, A. et al. (2013). « GAMA 1.6: Advancing the Art of Complex Agent-Based Modeling and Simulation ». In : *PRIMA 2013: Principles and Practice of Multi-Agent Systems*. Sous la dir. de G. BOELLA et al. Berlin, Heidelberg : Springer, p. 117–131 (cf. p. 24, 25, 38, 40).

GRIMM, V. (1999). « Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future? » In : *Ecological Modelling* 115.2–3, p. 129–148 (cf. p. 14, 41).

GRIMM, V. et al. (2006). « A standard protocol for describing individual-based and agent-based models ». In : *Ecological Modelling* 198.1–2, p. 115–126 (cf. p. 41).

GRIMM, V. et al. (2010). « The ODD protocol: A review and first update ». In : *Ecological Modelling* 221.23, p. 2760–2768 (cf. p. 41, 119).

GRUER, J. P. et al. (2004). « Heterogeneous formal specification based on Object-Z and statecharts: semantics and verification ». In : *Journal of Systems and Software* 70.1-2, p. 95–105 (cf. p. 70).

GUTKNECHT, O., J. FERBER et F. MICHEL (2001). « Integrating tools and infrastructures for generic multi-agent systems ». In : *Proceedings of the fifth International Conference on Autonomous Agents*. New York, USA : ACM Press, p. 441–448 (cf. p. 31).

HAMER, W. H. (1906). *The Milroy lectures on epidemic disease in England; the evidence of variability and of persistency of type*. London : Bedford Press (cf. p. 13, 25).

HAN, Y. et al. (2010). « System entity structure for XML meta data modeling: application to the logistics ». In : *the 2010 Spring Simulation Multiconference*. New York, USA : ACM Press, p. 1 (cf. p. 117).

HARDEBOLLE, C. (2008). « Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes ». Thèse de doct. Université Paris-Sud XI (cf. p. 49).

HAUGELAND, J. (1989). *Artificial Intelligence: The Very Idea*. MIT Press. A Bradford Book (cf. p. 87).

HEIDELBERGER, P. (1986). « Statistical analysis of parallel simulations ». In : *Proceedings of the 18th conference on Winter Simulation*. ACM Press, p. 290–295 (cf. p. 159).

HENDERSON, S. G. (2003). « Input modeling: input model uncertainty: why do we care and what should we do about it? » In : *Proceedings of the 35th conference on winter simulation: driving innovation*. New Orleans, Louisian, p. 90–100 (cf. p. 159).

HILAIRE, V. et al. (2000). « Formal Specification and Prototyping of Multi-agent Systems ». In : *Engineering Societies in the Agents World*. Sous la dir. d'A. OMICINI, R. TOLKSDORF et F. ZAMBONELLI. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 114–127 (cf. p. 68, 91, 101).

HILL, D. R. C. (1993). *Analyse orientée objets et modélisation par simulation*. Addison-Wesley France (cf. p. 11).

— (2010). *Simulation informatique au service des Sciences de la Vie*. Presses Universitaires Blaise-Pascal (cf. p. 10).

HIMMELSPACH, J. et A. M. UHRMACHER (2006). « Sequential Processing of PDEVS models ». In : *Proceedings of the 3rd EMSS*, p. 239–244 (cf. p. 148, 150).

— (2007a). « Plug'n simulate ». In : *In Proceedings of the 40th Annual Simulation Symposium (2007)*. IEEE, p. 137–143 (cf. p. 127).

— (2007b). « The Event Queue Problem and PDEVS ». In : *Proceedings of the 2007 Spring Simulation Multiconference - Volume 2*. San Diego, CA, USA : Society for Computer Simulation International, p. 257–264 (cf. p. 152, 153, 214, 216).

HU, X. (2005). « Variable Structure in DEVS Component-Based Modeling and Simulation ». In : *SIMULATION* 81.2, p. 91–102 (cf. p. 66).

HUDAK, P (1998). « Modular domain specific languages and tools ». In : *Fifth International Conference on Software Reuse, 1998. Proceedings*. IEEE, p. 134–142 (cf. p. 138).

IEEE (2008). *IEEE Standard for Floating-Point Arithmetic* (cf. p. 135).

IGARASHI, A. et H. NAGIRA (2006). « Union types for object-oriented programming ». In : *the 2006 ACM symposium*. New York, New York, USA : ACM Press, p. 1435 (cf. p. 130).

ISO et IEC (1996). *Information technology — Syntactic metalanguage — Extended BNF*. International Organization for Standardization / International Electrotechnical Commission (cf. p. 49).

IZQUIERDO, L. R. et al. (2009). « Techniques to Understand Computer Simulations: Markov Chain Analysis ». In : *Journal of Artificial Societies and Social Simulation* 12.6 (cf. p. 178).

JAFER, S. et G. A. WAINER (2009). « Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS ». In : *International Conference on Computational Science and Engineering, 2009. CSE '09*. IEEE, p. 443–448 (cf. p. 150).

JEFFERSON, D. R. et H. A. SOWIZRAL (1982). *Fast Concurrent Simulation Using the Time Warp Mechanism*. Rapp. tech. N-1906-AF. Santa Monica, CA, USA : RAND Corporation (cf. p. 160).

JONES, D. W. (1986). « An Empirical Comparison of Priority-queue and Event-set Implementations ». In : *Communications of the ACM* 29.4, p. 300–311 (cf. p. 152).

KAELBLING, L. P., M. L. LITTMAN et A. R. CASSANDRA (1998). « Planning and acting in partially observable stochastic domains ». In : *Artificial Intelligence* 101.1-2, p. 99–134 (cf. p. 69).

KAMINKA, G. A., D. V. PYNADATH et M TAMBE (2002). « Monitoring Teams by Overhearing: A Multi-Agent Plan-Recognition Approach ». In : *Journal of Artificial Intelligence Research* 17, p. 83–135 (cf. p. 30).

KANG, K. et al. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Rapp. tech. CMU/SEI-90-TR-021. Pittsburgh, PA, USA : Software Engineering Institute, Carnegie Mellon University (cf. p. 118).

KIM, J.-H. et T. G. KIM (2000). « Framework for modeling/simulation of mobile agent systems ». In : *Proceedings of 2000 Conference on AI, Simulation and Planning in High Autonomy Systems*, p. 53–59 (cf. p. 66).

KIM, S., H. S. SARJOUGHIAN et V. ELAMVAZHUTHI (2009). « DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring ». In : *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, p. 161 (cf. p. 126).

KITANO, H. et al. (1997). « RoboCup ». In : *the first international conference*. New York, USA : ACM Press, p. 340–347 (cf. p. 2).

KLÜGL, F., M. FEHLER et R. HERRLER (2005). « About the Role of the Environment in Multi-agent Simulations ». In : *Environments for Multi-Agent Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 127–149 (cf. p. 17).

KOFMAN, E. (2002). « A Second-Order Approximation for DEVS Simulation of Continuous Systems ». In : *SIMULATION* 78.2, p. 76–89 (cf. p. 52).

KORNFELD, W. A. (1979). « ETHER: a parallel problem solving system ». In : *Proceedings of the 6th international joint conference on Artificial Intelligence*. Tokyo, Japan, p. 490–492 (cf. p. 1).

KUBERA, Y., P. MATHIEU et S. PICAULT (2011). « IODA: an interaction-oriented approach for multi-agent based simulations ». In : *Autonomous Agents and Multi-Agent Systems* 23.3, p. 303–343 (cf. p. 33, 34, 118–120).

LANGTON, C. G. (1989). *Artificial Life: The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc. (cf. p. 1, 16).

LAWSON, B. G. et S. PARK (2000). « Asynchronous Time Evolution in an Artificial Society Model ». In : 3.1 (cf. p. 39, 45, 178, 189, 192, 195).

LE HUNG, V., D. FOURES et V. ALBERT (2015). « ProDEVS : an Event-Driven Modeling and Simulation Tool for Hybrid Systems using State Diagrams ». In : *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*. Athens, Greece : ACM, p. 29–37 (cf. p. 95, 126, 128).

LE MOIGNE, J.-L. (1977). *La théorie du système général*. Théorie de la modélisation. Presses Universitaires de France (cf. p. 7, 8).

Le Moigne, J.-L. (1999). *La modélisation des systèmes complexes*. Dunod. Broché (cf. p. 8).

Lenat, D. B. (1975). « Beings: knowledge as interacting experts ». In : *Proceedings of the 4th international joint conference on Artificial Intelligence*. Tblisi, USSR, p. 126–133 (cf. p. 1).

Levine, J. R., T. Mason et D. Brown (1992). *Lex & yacc*. 2nd. Sebastopol, CA, USA : O'Reilly & Associates (cf. p. 137).

Lomazova, I. A. (2000). « Nested Petri Nets — a Formalism for Specification and Verification of Multi-Agent Distributed Systems ». In : *Fundamenta Informaticae* 43.1-4, p. 195–214 (cf. p. 70).

Lorenz, M. O. (1905). « Methods of Measuring the Concentration of Wealth ». In : *Publications of the American Statistical Association* 9.70, p. 209 (cf. p. 183).

Lotka, A. J. (1925). *Elements of physical biology*. Baltimore : Williams et Wilkins Company (cf. p. 13, 164).

Luck, M. et M DInverno (1995). « A formal framework for agency and autonomy ». In : *Proceedings of the First International Conference on Multiagent Systems*. Menlo Park, CA, USA, p. 254–260 (cf. p. 70).

Luke, S. et al. (2005). « MASON: A Multiagent Simulation Environment ». In : *SIMULATION* 81.7, p. 517–527 (cf. p. 191).

MacKenzie, D. C., R. Arkin et J. M. Cameron (1997). « Multiagent Mission Specification and Execution ». In : *Autonomous Robots* 4.1, p. 29–52 (cf. p. 2).

Maes, P. (1995). « Artificial life meets entertainment: lifelike autonomous agents ». In : *Communications of the ACM* 38, p. 108–114 (cf. p. 1).

Maione, G et D Naso (2003). « A discrete-event system model for multi-agent control of automated manufacturing systems ». In : *SMC '03 2003 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, p. 1723–1728 (cf. p. 51).

Marzougui, B., K. Hassine et K. Barkaoui (2010). « A New Formalism for Modeling a Multi Agent Systems: Agent Petri Nets ». In : *Journal of Software Engineering and Applications* 03.12, p. 1118–1124 (cf. p. 70).

Mathieu, P., J.-C. Routier et Y. Secq (2003). « RIO : Rôles, Interaction et Organisations ». In : *Secondes Journées Francophones sur les Modèles Formels de l'Interaction*. Lille, France, p. 179–188 (cf. p. 34).

Mathieu, P., S. Picault et Y. Secq (2016). « Design patterns pour les environnements dans les simulations multi-agents ». In : *Revue dIntelligence Artificielle* 30.1-2, p. 133–158 (cf. p. 91, 97, 99).

McCormack, W. M. et R. G. Sargent (1981). « Analysis of Future Event Set Algorithms for Discrete Event Simulation ». In : *Communications of the ACM* 24.12, p. 801–812 (cf. p. 152).

Metropolis, N. et S Ulam (1949). « The Monte Carlo Method ». In : *Journal of the American Statistical Association* 44.247, p. 335–341 (cf. p. 13).

MICHEL, F. (2004). « Formalisme, outils et éléments methodologiques pour la modélisation et la simulation multi-agents ». Thèse de doct. Université Montpellier II (cf. p. 3, 35, 37–39, 41).

— (2007a). « Le modèle IRM4S. De l'utilisation des notions d'influence et de réaction pour la simulation de systèmes multi-agents ». In : *Revue dIntelligence Artificielle* 21.5-6, p. 757–779 (cf. p. 26, 27, 42, 43, 66, 67, 70, 88, 90, 105, 199).

— (2007b). « The IRM4S Model: The Influence/Reaction Principle for Multiagent Based Simulation ». In : *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, New York, USA : ACM Press, p. 1–3 (cf. p. 105).

MILLISCHER, L. (2000). « Modélisation individu-centrée des comportements de recherche des navires de pêche ». Thèse de doct. Rennes (cf. p. 15).

MINSKY, M. L. (1965). « Matter, minds, models ». In : *Proceedings of International Federation for Information Processing*, p. 45–49 (cf. p. 10).

MORIN, E. (1977). *La Nature de la nature*. T. 1. La Méthode. Seuil (cf. p. 8).

MOSTERMAN, P. J. et G. BISWAS (2000). « A comprehensive methodology for building hybrid models of physical systems ». In : *Artificial Intelligence* 121.1-2, p. 171–209 (cf. p. 75).

MÜLLER, J.-P. (2004). « The MIMOSA generic modelling and simulation platform: the case of multi-agent systems ». In : *Proceedings of the 5th workshop on agent-based simulation*. Lisbon, p. 77–86 (cf. p. 73, 127).

— (2009). « Towards a Formal Semantics of Event-Based Multi-agent Simulations ». In : *Multi-Agent-Based Simulation IX*. Sous la dir. de N. DAVID et J. S. SICHMAN. Berlin, Heidelberg : Springer, p. 110–126 (cf. p. 73, 77, 127).

MÜLLER, J. P. (1996). « The agent architecture interrap ». In : *The Design of Intelligent Agents*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 45–123 (cf. p. 22).

MUZY, A. et J. NUTARO (2005). « Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators ». In : *2008 12th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*. IEEE, p. 273–279 (cf. p. 148).

MUZY, A. et B. P. ZEIGLER (2014). « Specification of dynamic structure discrete event systems using single point encapsulated control functions ». In : *International Journal of Modeling, Simulation, and Scientific Computing* 05.03, p. 1450012 (cf. p. 64).

NEWELL, A. (1969). « Heuristic Programming: Ill-Structured Problems ». In : *Progress in Operations Research, Volume III* (cf. p. 28).

NIAZI, M. A. et A. HUSSAIN (2011). « A Novel Agent-Based Simulation Framework for Sensing in Complex Adaptive Environments ». In : *IEEE Sensors Journal* 11.2, p. 404–412 (cf. p. 70).

NIKOLOPOULOS, S. D. et R. MACLEOD (1993). « An experimental analysis of event set algorithms for discrete event simulation ». In : *Microprocessing and Microprogramming* 36.2, p. 71–81 (cf. p. 152).

North, M. J. et al. (2013). « Complex adaptive systems modeling with Repast Simphony ». In : *Complex Adaptive Systems Modeling* 1.1, p. 3 (cf. p. 178).

Nutaro, J. (1999). « ADEVS (A Discrete EVent System simulator) ». In : *Arizona Center for Integrative Modeling & Simulation (ACIMS), University of Arizona, Tucson. Available at http://www. ece. arizona. edu/nutaro/index. php* (cf. p. 126, 150, 213).

Object Management Group (2014). *Object Constraint Language* (cf. p. 50).

— (2015). *OMG Unified Modeling Language (OMG UML)* (cf. p. 68, 127).

Odell, J. J. et al. (2003a). « Modeling Agents and Their Environment ». In : *Agent-Oriented Software Engineering III*. Sous la dir. de F. Giunchiglia, J. J. Odell et G. Weiss. Springer Berlin Heidelberg, p. 16–31 (cf. p. 17, 18, 91).

Odell, J. J., H. V. D. Parunak et M. Fleischer (2003b). « The Role of Roles in Designing Effective Agent Organizations ». In : *Software Engineering for Large-Scale Multi-Agent Systems*. Sous la dir. d'A. Garcia et al. Berlin, Heidelberg : Springer, p. 27–38 (cf. p. 34).

Oh, S. et J. Ahn (1999). « Dynamic calendar queue ». In : *Simulation Symposium, 1999. Proceedings. 32nd Annual*. IEEE Comput. Soc, p. 20–25 (cf. p. 155).

Oliveira, E., K. Fischer et O. Štěpánková (1999). « Multi-agent systems: which research for which applications ». In : *Robotics and Autonomous Systems* 27.1-2, p. 91–106 (cf. p. 2).

Pasquier, P., R. A. Flores et B. Chaib-draa (2005). « Modelling Flexible Social Commitments and Their Enforcement ». In : *Engineering Societies in the Agents World V*. Sous la dir. d'A. Omicini, R. Tolksdorf et F. Zambonelli. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 139–151 (cf. p. 30).

Pavón, J., C. Sansores et J. J. G. Sanz (2008). « Modelling and simulation of social systems with INGENIAS ». In : *International Journal of Agent-Oriented Software Engineering* 2.2, p. 196–221 (cf. p. 34).

Picard, G. et M.-P. Gleizes (2004). « The ADELFE Methodology ». In : *Methodologies and Software Engineering for Agent Systems*. Boston : Kluwer Academic Publishers, p. 157–175 (cf. p. 34, 49).

Pnueli, A. (1986). « Specification and development of reactive systems ». In : *IFIP Congress*, p. 845–858 (cf. p. 16, 20, 88).

Poslad, S. (2007). « Specifying protocols for multi-agent systems interaction ». In : *ACM Transactions on Autonomous and Adaptive Systems* 2.4 (cf. p. 29, 30).

Quesnel, G. (2006). « Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes ». Thèse de doct. Université du Littoral - Côte d'Opale (cf. p. 72, 77, 107, 127).

Quesnel, G., R. Duboz et E. Ramat (2009). « The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems ». In : *Simulation Modelling Practice and Theory* 17.4, p. 641–653 (cf. p. 126, 127).

RAMAT, E. (2003). *Contributions à la modélisation et à la simulation des systèmes complexes*. Habilitation à diriger des recherches (cf. p. 51, 52).

RAO, A. S. et M. P. GEORGEFF (1992). « An abstract architecture for rational agents ». In : *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*. Cambridge, MA, USA, p. 439–449 (cf. p. 21, 22, 69, 88).

RED (2014). *DEVS Network*. URL : https://devs-network.org/About (cf. p. 127).

REUILLON, R., M. LECLAIRE et S. REY-COYREHOURCQ (2013). « OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models ». In : *Future Generation Computer Systems* 29.8, p. 1981–1990 (cf. p. 161).

REYNOLDS JR., P. F. (1988). « A spectrum of options for parallel simulation ». In : *Proceedings of the 20th Conference on Winter Simulation*. New York, New York, USA : ACM Press, p. 325–332 (cf. p. 148).

RICCI, A., M. VIROLI et M. PIUNTI (2010). « Formalising the Environment in MAS Programming: A Formal Model for Artifact-Based Environments ». In : *Programming Multi-Agent Systems*. Sous la dir. de M. M. DASTANI, J. DIX et A. EL FALLAH SEGHROUCHNI. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 133–150 (cf. p. 70).

RICHIARDI, M. et al. (2006). « A Common Protocol for Agent-Based Social Simulation ». In : *Journal of Artificial Societies and Social Simulation* 9.1 (cf. p. 2).

RODRIGUEZ, S. et al. (2007). « An Analysis and Design Concept for Self-organization in Holonic Multi-agent Systems ». In : *Engineering Self-Organising Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 15–27 (cf. p. 34).

RÖNNGREN, R., J. RIBOE et R. AYANI (1991). « Lazy Queue: An Efficient Implementation of the Pending-event Set ». In : *Proceedings of the 24th Annual Symposium on Simulation*. Los Alamitos, CA, USA : IEEE Computer Society Press, p. 194–204 (cf. p. 215).

ROUCHIER, J. (2003). « Re-implementation of a multi-agent model aimed at sustaining experimental economic research: The case of simulations with emerging speculation ». In : *Journal of Artificial Societies and Social Simulation* 6 (cf. p. 35, 197).

RUSSEL, S. J. et P. NORVIG (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall (cf. p. 18, 23, 26, 39).

— (2002). *Artificial Intelligence: A Modern Approach (2nd edition)*. Prentice Hall (cf. p. 16, 19).

SANTORO, T et F. QUAGLIA (2010). « A low-overhead constant-time LTF scheduler for optimistic simulation systems ». In : *2010 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, p. 948–953 (cf. p. 155).

SARGENT, R. G. (2005). « Verification and validation of simulation models ». In : *Proceedings of the 37th Conference on Winter Simulation*, p. 130–143 (cf. p. 37, 48, 118).

SAUNIER, J. (2015). « De l'intérêt de la cognition incarnée pour les agents logiciels ». In : *23èmes Journées Francophones sur les Systèmes Multi-Agents*. Rennes, France, p. 101–110 (cf. p. 87, 89).

SCHATTENBERG, B et A. M. UHRMACHER (2001). « Planning agents in JAMES ». In : *Proceedings of the IEEE* 89.2, p. 158–173 (cf. p. 127).

SEARLE, J. R. (1969). *Speech Acts*. An essay in The philosophy of Language. Cambridge : Cambridge University Press (cf. p. 29).

— (2005). « What is an institution? » In : *Journal of Institutional Economics* 1.1, p. 1–22 (cf. p. 30).

SELIC, B (2003). « The pragmatics of model-driven development ». In : *IEEE Software* 20.5, p. 19–25 (cf. p. 127).

SEO, C. et al. (2013). « DEVS Modeling and simulation methodology with MS4 Me software tool ». In : *DEVS 13: Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*. San Diego, CA, USA, p. 1–8 (cf. p. 126).

SHANG, H. et G. A. WAINER (2006). « A Simulation Algorithm for Dynamic Structure DEVS Modeling ». In : *Proceedings of the 38th Conference on Winter Simulation*. Monterey, California : Winter Simulation Conference, p. 815–822 (cf. p. 66).

SHANNON, C. E. et W. WEAVER (1949). *The Mathematical Theory of Communication*. University of Illinois Press (cf. p. 29).

SHANNON, R. E. (1998). « Introduction to the art and science of simulation ». In : *Proceedings of the 30th Conference on Winter Simulation*. Washington, D.C., USA, p. 7–14 (cf. p. 118).

SILVA, L. P. da (2005). « A Formal Model for the Fifth Discipline ». In : *Journal of Artificial Societies and Social Simulation* 8.3, p. 6 (cf. p. 70).

SOULIÉ, J.-C. (2001). « Vers une approche multi-environnements pour les agents ». Thèse de doct. Université de la Rénunion (cf. p. 19, 25, 26, 45, 72, 88, 100).

— (2012). *Multi-modélisation & Simulation de Systèmes Complexes : de la théorie à l'application*. Habilitation à Diriger les Recherches (cf. p. 107).

STACHOWIAK, H. (1973). *Allgemeine Modelltheorie*. Springer (cf. p. 9).

STEEN, L. A. et J. A. SEEBACH JR (1978). *Counterexamples in Topology*. New York, NY, USA : Springer (cf. p. 97).

STEINBERG, D. et al. (2008). *EMF: Eclipse Modeling Framework*. 2nd. Broché (cf. p. 49).

STEINIGER, A., F. KRUGER et A. M. UHRMACHER (2012). « Modeling agents and their environment in Multi-Level-DEVS ». In : *2012 Winter Simulation Conference*. IEEE, p. 1–12 (cf. p. 71, 127).

STODDEN, V et al. (2013). *Setting the default to reproducible: reproducibility in computational and experimental mathematics*. Rapp. tech. (cf. p. 35, 36, 38, 40, 50, 201).

STRATULAT, T., J. FERBER et J. TRANIER (2009). « MASQ: towards an integral approach to interaction ». In : *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. Utrecht University. International Foundation for Autonomous Agents et Multiagent Systems, p. 813–820 (cf. p. 34, 35).

Su, R. (2013). « Discrete-event modeling of multi-agent systems with broadcasting-based parallel composition ». In : *Automatica* 49.11, p. 3502–3506 (cf. p. 51).

Tan, K. L. et L.-J. Thng (2000). « SNOOPy Calendar Queue ». In : *Proceedings of the 32nd Conference on Winter Simulation*. Orlando, Florida : Society for Computer Simulation International, p. 487–495 (cf. p. 155).

Tang, W. T., R. S. M. Goh et I. L.-J. Thng (2005). « Ladder Queue: An O(1) Priority Queue Structure for Large-scale Discrete Event Simulation ». In : *ACM Transactions on Modeling and Computer Simulation* 15.3, p. 175–204 (cf. p. 156).

Touraille, L., M. K. Traoré et D. R. C. Hill (2010). *SimStudio : une Infrastructure pour la Modélisation, la Simulation et l'Analyse de Systèmes Dynamiques Complexes*. Rapp. tech. LIMOS - Laboratoire d'Informatique, de Modélisation et d'optimisation des Systèmes (cf. p. 128).

Traoré, M. K. et A. Muzy (2006). « Capturing the dual relationship between simulation models and their context ». In : *Simulation Modelling Practice and Theory* 14.2, p. 126–142 (cf. p. 41).

Uhrmacher, A. M. (2001). « Dynamic Structures in Modeling and Simulation: A Reflective Approach ». In : *ACM Transactions on Modeling and Computer Simulation* 11.2, p. 206–232 (cf. p. 64, 66, 71–73).

Uhrmacher, A. M. et R Arnold (1994). « Distributing and maintaining knowledge: agents in variable structure environments ». In : *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*. IEEE Comput. Soc. Press, p. 178–184 (cf. p. 71, 87, 88).

Uhrmacher, A. M. et C. Kuttler (2006). « Multi-Level Modeling in Systems Biology by Discrete Event Approaches ». In : *it - Information Technology* 48, p. 148–153 (cf. p. 76).

Uhrmacher, A. M. et B Schattenberg (1998). « Agents in discrete event simulation ». In : *European Simulation Symposium* (cf. p. 53, 71, 76, 77).

Uhrmacher, A. M., P. Tyschler et D. Tyschler (2000). « Modeling and simulation of mobile agents ». In : *Future Generation Computer Systems* 17.2, p. 107–118 (cf. p. 127).

Uhrmacher, A. M. et al. (2006a). « Introducing Variable Ports and Multi-Couplings for Cell Biological Modeling in DEVS ». In : *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*. IEEE, p. 832–840 (cf. p. 51, 71).

— (2006b). « Introducing Variable Ports and Multi-Couplings for Cell Biological Modeling in DEVS ». In : *2006 Winter Simulation Conference*. IEEE, p. 832–840 (cf. p. 64, 66).

Urbani, D. (2006). « Elaboration d'une approche hybride SMA-SIG pour la définition d'un système d'aide à la décision ». Thèse de doct. Université de Corse Pascal Paoli (cf. p. 2).

Van Tendeloo, Y. et H. Vangheluwe (2014). « The Modular Architecture of the Python(P)DEVS Simulation Kernel Work In Progress paper ». In : *Proceedings of the Symposium On Theory of Modeling and Simulation (TMS'14) SpringSim 2014*. Tampa, FL, USA : SCS, p. 387–392 (cf. p. 126).

Van Tendeloo, Y. et H. Vangheluwe (2016). « An evaluation of DEVS simulation tools ». In : *SIMULATION* (cf. p. 126, 129, 150, 213).

Vangheluwe, H. (2000). « DEVS as a common denominator for multi-formalism hybrid systems modelling ». In : *IEEE International Symposium on Computer-Aided Control System Design, 2000. CACSD 2000*. IEEE, p. 129–134 (cf. p. 52, 54, 80).

Varela, F. J. (1989). *Autonomie et connaissance : essai sur le vivant*. Seuil (cf. p. 8).

Vaucher, J. G. et P. Duval (1975). « A Comparison of Simulation Event List Algorithms ». In : *Communications of the ACM* 18.4, p. 223–230 (cf. p. 152, 214).

Vicino, D., O. Dalle et G. A. Wainer (2014). « A data type for discretized time representation in DEVS ». In : *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*. ICST, p. 11–20 (cf. p. 135).

Vicino, D. et al. (2015). « Sequential PDEVS Architecture ». In : *DEVS 15: Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Alexandria, VA, USA, p. 906–913 (cf. p. 148, 150).

Voelter, M. et V. Pech (2012). « Language modularity with the MPS language workbench ». In : *2012 34th International Conference on Software Engineering (ICSE 2012)*. Zurich, Switzerland : IEEE, p. 1449–1450 (cf. p. 137).

Volterra, V. (1926). « Fluctuations in the Abundance of a Species considered Mathematically ». In : *Nature* 118, p. 558–560 (cf. p. 13, 164).

— (1928). « Variations and Fluctuations of the Number of Individuals in Animal Species living together ». In : *ICES Journal of Marine Science* 3.1, p. 3–51 (cf. p. 13).

Von Bertalanffy, L. (1968). *Théorie générale des systèmes*. Dunod (cf. p. 7).

Von Neumann, J. et A. W. Burks (1966). *Theory of Self-Reproducting Automata*. Champaign, IL, USA : University of Illinois Press (cf. p. 14).

Wainer, G. A. (2000). « Improved Cellular Models with Parallel Cell-DEVS ». In : *TRANSACTIONS of The Society for Modeling and Simulation International* 17.2, p. 73–88 (cf. p. 161).

— (2002). « CD++: a toolkit to develop DEVS models ». In : *Software: Practice and Experience* 32.13, p. 1261–1306 (cf. p. 126).

Wainer, G. A. et N. Giambiasi (2001). « Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation ». In : *SIMULATION* 76.1, p. 22–39 (cf. p. 63, 72, 74, 91, 203).

Wainer, G. A., E. Glinsky et M. Gutierrez-Alcaraz (2011). « Studying performance of DEVS modeling and simulation environments using the DEVStone benchmark ». In : *SIMULATION* 87.7, p. 555–580 (cf. p. 150, 213).

Weiss, Gerhard (1999). *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. Sous la dir. de G. Weiss. MIT Press (cf. p. 16, 17).

WEYNS, D. et T. HOLVOET (2003). « Model for Simultaneous Actions in Situated Multi-agent Systems ». In : *Multiagent System Technologies*. Sous la dir. de M. SCHILLO et al. Springer Berlin Heidelberg, p. 105–118 (cf. p. 67).

— (2004). « A Formal Model for Situated Multi-Agent Systems ». In : *Fundamenta Informaticae* 63, p. 125–158 (cf. p. 27, 70).

WEYNS, D., H. V. D. PARUNAK et F. MICHEL, éds. (2005a). *Environments for Multi-Agent Systems*. T. 3374. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer (cf. p. 102).

WEYNS, D. et al. (2005b). « Environments for Multiagent Systems. State-of-the-Art and Research Challenges ». In : *Environments for Multi-Agent Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 1–47 (cf. p. 17, 22, 26–28, 31, 91).

WEYNS, D., A. OMICINI et J. J. ODELL (2006). « Environment as a first class abstraction in multiagent systems ». In : *Autonomous Agents and Multi-Agent Systems* 14.1, p. 5–30 (cf. p. 18, 19, 44, 91).

WIENER, N. (1948). *Cybernetics: Or Control and Communication in the Animal and the Machine*. New York : J. Wiley & Sons (cf. p. 7, 16).

WILENSKY, U. (1999). *NetLogo*. URL : http://ccl.northwestern.edu/netlogo/ (cf. p. 27, 38, 40, 178, 191).

WILENSKY, U. et W. RAND (2007). « Making Models Match: Replicating an Agent-Based Model ». In : *Journal of Artificial Societies and Social Simulation* 10 (cf. p. 35, 197).

WILSON, M. (2002). « Six views of embodied cognition ». In : *Psychonomic Bulletin & Review* 9.4, p. 625–636 (cf. p. 87).

WOOLDRIDGE, M. J. et N. R. JENNINGS (1995). « Intelligent agents: theory and practice ». In : *The Knowledge Engineering Review* 10.02, p. 115–152 (cf. p. 16, 44, 87).

WOOLDRIDGE, M. J., N. R. JENNINGS et D. KINNY (2000). « The Gaia Methodology for Agent-Oriented Analysis and Design ». In : *Autonomous Agents and Multi-Agent Systems* 3.3, p. 285–312 (cf. p. 34, 49).

WU, L. et al. (2012). « An Agent Formal Model for Autonomous Decision-Making ». In : *2012 4th International Conference on Multimedia Information Networking and Security (MINES)*. Nanjing, China : IEEE, p. 943–946 (cf. p. 70).

ZACHAREWICZ, G. et al. (2010). « A Generalized Discrete Event System (G-DEVS) Flattened Simulation Structure: Application to High-Level Architecture (HLA) Compliant Simulation of Workflow ». In : *SIMULATION* 86.3, p. 181–197 (cf. p. 150).

ZEIGLER, B. P. (1976). *Theory of Modelling and Simulation*. New York : Wiley Interscience (cf. p. 52, 53).

ZEIGLER, B. P., T. G. KIM et H. PRAEHOFER (2000). *Theory of Modeling and Simulation, 2nd Ed.* 2nd. Integrating Discrete Event and Continuous Complex Dynamic Systems. Orlando, FL, USA : Academic Press (cf. p. 10, 12, 52, 53, 59, 60, 63, 69, 80, 91, 123, 125, 132, 141, 149, 199, 205–207, 216).

ZEIGLER, B. P. et al. (2013). « Creating suites of models with system entity structure: global warming example ». In : *DEVS 13: Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*. Society for Computer Simulation International, p. 32 (cf. p. 116).

ZHU, H. (2001). « SLABS: A FORMAL SPECIFICATION LANGUAGE FOR AGENT-BASED SYSTEMS ». In : *International Journal of Software Engineering and Knowledge Engineering* 11.05, p. 529–558 (cf. p. 70).

TABLE DES FIGURES

LISTE DES TABLEAUX

TABLE DES LISTINGS

Résumé

Cette thèse aborde les problématiques liées à la reproductibilité des expériences numériques dans le cadre des systèmes complexes environnementaux, et plus particulièrement dans la cadre de la modélisation de systèmes multi-agents. L'activité de M&S peut s'apparenter à une expérience numérique, au même titre qu'une expérience scientifique menée dans une discipline expérimentale, c'est pourquoi la description des modèles SMA doit être partagée de façon non ambigüe. Dans cet objectif, nous nous sommes dirigés vers une approche formelle, qui permet de décrire les modèles d'un point de vue structurel et sémantique. Nous présentons dans ce manuscrit deux contributions majeures : une proposition de formalisation du paradigme agent associant des méthodes de conception pour l'élaboration des modèles et une mise en œuvre des abstractions permettant de faciliter le développement de SMA, d'après les concepts définis par l'approche formelle à travers l'outil Quartz. Nous proposons également une mise en application de l'approche à travers deux exemples, dans lesquels la définition d'un modèle est établie, de la description informelle à son implémentation.

Mots-clés : modélisation, simulation, systèmes multi-agents, Influence/Réaction, PDEVS, reproductibilité.

Abstract

This thesis addresses questions related to the reproducibility of numerical experiments in the context of complex environmental systems, and more specifically in the context of agent-based modeling. The activity can be seen to a numerical experiment, just like any other scientific experiment in an experimental discipline. Therefore, the description of MAS models should be shared in an unambiguous manner. To this end, we headed to a formal approach to describe models of structural and semantic point of view. We present in this manuscript two major contributions : an agent paradigm formalization proposal combined to design methods for models developments and an implementation of abstractions to facilitate the MAS development, according to the concepts defined by the formal approach through the Quartz tool. We also propose an implementation of the approach through two examples, in which the definition of a model is established from the informal description to its implementation.

Keywords : modeling, simulation, multi-agent systems, Influence/Reaction, PDEVS, reproducibility.