Architecture Agent pour la modélisation et simulation de systèmes complexes multidynamiques : une approche multi-comportementale basée sur le pattern "Agent MVC"

Yassine Gangat

Architecture Agent pour la modélisation et simulation de systèmes complexes multidynamiques : Une approche multicomportementale basée sur le pattern "Agent MVC"

Thèse présentée et soutenue publiquement
le 27 Août 2013

au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Université de Montpellier 2

pour l'obtention du grade de Docteur ès Sciences
de l'Université de La Réunion
par

Yasine (Yassine) **Gangat**

Membres du Jury :

Rémy Courdier, Professeur des Universités, Université de La Réunion, *Directeur de thèse*
Denis Payet, Maître de Conférence, examinateur, Université de La Réunion, *Co-encadrant de thèse*

Alexis Drogoul, Directeur de Recherche, IRD, *Rapporteur*
Olivier Simonin, Maître de Conférence HDR, Université de Lorraine, *Rapporteur*

Sylvie Pesty, Professeur des Universités, Université Pierre Mendès France, *Examinateur*
Joel Quinqueton, Professeur des Universités, Université Montpellier 3, *Examinateur*
François Guerrin, Ingénieur de Recherches HDR, INRA - CIRAD, *Examinateur*
Pascal Degenne, Docteur - Chercheur - Ingénieur, CIRAD, *Examinateur*

سافر تجد عوضا عمن تفارقه * و انصب فان لذيذ العيش في النصب

Voyage, tu y trouveras une compensation à ceux que tu laisses.
Plante la tente, le délice de la vie est le campement.

اني رايت وقوف الماء يفسده * ان سال طاب و ان ما يجر لم يطب

J'ai vu l'eau se corrompre en stagnant,
Alors que lorsqu'elle coule, sa saveur est bonne.

— Mouhammad ibn Idris al-Shafi'i RA

A ma famille…

Remerciements

Ma thèse : ce long voyage où, depuis ses prémices, je fus accompagné d'éclaireurs qui m'ont aidé à gravir les sommets, me guidant tout le long de ma route, jusqu'à son aboutissement...

Mes remerciements vont tout d'abord aux amis et conseillers, qui ont su créer en moi l'étincelle de la recherche me permettant d'éclairer mon chemin : les membres de l'équipe multi-agents. En premier, mon directeur de thèse Rémy Courdier et son assistant Denis Payet, qui m'ont accordé leur confiance en me laissant le choix de ma destination, tout en m'aidant à éviter les obstacles. Ils ont su être disponibles aux moments opportuns, apporter des réponses à mes questions, me remotiver lorsque cela était nécessaire, grâce à leurs qualités scientifiques et humaines. Ensuite, je mentionnerai, par ordre d'ancienneté, Tiana, Nico et Dan, qui m'ont accompagné tout au long de mon périple dans le monde de la recherche et qui ont su me guider dans le labyrinthe de la Simulation Orientée Agent.

Je tiens ensuite à exprimer ma gratitude à mes rapporteurs, Alexis Drogoul et Olivier Simonin, qui ont accepté de se placer au bout de cette odyssée pour se plonger dans mon manuscrit et apporter un regard critique sur mes travaux, ainsi que les autres membres du jury, Sylvie Pesty, Joel Quinqueton, François Guerrin et Pascal Degenne qui ont témoigné de l'intérêt pour mon sujet de thèse en acceptant le rôle d'examinateur.

Je remercie aussi tous les membres de l'université de La Réunion, en particulier Mohammed, Khalid, Fanilo, Olivier, Didier, Véronique, Marie-Annick, Gisèle, Joel, Anne... qui ont tous été un compagnon de route à un moment à un autre, ainsi que toutes les autres personnes extérieures qui ont, d'une manière ou d'une autre, apporté leur contribution à cette traversée.

Je n'oublierai pas bien sûr les personnes qui me sont les plus chères, notamment les membres de ma famille, qui m'ont témoigné d'un soutien indéfectible, d'une confiance absolue et d'un encouragement permanent. Ils ont su être à mes côtés dans les moments les meilleurs comme les plus difficiles durant ce périple.

Et pour finir, je remercie celui, grâce à qui je suis ce que je suis aujourd'hui.

Saint Denis, avril 2013 Y. G.

Mis en page sous LaTeX, adapté à partir du template EPFL EDOC.

Abstract

Co-building and **reuse** of models are at the center of several studies in the field of simulation. However, in the more specific field of Multi-Agent Based Simulation (MABS), there is a lack of methodology to resolve these two issues, despite a strong need by experts.

Model co-building is essential to optimize knowledge sharing amongst different experts, but we often face divergent viewpoints. Existing methodologies for the MABS co-building allow only a low level of collaboration among experts during the initial phase of modeling, and between domain experts with modelers or computer scientists... In order to help this co-building, we propose and follow a methodology to facilitate this collaboration.

Model reuse can provide significant time savings, improve models' quality and offer new knowledge. Some MABS methodologies in this area exist. However, in the spectrum of reuse, they are often limited to a full model's reuse or agent's reuse with the impossibility of reusing smaller parts such as behaviors.

The **EDMMAS** experiment was a concrete case of three successive model reuses. It allowed us to observe new complexity arising from the increase of agents' behaviors. This creates a gap between operational model and conceptual model.

Our goal is to promote the reuse of models, agents and their behaviors.

To answer these questions, we propose in this thesis a new way to codify and integrate knowledge from different disciplines in the model, while using "composable" modules that facilitate reuse. We propose (*i*) a new agent architecture (**aMVC**), applied to a multidynamical approach (**DOM**), with the support (*ii*) of a methodology (**MMC**) based on the decomposition and reuse of behaviors.

Proposals (*i*) and (*ii*) allow us to lead a multidisciplinary MABS project with a large number of actors, helping the co-building of models through the introduction of synergies among the different actors involved in the modeling. They can work independently on their dynamics and the platform will integrate those, ensuring cohesion and robustness of the system. Our contributions include the ability to create the building blocks of the system independently, associate and combine them to form agents. This allows us to compare possibilities for the same dynamic and open the prospect of studying many alternate models of the same complex system, and then analyze at a very fine scale.

Résumé

La **co-construction** et la **réutilisation** de modèles font l'objet de plusieurs travaux dans le domaine de la simulation. Cependant, dans le domaine plus spécifique de la Simulation Orientée Agent (SOA), nous pouvons constater un manque sur ces deux points malgré un besoin fort de la part des thématiciens.

La co-construction est essentielle pour optimiser la mise en commun du savoir de différents experts, mais nous nous heurtons souvent à des divergences de points de vue. Les méthodologies existantes pour la co-construction en SOA ne permettent qu'un faible niveau de collaboration notamment entre thématiciens durant la phase initiale de modélisation, ainsi qu'entre les thématiciens avec les modélisateurs ou les modélisateurs-informaticiens... Pour faciliter cette co-construction, nous proposons de suivre une méthodologie de conception favorisant cette collaboration.

La réutilisation de modèle octroie, avec certaines contreparties, un gain de temps significatif, une amélioration du modèle et l'apport de nouvelles connaissances. Les méthodologies en SOA dans ce domaine existent. Cependant, dans le spectre de réutilisation, elles sont souvent limitées au niveau du modèle complet ou de l'agent avec l'impossibilité de "descendre" plus bas.

L'expérience de **EDMMAS**, cas concret d'un modèle issu de trois réutilisations successives, nous a permis de constater une nouvelle complexité qui découle de la démultiplication des comportements des agents et crée un décalage conséquent entre le modèle opérationnel et le modèle conceptuel.

Notre objectif est de promouvoir la réutilisation aussi bien des modèles, que des agents et de leurs comportements.

Pour répondre à ces questionnements, nous proposons dans ce manuscrit une manière de codifier et d'intégrer la connaissance provenant de disciplines différentes dans le modèle, tout en utilisant des modules "composables" qui facilitent la réutilisation. Nous proposons (*i*) une nouvelle architecture Agent (**aMVC**), appliquée dans un cadre multidynamique (**DOM**), avec l'appui (*ii*) d'une approche méthodologique (**MMC**) basée sur la décomposition et réutilisation des comportements.

Cet ensemble de propositions, (*i*) et (*ii*), permet de conduire un projet pluridisciplinaire de SOA avec un grand nombre d'acteurs, facilitant la co-construction des modèles grâce à l'instauration de nouvelles synergies entre les différents acteurs participant à la modélisation. Les concepteurs pourront travailler de manière autonome sur leur dynamique et la plateforme en fera l'intégration assurant ainsi la cohésion et la robustesse du système. Nos contributions

offrent la capacité de créer les briques élémentaires du système de manière indépendante, de les associer et de les combiner pour former des agents, selon des dynamiques conformément à l'approche DOM. Elles permettent ainsi de comparer la logique selon différentes possibilités pour une même dynamique et d'ouvrir la perspective d'étudier un grand nombre d'alternatives de modélisation d'un même système complexe pour les analyser ensuite à une échelle très fine.

Table des matières

1 Introduction générale

Le savant sans œuvres est semblable à un arc sans corde.

— *Abou Hamid al-Ghazali RA*

1.1 Contexte

Le concept d'agent, dans le **S**ystème **M**ulti-**A**gents (SMA) et la **S**imulation **O**rientée **A**gent (SOA), est devenu depuis les années 1980 [Jennings *et al.*, 1998] un champ de recherches et d'applications en constant développement. La SOA permet de simuler ce qu'on appelle les systèmes dynamiques complexes. Selon [Mikulecky, 1999], la complexité est une propriété du monde réel qui se manifeste par l'impossibilité d'embrasser toutes ses facettes par l'intermédiaire de n'importe quel formalisme.

Ces systèmes se caractérisent par une grande variété d'interactions entre les entités qui les composent. La SOA offre un moyen d'appréhender leurs caractéristiques dynamiques et de représenter leurs fonctionnements, ce qui facilite leur étude. À travers les simulations s'expriment les "représentations informatiques" de ces systèmes et leurs comportements par l'intermédiaire de modèles, dont les données correspondent à des images relativement fidèles des systèmes réels au cours du temps [Courdier *et al.*, 2002].

Dans ces modèles informatiques, les entités, nommées agents, sont directement représentées avec leurs comportements propres et leurs interactions. Ils permettent d'analyser un phénomène comme le résultat d'interactions entre des entités autonomes. La SOA permet de construire de véritables microcosmes artificiels, dont on peut contrôler tous les paramètres (quantitatifs ou qualitatifs) à tous niveaux : de l'entité, du groupe, de la société ou des effets externes sur l'environnement.

Une des applications les plus connues du grand public est le tournage des scènes de combat du film "*Le Seigneur des anneaux*" : le logiciel de SOA MASSIVE (**M**ultiple **A**gent **S**imulation **S**ystem in **V**irtual **E**nvironment)[1], basé sur des algorithmes de logique floue, y a été utilisé. MASSIVE travaille en créant des agents qui possèdent leurs propres caractéristiques

[1] http://www.massivesoftware.com/about.html (Page consultée le 1er février 2013).

aléatoires et ont la capacité de prendre leurs propres décisions dans une situation de foule. Ce logiciel donne un comportement "réaliste" à chacun des centaines de milliers de personnages de synthèse apparaissant dans des scènes de foule ou de bataille. Il confère un réalisme accru à des chevaux de synthèse et autorise des cascades virtuelles irréalisables autrement[2] [Duncan, 2002, Anderson *et al.*, 2003, Woosnam-Savage, 2011].

La démarche de modélisation et de simulation orientée agent est particulièrement appropriée lorsque les systèmes ne sont pas accessibles directement à l'observation ou à la mesure : soit ils ne peuvent être reproduits, soit ils ne peuvent faire l'objet d'expérimentations directes [Courdier *et al.*, 2002].

La SOA constitue une technique de simulation pour valider des hypothèses scientifiques et aider à la prise de décision. Les systèmes étudiés pouvant être naturels ou sociaux : la dynamique d'un édifice volcanique ou celle d'une population, par exemple.

Dans le domaine de la SOA, plusieurs travaux ont été réalisés sur le thème de la modélisation du comportement. La modélisation des aptitudes cognitives de l'agent est un champ vaste, encore en cours d'exploration, qui permet d'obtenir des simulations plus fidèles à la réalité. Elle est utile tant dans le secteur des jeux vidéo (par exemple, pour créer un monde ouvert plus réaliste [Ocio et Brugos, 2009]) que dans des outils d'aide à la décision (quand il s'agit de tester la cohérence de différents scénarios pour les choix d'affectations des terres par exemple). Cependant, encore en plein essor, la modélisation comportementale manque d'expérimentation et de standardisation.

Pour tirer profit du paradigme agent et favoriser son adoption à plus grande échelle, il faut fédérer les compétences de différentes disciplines impliquées. Ainsi nous bénéficierons de l'expérience acquise et proposerons des outils robustes et cohérents qui, nous l'espérons, contribueront à son succès. Un besoin à ce niveau se fait justement sentir dans la modélisation du comportement, domaine dans lequel la collaboration (entre les thématiciens eux-mêmes, avec les modélisateurs...) et la capitalisation de leurs expériences sont importantes. Comme le mentionnent les auteurs de "Software Pattern" [Schmidt *et al.*, 1996], *"Success is more important than novelty"* : le but n'est pas forcément d'innover pour innover, mais d'innover pour faciliter l'utilisation d'une technique débouchant sur une technologie. C'est dans cette ligne de pensée que se situent nos travaux, qui, sur la base d'un travail conceptuel de représentation des connaissances, conduisent à la proposition d'un cadre technique concret.

1.2 L'équipe SMART du LIM à La Réunion

Notre équipe **S**ystème **M**ulti-**A**gents et **R**éseaux de **T**élécommunication (SMART), du **L**aboratoire d'**I**nformatique et de **M**athématiques — EA2525 (LIM)[3] de l'université de La Réunion, travaille selon deux axes de recherche informatique liés : les outils de Simulations Orientées Agent distribués et la technologie de télécommunication s'attachant aux services sur

[2] http://www.lordoftherings.net/effects/prologue_frame.html (Page consultée le 1er février 2013).
[3] http://lim.univ-reunion.fr/

les réseaux. Les projets de recherche de l'équipe portent sur l'étude des systèmes dynamiques complexes au travers des SOA.

De par son contexte, l'île de La Réunion est un terrain d'expérience parfait pour la modélisation de systèmes complexes. Nous l'avons montré au travers de prototypes probatoires d'applications (gestion collaborative de déchets, gestion de l'espace foncier, *etc.*), les approches mises en œuvre par la SOA ouvrent des perspectives très intéressantes pour modéliser et simuler de multiples problèmes collectifs, tout en tenant compte des interactions entre les différents acteurs concernés par la problématique et le cadre institutionnel (services de l'État, collectivités locales, *etc.*). L'élaboration de ces prototypes révèle une grande complexité, notamment due à la grande richesse des comportements portés par les agents.

Dans l'équipe, nous nous intéressons à la problématique d'ingénierie de conception de SOA, en intégrant celle-ci comme réponse à des besoins émanant de partenaires issus de domaines autres que l'informatique (géographie, énergie...). Nos travaux sont structurés selon trois grands axes :

- **Développement d'une plateforme multi-agents** dédiée à la simulation de systèmes dynamiques complexes naturels et sociaux, nommée **GEAMAS N**ew-**G**eneration (GEAMAS-NG), qui succède à **GE**neric **A**rchitecture for **M**ulti**A**gent **S**imulations (GEAMAS), ancienne version de notre plateforme de Simulation Orientée Agent [Marcenac et Giroux, 1998, Soulié *et al.*, 1998]. GEAMAS-NG repose sur une réécriture intégrale du code, fondée sur de nouvelles bases conceptuelles spécifiques au domaine de la SOA , présentées dans [Payet *et al.*, 2006].
- **Réalisation d'expérimentations et développement d'applications** "partenaires". Notre équipe a mené plusieurs projets pluridisciplinaires avec des chercheurs de différentes institutions (universités, IRD[4], CIRAD[5], IFREMER[6], *etc.*). Ainsi BIOMAS [Courdier *et al.*, 2002] est un outil de simulation de gestion collective de déchets organiques, utilisé par le CIRAD. Cette application permet la modélisation et la simulation de situations agricoles réalistes de localités de La Réunion, comme celles de Grand-Ilet ou du Petit-Tampon/Grand-Tampon. Ce prototype, reposant sur la première plateforme GEAMAS, produit des données par des simulations impliquant plus de 200 agents en interaction sur des périodes de plusieurs années, qui sont ensuite exploitées. Plus récemment, nous avons mis en place EDMMAS[7] [Gangat *et al.*, 2009a, Gangat *et al.*, 2009b] basé sur DS[8] [David *et al.*, 2007, David, 2010], outil issu d'un autre projet conduit par le CIRAD (voir Chapitre 3). Il s'agit cette fois d'un prototype construit sur la seconde plateforme GEAMAS-NG . Récemment, un prototype Tortue [Gangat *et al.*, 2010], issu d'une collaboration avec l'IFREMER, a vu le jour.

[4] **I**nstitut de **R**echerche pour le **D**éveloppement
[5] **C**entre de coopération **I**nternationale en **R**echerche **A**gronomique pour le **D**éveloppement
[6] **I**nstitut **F**rançais de **R**echerche pour l'**E**xploitation de la **Mer**
[7] **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation (EDMMAS)
[8] **D**OMINO **S**MAT (DS) : **D**émarche **O**bjet **M**ultisite pour l'étude des **I**nteractions entre **N**iveaux d'**O**rganisation - **S**ystème **M**ulti-**A**gents **T**erritoires

- **Proposition de démarches méthodologiques** en ingénierie de conception de SMA pour la simulation. Notre plateforme de base a servi à de nombreuses expérimentations et donné lieu à de multiples enrichissements et validations, intégrés ensuite dans ses nouvelles versions. Les travaux de recherche de l'équipe SMART se sont focalisés sur trois axes : l'observation dans la SOA [Ralambondrainy *et al.*, 2006, Ralambondrainy, 2009], la distribution et la parallélisation de SOA [Sébastien *et al.*, 2008, Sébastien, 2009], la gestion de l'émergence dans la SOA [David *et al.*, 2009, David, 2010, David *et al.*, 2012].

La recherche effectuée sur ces trois points nous a permis de soulever une question transversale, définissant un nouvel axe : la modélisation du comportement des agents dans la SOA. Plus cette modélisation est efficace et pertinente, plus les simulations sont fidèles à la réalité.

1.3 Problématique et propositions

1.3.1 Problématique

Notre équipe, à travers la recherche et son application, propose des modélisations et des logiciels d'aide à la décision sur des problèmes délicats à gérer. Nous avons ainsi remarqué un manque de méthodologie dans la **co-construction** (la collaboration pour construire un modèle), spécifiquement dans la sphère de la modélisation pour la simulation. La co-construction est essentielle pour optimiser la mise en commun du savoir de différents experts, mais nous nous heurtons souvent à des divergences de points de vue. Sur un même agent, un expert A peut avoir une vision différente de l'expert B (par exemple, si A est expert en gestion de l'énergie et B en évolution de la population). De cet écart naît la possibilité d'une incompréhension, alors que les deux expertises sont pourtant nécessaires pour la résolution d'un même problème. Cette divergence apparaît parfois dans le vocabulaire, dans l'échelle adoptée pour la gestion de l'agent, au niveau de l'ensemble des données utilisables...

Pour la modélisation des objets dans les domaines du bâtiment ou de l'aéronautique, la littérature (voir Chapitre 2) propose plusieurs démarches pour résoudre ce problème. Il en est de même pour la modélisation dans les simulations en général, mais pour la SOA, les méthodologies et architectures existantes n'ont pas été conçues dans cette optique.

D'autre part, la majorité des projets engagés par des groupes pluridisciplinaires se terminent par des prototypes très riches en couches de connaissances. Malheureusement au terme de leur utilisation, ces richesses finissent souvent par être délaissées. L'effort investi ne génère que peu de bénéfices après la fin de vie des projets. Or un gain de temps significatif pourrait être obtenu si au moins une partie des modèles, par exemple selon un certain point de vue, pouvait être plus aisément exploitée dans d'autres travaux. La **réutilisation** de modèle, favorisant l'isolation des connaissances par champs de compétences, est donc une demande des acteurs des projets.

Un des apports de notre équipe sur ce point a été la **M**odélisation **O**rientée **D**ynamique (DOM)[9] (voir Section 3.2). Elle propose de se focaliser sur les dynamiques du système. Elle

[9] De l'anglais **D**ynamic **O**riented **M**odeling.

découpe un système complexe en sous-systèmes sur la base d'un ensemble de dynamiques pré-identifiées. Cependant, nous verrons dans ce manuscrit que DOM n'est pas suffisant pour répondre pleinement à ce problème de réutilisation de modèles.

Notre objectif est de faire des propositions au niveau de ces deux problématiques, avec au centre de nos travaux, la gestion du multicomportement. Ainsi, les deux axes que nous avons décidé d'explorer sont les suivants :

1. **La co-construction de modèles** ou comment faciliter la conception de modèles dans un projet pluridisciplinaire ? Comment améliorer la collaboration entre thématiciens (experts) pour la construction d'une SOA ?
2. **La réutilisation de modèles** ou comment faciliter la conception de modèles pour leur réemploi ? Comment faire pour que les prototypes, souvent très riches en connaissances, survivent à la fin des projets pour lesquels ils ont été initialement conçus ?

1.3.2 Organisation des travaux, propositions et contributions

Pour comprendre les mécanismes de modélisation de la connaissance dynamique dans les systèmes complexes, nous avons utilisé le modèle EDMMAS comme support sur la réflexion (voir Chapitre 3) et nous nous sommes plongé dans la bibliographie de la co-construction des modèles en SOA et de leurs réutilisations.

Le présent document est organisé en huit chapitres :

1. Le premier chapitre, cette introduction, présente le contexte, la problématique et les objectifs de nos travaux.
2. Le second chapitre développe l'état de l'art sur la co-construction des modèles et leurs réutilisations ainsi que sur les méthodologies de conceptions de simulations. Nous mettons en avant les limites de la modélisation actuelle dans le domaine de la SOA, en particulier au niveau de nos deux axes de recherches. Les solutions pour résoudre ce problème de fond nous amènent à repenser certains concepts de base des agents et à une méthodologie de conception adaptée.
3. Le troisième chapitre présente la première contribution : le modèle EDMMAS, fruit d'un projet pluridisciplinaire, où nous avons expérimenté le passage à l'échelle de DOM ainsi que ses limites, qui s'est instancié sous forme d'un prototype. Il s'agit d'un cas concret d'une démarche pour la réutilisation de modèle.
4. Le chapitre 4 propose notre seconde contribution : le cadre de travail théorique dans lequel nous nous plaçons pour l'utilisation des Design Patterns, briques de base pour la construction et la formalisation d'un agent de type **a**gent **MVC** (aMVC). Ce modèle est inspiré des Design Patterns du monde objet, en particulier du **M**odèle **V**ue **C**ontrôleur (MVC) qui, malgré sa simplicité apparente, a facilité notamment la réutilisation des modules des applications multivues, multifenêtres... Nous avons transposé MVC aux besoins du monde agent tel que nous l'abordons.

5. Le chapitre 5 offre un cadre de validation de ce modèle en l'appliquant à un cadre opérationnel : le multidynamique. Nous intégrons d'abord le concept aMVC à celui de DOM, en élaborant une agrégation des briques élémentaires de notre agent. Puis nous proposons une discussion sur cet agent aMVC ainsi que les caractéristiques d'une plateforme adaptée pour la bonne gestion de ces agents.
6. Le chapitre 6 expose notre troisième contribution : un cadre **méthodologique** pour la conception d'un modèle co-construit et réutilisable. Cette approche méthodologique de **M**odélisation **M**ulti**C**omportementale (MMC) se déroule en six phases qui permettent de découper précisément, avec des étapes identifiées, la modélisation d'un système complexe. Chaque étape est définie par un ensemble de descriptions élémentaires, pouvant être représentées sous forme de tableaux, incluant les notions d'activités, de comportements, de dynamiques, *etc.* Il s'agit en effet de représenter les différents éléments acteurs du système complexe de la manière la plus simple pour construire la simulation à partir de ces briques élémentaires.
7. Cette approche méthodologique MMC a été mise en œuvre au travers de la réalisation d'un prototype probatoire dans le chapitre 7. Ce qui montre ainsi l'automatisation du processus et les possibilités d'inférences sur ses éléments.
8. Le dernier chapitre, notre conclusion, présente nos perspectives de recherches.

Ce travail a pris son essence dans la proposition d'une extension d'une SOA existante qui a révélé les limites de co-construction de SMA et ceci même en considérant l'utilisation d'une démarche de modélisation dynamique conçue à cet effet. Nous avons alors identifié la faiblesse des propositions existantes ce qui nous a incité à proposer un nouveau cadre de conception de l'agent lui-même. Ce nouveau cadre de structuration d'un agent qui s'articule autour du Design Pattern **agent MVC** (aMVC) est au cœur de notre travail de thèse. Cette proposition a permis de répondre aux limites de la **M**odélisation **O**rientée **D**ynamique (DOM) et conduit à l'approche méthodologique de **M**odélisation **M**ulti**C**omportementale (MMC).

2 État de l'art sur les modèles : Leur méthodologie de conception, leur co-construction pluridisciplinaire et leur réutilisation

Plan du chapitre

2.1 Introduction

Notre préambule a montré que le problème de la co-construction pluridisciplinaire de modèle n'est pas récent. Cependant, les premières recherches concernent plus le domaine de l'ingénierie en général que celui de la simulation. L'**O**ptimisation **M**ulti**D**isciplinaire (OMD ou MDO en anglais [1]) utilise des méthodes d'optimisation pour résoudre des problèmes de conception mettant en œuvre plusieurs disciplines [Braun et Kroo, 1997]. Ces techniques sont utilisées dans plusieurs domaines : conceptions aéronautique [Berends et van Tooren, 2007], automobile, navale, électronique et informatique.

De même, la réutilisation de modèles dans le domaine de la simulation [Robinson *et al.*, 2004, Balci *et al.*, 2011] demeure une préoccupation au sein de la communauté scientifique depuis 1986 [Sargent, 1986].

Ce chapitre, après une courte introduction sur les méthodologies de conception de simulations, abordera ces deux objectifs : la co-construction et la réutilisation de modèles.

2.2 Méthodologie de conception de simulations

Lorsqu'on comprend ce qu'est la **S**imulation **O**rientée **A**gent (SOA), avec ses avantages et ses inconvénients, la première question qui se pose est : "Comment aboutir à la SOA à partir du monde réel ?". Il n'existe pas de réponse unique, mais la constante présente dans toutes les réponses est la nécessité d'une méthodologie.

Définition 1

Méthodologie de conception de simulations : Ensemble ordonné d'étapes par lesquelles il convient de passer pour obtenir une simulation du système considéré. Ces étapes peuvent être reconduites pour être affinées, au moyen de cycles ou d'itérations. Nous parlons aussi de **processus de simulation**.

Drogoul [Drogoul *et al.*, 2002, Drogoul *et al.*, 2003] et Ramat [Ramat, 2006] expliquent qu'un processus de simulation peut être décomposé de manière plus ou moins fine, selon la granularité de la description voulue par les auteurs, dans le contexte de leurs travaux. Ainsi, pour situer les différents points où nous apportons notre contribution, définissons certaines notions de base en nous appuyant sur les travaux existants [Minsky, 1965, Drogoul *et al.*, 2002, Robinson, 2006, Sargent, 1999, Ralambondrainy, 2009, Galán *et al.*, 2009].

2.2.1 Définitions

Les définitions détaillées dans cette section s'inscrivent comme bases pour la méthodologie de conception de SOA proposée dans le Chapitre 6. La première définition, classique, est celle du modèle général [Minsky, 1965] :

[1] **M**ultidisciplinary **D**esign **O**ptimization

Définition 2

Modèle : Pour un observateur B, un objet A* est un modèle d'un objet A dans la mesure où B peut utiliser A* pour répondre aux questions qui l'intéressent à propos de A.

Cette définition étant générique, nous pouvons distinguer les différents types de modèles suivant l'avancée de cette conception, en nous basant sur la granularité de nos travaux.

Définition 3

Modèle de domaine : Modèle réalisé à partir du monde réel, contenant les micro-connaissances[2] des spécialistes de la partie du monde réel considérée et appelé également "modèle non-formel" par Galán.

Il s'agit ici du travail principal des thématiciens : le modèle de domaine représente le système complexe, d'après leurs connaissances. Ce modèle est défini par un langage naturel et spécifique au domaine considéré, souvent peu formel, parfois ambigu, afin de définir l'objectif de la simulation. Par exemple, un thématicien voudra représenter "la migration d'une personne vers une autre ville en cas de surpopulation".

Définition 4

Modèle de conception ou modèle conceptuel : Modèle réalisé à partir de celui de domaine que le modélisateur traduit en un modèle formel, sur les bases du paradigme-cible (dans notre cas, le paradigme agent). Ce modèle est appelé "modèle formel" par Galán.

Ici commence le travail des modélisateurs : ils élaborent un modèle décrit au travers d'un support de formalisation, à partir du précédent. Dans ce but, ils clarifient les abstractions proposées par les thématiciens et mentionnent les propriétés en accord avec les concepts agents (indépendamment de l'implémentation qui pourrait être faite par la suite). Par exemple, le modélisateur exprimera : "Si la ville est surpeuplée (*c'est-à-dire* que la population dépasse un seuil X), alors il faut que l'agent se déplace vers une autre ville".

Selon Drogul et Galán, le rôle de modélisateur est très important, car :

[2] Selon Drogoul, les experts manipulent deux niveaux de connaissances :

- les micro-connaissances : ensemble de connaissances locales sur les "individus" sans lesquels le système-cible n'existerait pas, composées à la fois d'observations et d'hypothèses.
- les macro-connaissances : ensemble de connaissances "globales" sur le système-cible, souvent tirées d'observations.

- Il fait office de médiateur entre thématiciens et modélisateurs-informaticiens[3].
- Il opère des choix pour adapter le modèle de domaine : celui du modélisateur est moins générique que celui du thématicien.
- Il doit transformer le modèle non-consistant du thématicien (exprimé en langage naturel) en un modèle formel.

Définition 5

Modèle opérationnel : Modèle réalisé par le modélisateur-informaticien, à partir du modèle de conception. Il le définit, en utilisant une sémantique adaptée à la technique d'implémentation choisie. Il est aussi appelé "modèle exécutable" par Galán ou "modèle de simulation" par Sargent.

Selon Drogoul, dans la plupart des projets existants, ce modèle est souvent négligé au profit d'une implémentation directe, *c'est-à-dire* en passant directement au modèle informatique. Dans le cadre de la SOA, il s'agit pour les modélisateurs-informaticiens d'exprimer les agents en leur donnant une existence dans le modèle global, au travers de propriétés techniques, en ajoutant ce qui est nécessaire à l'implémentation. Cette opération permet ainsi aux thématiciens et modélisateurs de comprendre, voire de modifier ce qui va être implémenté. Le rôle de ces modélisateurs-informaticiens consiste aussi à tenter de simplifier le modèle des modélisateurs pour ne pas (trop ?) affecter les résultats, en se basant sur les limites de l'implémentation. L'exemple précédent se traduirait par : $population(ville_{EnCours}) > X \Rightarrow migration(agent, ville_{Destination})$.

Définition 6

Modèle informatique : Modèle réalisé à partir du modèle opérationnel. Il s'agit de l'implémentation informatique du modèle, appelé également "système computationnel" par Drogoul ou "programme informatique" Galán.

Galán se distingue de Drogoul sur ce point en exprimant un quatrième rôle : celui des programmeurs. Ils n'ont pas besoin de simplifier ou d'approximer le modèle précédent. Ils jouent une fonction différente de celle des modélisateurs-informaticiens : ils s'occupent du modèle informatique qui sera exécutable. Ce modèle est obtenu à partir du modèle opérationnel précédent, qui contient les concepts nécessaires à son exécution, mais qui, lui, n'est pas implémenté. Le modèle informatique s'appuie sur un langage de programmation ou une plateforme de simulation particulière (à l'inverse du modèle opérationnel qui en est indépendant).

[3] Galán propose dans sa méthodologie [Galán *et al.*, 2009] le rôle de "*Computer Scientist*". Ce terme n'a pas d'équivalent qui exprime le même sens en français. Il sera traduit par modélisateur-informaticien.

Parfois, le modélisateur étant lui-même informaticien et programmeur, le modèle informatique est réalisé directement à partir du modèle de domaine ou de conception.

Définition 7

Modèle simulé : Instance du modèle informatique initialisé avec des paramètres définis et qui évolue au cours d'un temps virtuel. Il s'agit d'un modèle créé à partir d'un modèle informatique qui a été initialisé. Il s'agit donc d'un exemplaire ou instance d'un modèle informatique.

Dans le cadre de la SOA, pour le modèle simulé, les entités sont instanciées : les agents sont créés et leurs structures de données sont initialisées, de même pour l'environnement (espace où les agents évoluent).

Cette suite de définitions nous aboutit à celle-ci :

Définition 8

Simulation : Évolution du modèle simulé suivant un temps virtuel, ce temps pouvant être explicite ou non.

2.2.2 Exemples de méthodologies de conception de simulations

Ces définitions aident à mieux comprendre les différentes méthodologies de conception de simulations rencontrées. Nous allons en citer quelques-unes brièvement, mais l'étude ou la comparaison des méthodologies existantes n'est pas notre but. Un processus de simulation peut être décomposé de manière plus ou moins fine ; ainsi le nombre d'étapes du processus varie en fonction des auteurs.

Méthodologie selon Ralambondrainy

Ralambondrainy [Ralambondrainy, 2009] décrit une méthodologie de conception de simulations, reprise par Sébastien [Sébastien, 2009], en accord avec les étapes représentées sur la Figure 2.1. Par rapport aux définitions précédentes, selon eux, la "conception" correspond au passage du modèle de domaine vers le modèle informatique directement, les modèles de conception et opérationnel étant sous-entendus car non utilisés. Cette conception est réalisée de manière transparente à partir du modèle de domaine. Ces étapes forment un cycle par des "retours" issus de l'analyse et de l'interprétation des résultats. Ce cycle permet l'affinement et l'amélioration des modèles à tous les niveaux.

Dans ce processus, l'accent est mis sur l'observation de la simulation, sur sa distribution et sa parallélisation potentielle. Cependant, cette méthodologie ne comporte que deux étapes de conception de modèles, ce qui ne permet pas de détailler les rôles présentés précédemment dans les définitions des différents modèles.

Figure

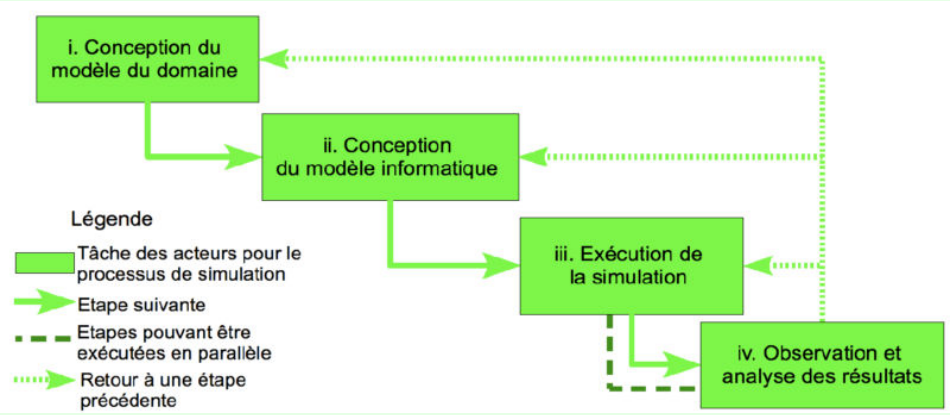


FIGURE 2.1 – Méthodologie de conception de simulations selon Ralambondrainy

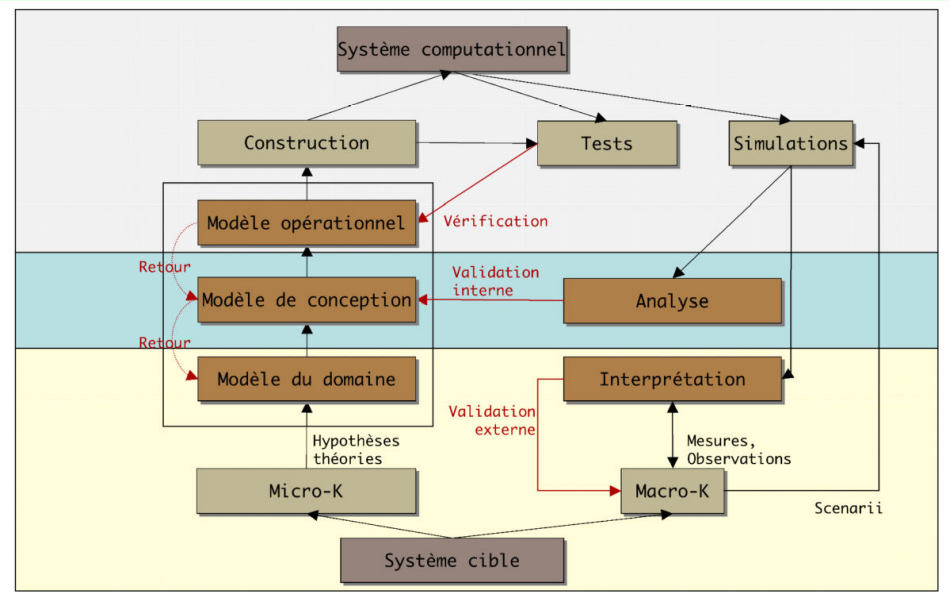


FIGURE 2.2 – Méthodologie de conception de simulations selon Drogoul *et al.*

Méthodologie selon Drogoul *et al.*

Drogoul, Vanbergue et Meurisse présentent une méthodologie de conception de simulations plus affinée et plus complète que la plupart des autres méthodologies [Drogoul *et al.*, 2002, Drogoul *et al.*, 2003].

Ils proposent une décomposition exhaustive du processus en mentionnant des étapes cycliques grâce aux "retours" et à l'utilisation des micro et macroconnaissances (voir Figure 2.2).

Ils mettent aussi l'accent sur les rôles des thématiciens, modélisateurs et modélisateurs-informaticiens, établissant ainsi un découpage du travail entre les intervenants. Ces auteurs présentent aussi toutes les étapes postérieures à la simulation : vérification, validation, analyse et interprétation.

Le niveau de détails de cette méthodologie permet donc une séparation des rôles claire entre les différents acteurs. Souvent, dans un projet réel, plusieurs de ces rôles sont joués par le même individu, ce qui rend parfois la distinction difficile. Les "retours" ne favorisent qu'un certain niveau de collaboration entre deux groupes d'acteurs.

Méthodologie selon Galán *et al.*

Figure

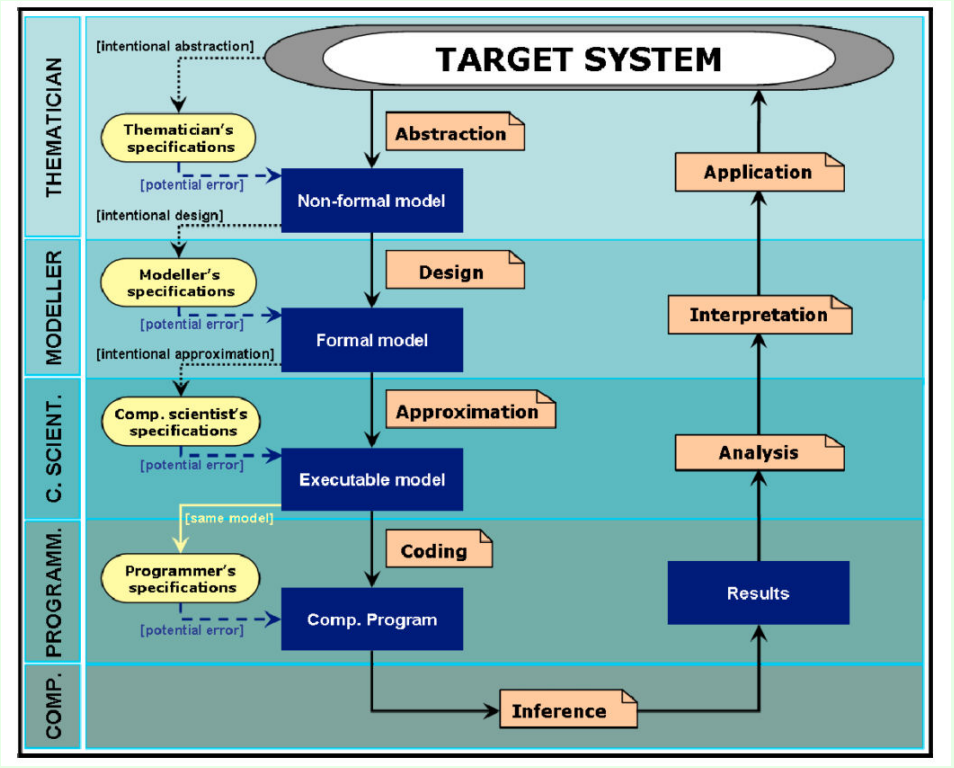


FIGURE 2.3 – Méthodologie de conception de simulations selon Galán *et al.*

Galán *et al.*[4] dans un article [Galán *et al.*, 2009] se basent sur les travaux de modélisations existants, notamment ceux de Drogoul, et y ajoutent le rôle de programmeur, repris dans les définitions de la Section 2.2.1.

[4] Galán, L. R. Izquierdo, S. S. Izquierdo, Santos, del Olmo, López-Paredes et Edmonds

Dans ce processus, les auteurs mettent en évidence la dégradation progressive des modèles due aux abstractions, approximations... à chacun des niveaux et les erreurs potentielles qui en découlent.

Méthodologie selon David

Dans sa thèse de doctorat [David, 2010], David propose une méthodologie où la métaconnaissance[5] occupe une position importante, au centre de la démarche, pour transformer des connaissances potentielles en nouvelles connaissances et les réinjecter dans le système si nécessaire.

Selon cette approche, l'auteur définit l'émergence comme une métaconnaissance et montre ses différents niveaux de détection, d'identification, de caractérisation et de prise en compte potentiels dans le système.

Figure

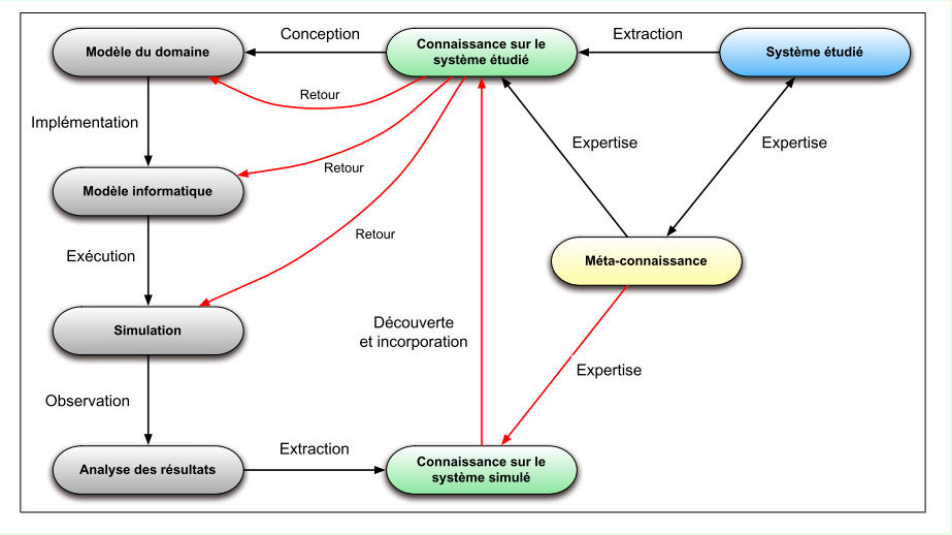


FIGURE 2.4 – Méthodologie de conception de simulations selon David

2.2.3 Synthèse

Ces quelques exemples montrent bien l'absence d'unanimité autour de l'explication détaillée du "comment ?". Selon le contexte, chaque auteur décompose de manière plus ou moins détaillée sa méthodologie de conception de simulations, en y ajoutant des étapes ou en combinant d'autres. De plus, ces méthodologies peuvent être une succession non-linéaire

[5] Une métaconnaissance est une connaissance sur la connaissance. Elle permet de découvrir, décrire, utiliser et manipuler des connaissances.

d'étapes. De nombreux autres travaux présentent des méthodologies, nous n'en avons cité que quelques-unes, chacune d'entre elles comportant ses spécificités dans son contexte.

Les travaux mentionnés ici sont ceux d'auteurs évoluant dans le domaine agent. Nous nous inspirerons de ces méthodologies pour en proposer une autre plus spécifique. Elle nous permettra de positionner nos travaux, centrés sur@ la notion de comportement, dans une démarche plus collaborative.

2.3 Problématique de la co-construction de modèles

2.3.1 La co-construction en général

Cette problématique est devenue un sujet de recherche dans les années 1990 [Wong et Sriram, 1993] après l'essor de la conception assistée par ordinateur (CAO) et les considérations sur l'économie et l'optimisation du temps et des ressources.

Dans le domaine de la conception informatique

La création de modèles d'objets, surtout dans le domaine de l'ingénierie des bâtiments [Rosenman et Gero, 1998, Gül et Maher, 2006], nécessite l'expertise d'un groupe de thématiciens, chacun expert dans son domaine. Des modélisateurs de différents horizons doivent collaborer avec d'autres chercheurs, des analystes, des hommes de terrain, *etc.* Pour ce faire, ces projets ont tiré profit des technologies de l'Ingénierie Assistée par Ordinateur (IAO)[6].

Il est important que chaque thématicien ou modélisateur possède sa (ses) propre(s) vue(s). Ainsi, dans le domaine de la construction, architectes, ingénieurs des structures et designers n'ont pas la même vision d'un mur, d'une colonne ou d'un escalier. Selon le point de vue, ces "objets" sont à positionner parmi d'autres pour des raisons d'esthétique, de soutènement, ou encore cet objet en lui-même est important au niveau du matériau utilisé... Les "objets" sont les mêmes ; la vision est différente.

C'est ce que Bucciarelli [Bucciarelli, 1994, Bucciarelli, 2003] explique à propos du processus de conception en ingénierie. Il pose le postulat suivant : le "Engineering Design" (conception technique) est un processus social nécessitant la participation de personnes différentes, ayant des compétences, des responsabilités et des intérêts techniques divers et variés. Chacun des participants voit l'objet de la conception différemment, en accord avec le cœur paradigmatique de sa discipline et selon son poste de responsabilité.

Il justifie ainsi cette vision différente : chaque personne responsable d'un sous-système ou d'une sous-fonction du design travaille à l'intérieur d'un domaine technique particulier (en l'occurrence le sien). Sa conception implique forcément des manières de modéliser ou de penser inhérentes à ce même domaine mais différentes de celles des autres participants. Sa conclusion : "*Il n'y a qu'un objet de conçu, mais plusieurs mondes pour ce même objet*", comme

[6] Appelé aussi Ingénierie Numérique, ce terme regroupe l'ensemble des moyens numériques utilisés par les ingénieurs et techniciens pour concevoir, simuler et valider de nouveaux produits ou concepts industriels. Parmi ces moyens, citons les logiciels de CAO (Conception Assistée par Ordinateur), de simulation, de FAO (Fabrication Assistée par Ordinateur), les SGBC (Système de Gestion de Bases de Connaissance)...

Figure

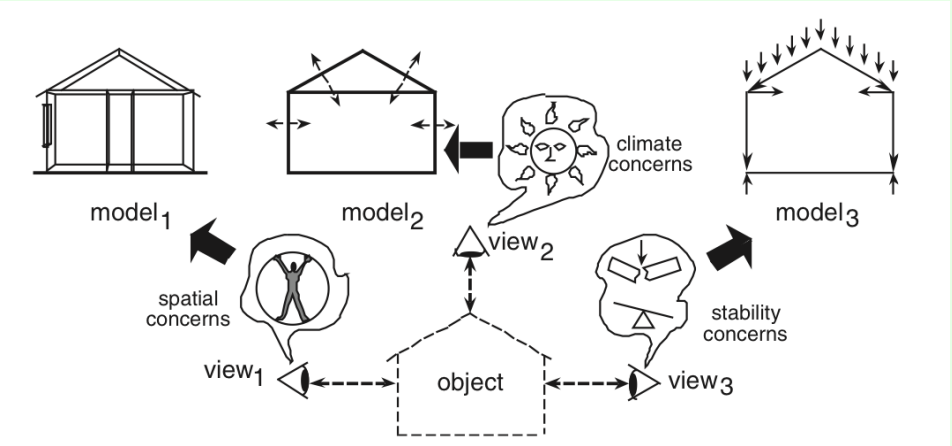


FIGURE 2.5 – Vues multiples d'un même objet selon [Rosenman *et al.*, 2007]

le montre la Figure 2.5. Nul ne peut avoir la compréhension totale du modèle : nul n'en a le "*god's eye view*"[7].

Conscient de ce problème, William H. Newell, professeur de Recherche Interdisciplinaire[8], a publié un article [Newell, 2001b, Newell, 2001a], repris ensuite par Rick Szostak [Szostak, 2002], professeur en économie, qui explique comment faciliter la co-construction pluridisciplinaire. Il propose une méthodologie en douze points, qui commence par "le questionnement initial" et se termine par "la communication des résultats", en passant par des étapes telles que "l'analyse de la littérature" ou "l'étude des phénomènes et sous-phénomènes".

La collaboration étroite des participants accélère le processus de modélisation, ce qui réduit ainsi les coûts et augmente la qualité du modèle. Un autre point important : la création de modèles n'est pas figée, un modèle évolue et peut/doit s'affiner au fur et à mesure de l'avancement du projet, des tests, des équilibrages...

Dans le domaine de la modélisation

La modélisation des systèmes complexes ressemble, par certains points, à la création de modèles d'objets en général (voir la Section 2.1) : elle demande l'expertise d'un groupe de thématiciens, chacun spécialiste dans son domaine. Des modélisateurs de différents horizons

[7] "*God's eye view*" est une expression anglaise pour désigner un point de vue où l'interlocuteur assume qu'il a une connaissance que seul Dieu pourrait avoir.

[8] "Interdisciplinary Research" ou Recherche Interdisciplinaire. Il s'agit d'un domaine en plein essor, en grande partie en raison de sa capacité à résoudre les problèmes complexes qui s'étendent au-delà d'un point de vue disciplinaire unique et qui exigent une réflexion et une recherche interdisciplinaire pour leur résolution. Pour reprendre le proverbe japonais utilisé par le Centre de Recherche Interdisciplinaire de Paris : "*None of us is as smart as all of us*".

collaborent avec d'autres chercheurs, des analystes, des hommes de terrain...

De la même manière que précédemment, dans une modélisation d'un système complexe, aucun modélisateur ne possède le "*god's eye view*" sur le modèle. Chacun n'aura qu'une "vision partielle" de la compréhension du modèle.

Les auteurs de l'article [Nikolic *et al.*, 2007] énoncent les trois principaux challenges de la recherche sur les systèmes complexes :

1. Comment codifier et intégrer la connaissance provenant de disciplines différentes ?
2. Comment avoir une compréhension réaliste du comportement d'un système complexe ?
3. Comment traduire cette connaissance et cette compréhension précitées dans le modèle pour obtenir un comportement du système proche de la réalité ?

2.3.2 La co-construction et la Simulation Orientée Agent

La collaboration lors de la conception de modèle est un défi important si on reste sur le plan général. Dans notre proposition, nous restreignons ce champ à la co-construction dans la SOA.

Les auteurs de [Borshchev et Filippov, 2004] expliquent que le type d'approche des SOA est plus général et puissant que les autres simulations. Ce constat provient de la comparaison faite entre simulations par les multi-agents, la dynamique des systèmes, les systèmes dynamiques[9] et les événements discrets[10] dans AnyLogic[11]. La SOA permet, en effet, de capturer des structures plus complexes et, de surcroit, dynamiques.

Autre avantage précieux, elle prévoit la construction de modèles malgré l'absence de connaissance sur les interdépendances globales. Vous n'avez que très peu (ou pas) d'éléments sur la façon dont les choses s'influencent mutuellement au niveau global ou sur la séquence globale des opérations, mais si vous avez quelque perception sur la façon dont les participants individuels du processus se comportent, vous pouvez construire le modèle centré sur les agents puis obtenir le comportement global.

Ainsi la co-construction de modèles dans la SOA devrait non seulement être présente, mais aussi avoir un impact non négligeable sur la méthodologie débouchant sur la conception des modèles.

Cet aspect collaboratif au sein de ce domaine en est encore au stade de développement. Selon les auteurs de [Sinha *et al.*, 2001], il nécessite :

[9] Un Système Dynamique (DS) est un système classique qui évolue de façon causale et déterministe alors que la Dynamique des Systèmes (SD) est une approche prenant en compte les boucles de rétroaction internes et les effets retard qui affectent le comportement global du système.

[10] Une simulation à événements discrets (DE) est une modélisation informatique représentée par une séquence chronologique d'événements discrets.

[11] AnyLogic est un outil de simulation multiparadigme développé par XJ Technologies qui s'appuie sur les événements discrets ou centrés sur des processus (DE), Dynamique de systèmes (SD) et Systèmes Multi-Agents (SMA). Ce logiciel commercial utilise un langage de modélisation graphique et permet l'extension du modèle en Java.

- Une représentation commune du modèle (par exemple, HLA[12]) pour permettre aux différents acteurs de communiquer entre eux [Sriram *et al.*, 1992, Wong et Sriram, 1993, Sierhuis et Selvin, 1996]. Une autre approche consiste en des représentations différentes sur des domaines reliés par des relations bidirectionnelles (qui permettent la consistance du modèle) comme le proposent [Rosenman et Gero, 1996, Maher *et al.*, 2006, Rosenman *et al.*, 2007, Maher *et al.*, 2008], avec des concepts comme "correspondTo", "a type of", "an instance of", "owner"...
- Une ontologie pour la modélisation et la simulation qui facilitera l'échange d'informations au sein du projet (entre les acteurs ou même entre les différents sous-modèles) et aussi pour la réutilisation de modèles [Devedžić, 1999, Ozawa *et al.*, 2000].
- Une manière de gérer les modèles et/ou les sous-modèles (stockage, référencement, réutilisabilité...) comme celle proposée par [Breunese *et al.*, 1998].
- Une abstraction du modèle, nécessaire dans certains cas [Robinson *et al.*, 2004], qui fournira des vues différentes selon le modélisateur.

Dans le domaine plus spécifique de la SOA, les recherches propres à cette problématique ne sont pas abondantes, car les modélisateurs sont le plus souvent en collaboration étroite pour la création des modèles. Dans des SOA de petite envergure ne nécessitant que très peu d'acteurs, le modélisateur-informaticien devient lui-même le modélisateur et l'interlocuteur des différents experts. La question reste en suspens pour les projets de grande envergure, où souvent c'est le modélisateur qui a pour "devoir" de réunir, comprendre et modéliser les "visions" des thématiciens.

Notre principale contrainte de co-construction est de faciliter la collaboration pour l'élaboration des modèles au niveau même des agents. Cette collaboration peut se retrouver sous deux angles :

- entre les experts de différents domaines,
- entre les groupes d'experts, de modélisateurs, de modélisateurs-informaticiens...

Parmi les approches les plus proches de la nôtre, nous pouvons notamment signaler les propositions suivantes.

PAMS

PAMS[13] (**P**lateforme d'**A**ide à la **M**odélisation-**S**imulation) [Nguyen *et al.*, 2008, Nguyen *et al.*, 2009, Gaudou *et al.*, 2009, Nguyen *et al.*, 2011, Gaudou *et al.*, 2011] est un environnement collaboratif conçu et développé par l'UMMISCO[14], en collaboration avec l'unité de recherche

[12] High Level Architecture ou l'Architecture de Haut Niveau [Dahmann et Morse, 1998] (HLA) est une spécification d'architecture logicielle qui définit comment créer une simulation globale composée de simulations distribuées interagissant sans être recodées. Elle a notamment été utilisée par le département de la Défense des États-Unis pour la simulation de scénarios de combat.

[13] Voir http://www.pamsproject.org/

[14] Unité mixte internationale : **U**nité de **M**odélisation **M**athématique et **I**nformatique de **S**ystèmes **CO**mplexes.

GEODES (IRD, France) et le laboratoire MSI (IFI, VietNam).

Ce projet, démarré en 2007, regroupe un ensemble d'outils conceptuels et logiciels pour modéliser et simuler des systèmes complexes. Cet environnement associe des outils conventionnels de collaboration distribuée entre plusieurs chercheurs de différentes zones géographiques avec une structure logicielle spécifique permettant la manipulation de simulateurs, d'expérimentations et de résultats, par une interface Web.

Ces outils sont synchrones (vidéoconférences, tableau blanc…) ou asynchrones (courriel, forums, wikis…). La structure incorpore plusieurs applications dont :

- Un framework de type "groupware" modulaire (Sakai et Tomcat) pour la collaboration en ligne.
- Un serveur d'application (Jonas : Java Open Application Server) pour exécuter les simulateurs, gérer les expérimentations…
- Des logiciels de simulations (Netlogo, Repast et GAMA).
- Une base de données (MySql) pour enregistrer les résultats des simulations.

Figure

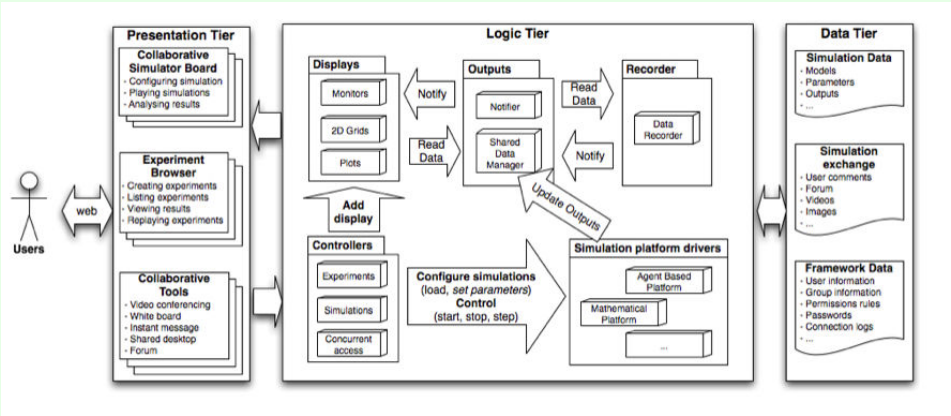


FIGURE 2.6 – Structure de PAMS

Cette plateforme adopte une architecture **M**odèle **V**ue **C**ontrôleur (MVC), au centre de laquelle sont regroupés cinq composants (voir Figure 2.6) :

- **Le Simulation platform drivers package** permet d'établir le lien avec les simulateurs. Il lance les modèles de simulation et renvoie les résultats au **Output package** (Sortie). L'architecture de PAMS permet d'adapter facilement une nouvelle plateforme simulateur en ajoutant un adaptateur.

- **Le Output package** manipule les données obtenues en les retransmettant aux modules **Displays package** ou **Recorder package**.
- **Le Displays package** permet de formater l'affichage des données.
- **Le Recorder package** enregistre les données dans la base de données.
- **Le Controllers package** a pour objectif de donner le contrôle de la simulation aux participants et de garder la cohérence entre les objets partagés par plusieurs utilisateurs.

Sur cette plateforme, des rôles différents ont été implantés : chaque utilisateur ne peut voir et modifier qu'un certain ensemble d'informations suivant son rôle. Un nouveau protocole CoODD (**C**ollaborative **ODD** protocol), basé sur ODD (**O**verview, **D**esign concepts, and **D**etails), actuellement utilisé avec EtherPad[15], est en cours d'implémentation pour faciliter la représentation commune, nécessaire lors d'une collaboration. D'autres adaptateurs vers de nouvelles plateformes de simulation sont aussi à l'étude, comme Madkit.

PAMS est une sorte d'application composite (*Business mashup*) avec de nombreux atouts, surtout grâce aux outils de collaborations synchrones et asynchrones (vidéoconférence, tableau blanc, forums...). Actuellement, PAMS offre ces outils aux modèles préenregistrés et paramétrables. Cependant comme l'expliquent les auteurs de [Nguyen *et al.*, 2009], elle ne permet pas d'éditer les modèles présents ni d'en créer au travers de la plateforme.

JDEVS

[Filippi, 2003, Filippi et Bisgambiglia, 2004] proposent un framework ainsi qu'une implémentation (JDEVS) permettant l'ajout ultérieur de packages. L'objectif de ce framework est qu'il soit suffisamment ouvert et tolère l'intégration de nouvelles techniques de modélisation, en gardant la base intacte par l'intermédiaire d'une séparation explicite entre les packages principaux et ceux des techniques de modélisation et de simulation.

Les packages principaux sont au nombre de quatre et constituent la partie générique, indépendante des techniques de modélisation utilisées (voir Figure 2.7) :

- Le **package moteur-DEVS** contient les classes nécessaires à la définition de modèles et à leur simulation.
- Le **package cadres-expérimentaux** contient les classes de base pour permettre la simulation de multimodèles à l'intérieur d'une interface homogène.
- Le **package interface-graphique** contient les composants graphiques permettant la composition visuelle et interactive de modèles.
- Le **package stockage** permet de formater et de lire les modèles en utilisant le langage de balise XML (eXtended Markup Language) et une DTD ad hoc.

[15] EtherPad est un éditeur de texte en ligne, sous licence Open Source, fonctionnant en mode collaboratif et en temps réel (Voir http://etherpad.org/).

Figure

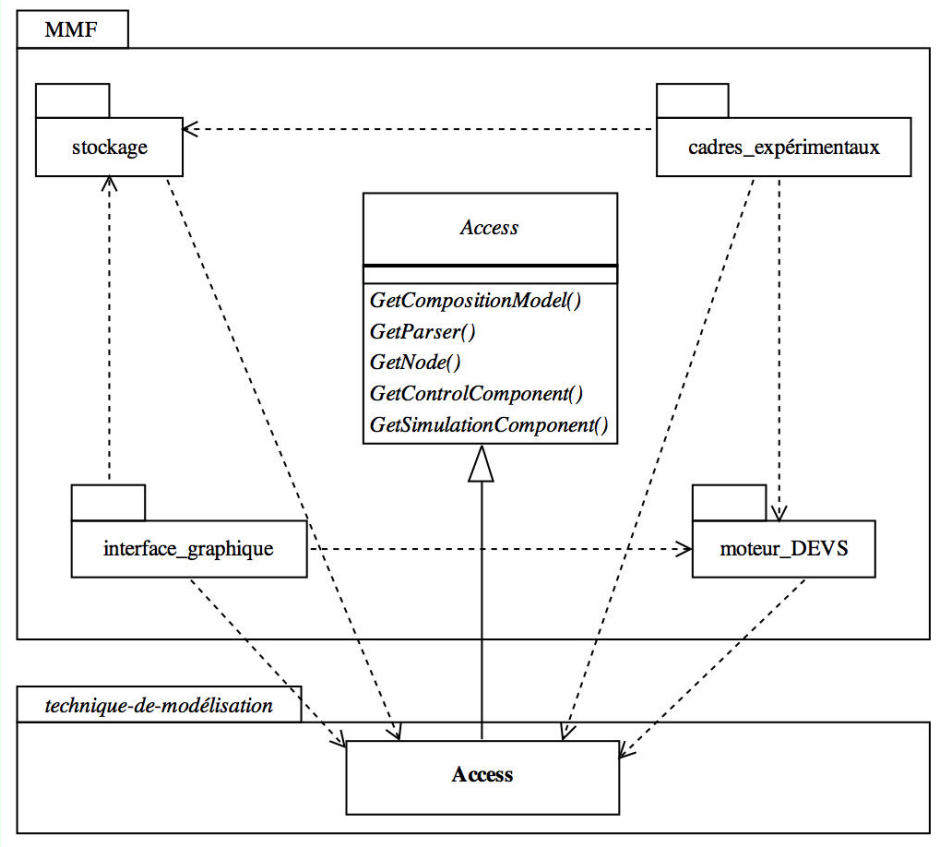


FIGURE 2.7 – Structure de JDEVS

Les packages des techniques de modélisation et de simulation sont développés indépendamment et peuvent être ajoutés de manière séparée. Les packages implémentés sont :

- **Feedback-DEVS**, qui permet l'intégration de modèles à comportements adaptatifs dans le framework de modélisation générique.
- **Vector-DEVS**, qui offre la spécification de modèles basés sur les vecteurs, avec le formalisme DSDEVS (Dynamic structure DEVS).
- **raster-DEVS**, package qui comprend les techniques de modélisation spatiale basées sur les automates cellulaires.

L'implémentation de ce framework de multimodélisation[16] a donné JDEVS, utilisé en-

[16] Un **multimodèle** est une composition de plusieurs sous-modèles homogènes ou hétérogènes avec différent

suite dans des expérimentations. Nous pouvons notamment citer la modélisation des "polluants dans un bassin versant", résultat du couplage entre **un modèle de réseau de neurones (Feedback-DEVS)** pour la concentration de nitrate et un **modèle cellulaire** pour la propagation de polluants.

Ainsi JDEVS permet le couplage entre différents types de modélisation en utilisant comme dénominateur commun DEVS, grâce à l'utilisation d'interfaces d'entrées/sorties bien définies. Le formalisme DEVS a été adopté comme base unificatrice assurant la coopération de différents types de modèles. Bien que JDEVS n'aborde pas directement la collaboration entre thématiciens durant la modélisation initiale, il permet aux utilisateurs d'unir les connaissances de ces thématiciens appliquées sur des modèles atomiques, qui sont ensuite couplés.

2.3.3 Synthèse

Cet état de l'art sur cette problématique, qui est un de nos deux axes de recherche, montre les limites de la modélisation actuelle dans le domaine de la SOA par rapport à la co-construction.

Beaucoup de méthodologies insistent sur la composition et l'assemblage de modèles provenant d'experts différents, ou encore sur les outils de collaborations externes au processus de modélisation et de simulation. D'autres méthodologies permettent, grâce à des "retours" ou des "cycles", une certaine dynamique dans le processus d'affinage du modèle avec une collaboration entre thématiciens et modélisateurs, ou entre modélisateurs et modélisateurs-informaticiens.

Cependant, si des méthodologies existent dans ce domaine, elles n'intègrent pas la collaboration entre thématiciens durant la phase initiale de modélisation. Les challenges principaux de [Nikolic *et al.*, 2007] et de [Sinha *et al.*, 2001] constituent des pistes pour répondre à ce problème :

1. De codifier, de traduire et d'intégrer la connaissance provenant de disciplines différentes dans le modèle.
2. D'avoir une représentation du modèle, de manière à permettre sa cohérence et sa consistance, malgré la pluridisciplinarité, avec certains niveaux d'abstraction.

Ces deux points, présentés dans le domaine général de la modélisation, s'appliquent bien au domaine de la SOA. Plus loin, nous proposerons une méthodologie et une architecture orientées pour relever ce challenge.

niveau d'abstraction possible [Amblard *et al.*, 2001].

2.4 Problématique de la réutilisation de modèles

2.4.1 La réutilisation en général

La problématique de la réutilisation de modèles suscite plusieurs questions développées ci-dessous.

Question n°1 : L'objet de la réutilisation

La première interpellation est la suivante : Que voulons-nous réutiliser ? Il ne faut pas faire de parallèle strict entre la réutilisation des modèles de simulation et la réutilisation des logiciels. Ce sont deux domaines différents, bien que liés entre eux.

Figure

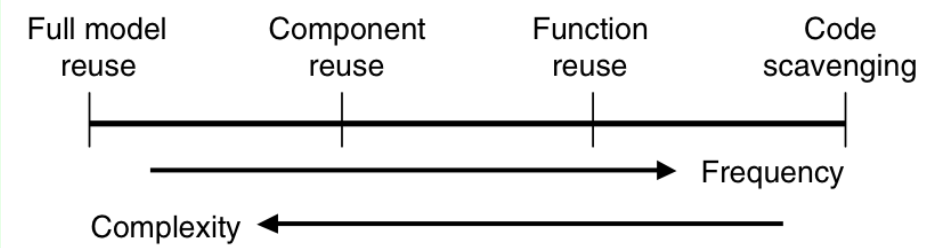


FIGURE 2.8 – Spectre de réutilisation selon [Pidd, 2002]

La réutilisation de modèles ne signifie pas simplement la réutilisation des modèles de simulation complets. L'auteur de [Pidd, 2002] nous propose un "spectre de réutilisation" (voir Figure 2.8) centré sur quatre positions et deux axes horizontaux différents.

- Les positions sont :
  - **Code scavenging :** Le recyclage de code consiste à prendre un morceau de code qui semble fonctionner et à l'utiliser pour créer de nouvelles structures, fonctions…
  - **Fonction reuse :** La réutilisation des fonctions est parfois nécessaire. Les fonctions réutilisées sont généralement très spécifiques dans leurs fonctionnalités, à granularité souvent fine : on vérifie ainsi facilement qu'elles s'exécutent comme prévu.
  - **Component reuse :** La réutilisation des composants consiste à reprendre un module encapsulé avec une interface définie, fournissant certaines fonctionnalités et apte à être utilisé dans une architecture définie. Sur le spectre de réutilisation, cette position marque le début des difficultés, au vu de la taille et de la complexité des composants.

  - **Full model reuse** : La réutilisation du modèle complet est l'objectif de beaucoup de recherches, mais aussi source de nombreuses problématiques.

- Les axes sont :

  - **La fréquence**, qui nous indique que la réutilisation est beaucoup plus fréquente à l'extrémité droite du spectre. Il est vrai que le recyclage de code (*code scavenging*) est courant pour tout modélisateur-informaticien.
  - **La complexité**, qui va dans la direction opposée à la précédente. Ainsi, le recyclage de code est relativement facile, tandis que la réutilisation efficace des modèles entiers de simulation peut s'avérer très difficile.

D'après ce spectre, la réutilisation à l'extrémité gauche du spectre est problématique, alors que celle de la partie droite est plus simple.

Question n°2 : Les avantages de la réutilisation

Cette interrogation est en relation avec l'effort fourni pour l'adaptation d'un modèle et le retour sur investissement (temporel, financier, en terme de connaissances...).

Le premier écueil est le questionnement suivant pour le primoconcepteur : Pourquoi investir autant d'efforts pour améliorer la réutilisabilité d'un modèle pour une utilisation tierce ? Quelles connaissances en retirera-t-il ? Se lancer dans un projet avec cette intention consommera beaucoup de ressources, notamment du fait de la complexité au moment de la modélisation.

Le second écueil tient à celui qui va réutiliser le modèle : Quels seraient les coûts pour comprendre puis adapter le modèle dans le cadre du projet concerné ? Quels bénéfices en terme de capitalisation de connaissances ?

Bien sûr, si on veut s'éloigner d'un des écueils, on se rapprochera de l'autre ! Moins le primoconcepteur s'investit pour favoriser la réutilisation, plus le réutilisateur rencontrera de difficultés. Il faut trouver le juste milieu, si tant est qu'il existe !

Il est bien évident que la réutilisation de modèle, dans un cadre idéal, apporte beaucoup d'avantages, autant pour celui qui le réutilise que pour le modèle lui-même. Le réutilisateur réduit les coûts et bénéficie d'un gain de temps, d'argent et de connaissances proportionnel à la facilité d'adaptation du modèle. Ce dernier, lui, s'enrichit car le réutilisateur apporte des améliorations, une flexibilité au niveau de la granularité si nécessaire, ce qui, au final, peut améliorer la qualité du modèle.

Par rapport à cette notion d'investissement d'effort, les auteurs de [Balci et Ormsby, 2007, Balci *et al.*, 2011] estiment de manière générale que la réutilisabilité (niveau de réutilisation possible) et la composabilité (niveau de combinaison possible) dans un modèle conceptuel pourraient être grandement augmentées dans le domaine de la modélisation et de la simulation si une organisation parrainait le développement et la maintenance de modèles conceptuels en relation avec leurs centres d'intérêt. C'est le cas de l'Agence fédérale

des situations d'urgence (Federal Emergency Management Agency, FEMA), organisme gouvernemental américain destiné à assurer l'arrivée des secours en situation d'urgence et la gestion des projets.

Question n°3 : La validité de la réutilisation

La dernière interrogation, et sûrement la plus importante, concerne la validité lors d'une réutilisation. [Robinson *et al.*, 2004, Robinson, 2004] soulèvent le problème de la validité et de la crédibilité d'une simulation : le plus important n'est pas la fidélité du modèle par rapport à la réalité, mais bien sa validation, et précisent dans leurs articles que cette validité[17] est en fait un certain niveau de confiance et de précision dans le modèle par rapport au contexte. En effet, un modèle peut ne pas être d'une très grande fidélité par rapport à la réalité, mais suffire largement aux besoins du projet. Cette concession est souvent faite dans un souci d'économie de temps : une plus stricte fidélité n'apportera pas forcément une validation plus conséquente. Il est parfois nécessaire d'avoir une modélisation très fine d'un système dans un cas, mais une modélisation moins détaillée du même système suffirait dans un autre.

La réutilisation d'un modèle repose aussi sur la confiance : le primoconcepteur l'a-t-il bien élaboré ? Est-ce que ce modèle, adapté à un contexte particulier, est valide dans un autre ? De plus, une approche différente peut être envisagée suivant la granularité requise de la simulation et sa finesse. Le niveau d'abstraction n'est-il pas trop haut ou trop bas entre les deux projets ?

C'est pour ces raisons qu'un modèle de simulation déjà validé ne sera uniquement valide (automatiquement) que pour le but et le contexte le concernant. Il faudra donc revalider le modèle réutilisé pour un but et/ou un contexte nouveau.

Le processus de modélisation de simulation n'est pas conçu pour trouver **LA** réponse. Il est là pour aider à prendre les mesures adéquates, mieux cerner, comprendre et aborder les problèmes rencontrés. [Epstein, 2008] énumère plus de 16 raisons de construire des modèles. La sortie numérique ou visuelle du modèle de simulation n'a parfois aucune valeur intrinsèque. Sa raison principale est de permettre d'en savoir plus sur les processus en interactions dans un environnement complexe, les relations entre les variables...

Les modèles génériques sont des cas particuliers de réutilisations de modèles/composants. Il est bien évident, selon Robinson, que l'utilisation correcte d'un modèle dit "générique" implique la réutilisation du modèle dans un contexte différent, mais pour le même but. Avec le développement du partage sur le Net, le besoin d'économie d'argent et de temps, beaucoup se sont intéressés à ce genre de réutilisation.

C'est de ce besoin que sont nés les COTS (Commercial Off-The-Shelf, composants pris sur étagère) : ce sont des plateformes génériques et non dédiées à un projet en particulier, contenant des librairies, des exemples, *etc.* De manière générale, selon Robinson, il existe trois formes de réutilisation qui doivent être validées :

- **La réutilisation des composants-modèles de base de la plateforme :** Ce cas impose

[17] La définition proposée par les auteurs est la validation est "*A perception, on behalf of the modeller, that the conceptual model will lead to a computer model that is sufficiently accurate for the purpose at hand*".

beaucoup de modifications. Les composants de base étant souvent très simples, il faudra forcément une série de tests obligatoires, donc une validation et du temps.

- **La réutilisation d'un sous-système d'un modèle :** L'utilisation d'un sous-système d'un premier modèle pour un second modèle n'apporte souvent qu'un gain de temps négligeable, voire nul, vu que le réutilisateur consacre beaucoup de temps à comprendre puis adapter ledit sous-système.
- **La réutilisation d'un modèle similaire :** Comme il n'existe pas deux modèles exactement similaires et qu'il faut toujours les adapter par rapport au but et/ou au contexte, cette réutilisation nécessite beaucoup de travail pour la validation.

Un exemple de contexte où la réutilisation complète du modèle doit être effectuée avec précaution est celui du modèle **D**OMINO **S**MAT (DS), présenté dans le Chapitre 3 : des spécificités particulières (démographie, politique...) ne permettent pas une transposition directe à d'autres contextes.

Éléments de réponses sur la validité de la réutilisation

La question de la validité est très importante. Il est impossible de donner une réponse générale, mais la littérature propose quelques points intéressants détaillés ci-après.

Figure

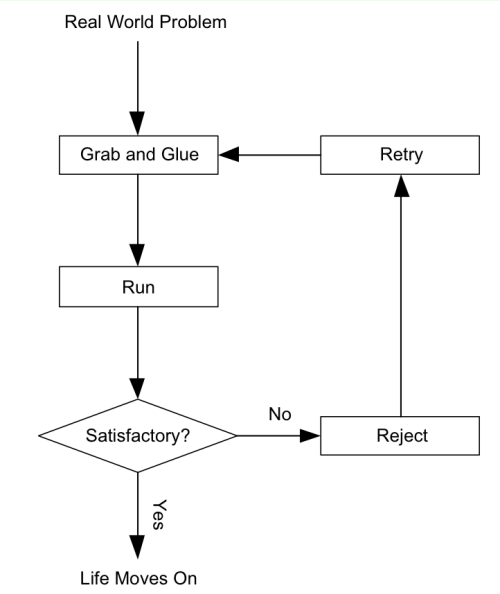


FIGURE 2.9 – Le processus de G^2R^3 selon [Paul, 2002]

G^2R^3. Une approche intéressante, présentée par [Paul, 2002] et reprise dans [Paul et Taylor, 2002], est le G^2R^3 (*Grab-and-Glue, Run, Reject, Retry*). Cette technique (voir Figure 2.9) consiste simplement à prendre des morceaux de modèles, les coller ensemble et faire tourner le tout. Si le résultat n'est pas satisfaisant, il suffit de recommencer.

L'originalité de cette approche réside dans l'utilisation des fragments de modèles séparés sans forcément les valider. Le modélisateur et son client n'ont plus les mêmes rôles. Traditionnellement, celui du client consistait à obtenir une modélisation valide par rapport à ses demandes tout en lui permettant de prendre des décisions en fonction des résultats. Ici, le modélisateur et le client travaillent ensemble non pour avoir une modélisation valide, mais une modélisation pour aider le client à comprendre son problème et à prendre des décisions grâce à la compréhension des interactions.

La technique de G^2R^3 est efficace si nous faisons abstraction des problèmes technologiques. Des auteurs tels que [Eldabi *et al.*, 2003] et [Lee et Chen, 2005] ont éclairé ce point au travers de l'application de G^2R^3 à leurs simulations. Le second article montre qu'ils n'ont pas réussi à prouver le gain de temps dans l'utilisation de cette technique, en particulier à cause des erreurs de syntaxe (Java) lors de l'étape "gluing" (collage ou fusion) des modèles. Ce problème est d'autant plus crucial que les logiciels de simulations sont distants [Sargent, 1986] au niveau de la méthodologie, du langage...

Documentation. Les supports d'informations sur la vérification et la validation du modèle convainquent généralement les utilisateurs de la "justesse" d'un modèle et de ses résultats et devraient être inclus dans la documentation du modèle de simulation. [Gass, 1984] explique comment appuyer les modèles informatiques par l'intermédiaire d'un ensemble de textes.

Ces écrits sur la vérification et la validation doivent être disponibles en deux versions :

- **La documentation détaillée** qui inclut des détails sur les tests, les évaluations faites, les données, les résultats, *etc.* Elle inspirera plus de confiance en ce modèle au réutilisateur.
- **La documentation sommaire** qui contient différentes tables d'évaluations, comme le montre [Sargent, 1996] : pour la validité des données et celle du modèle conceptuel, la vérification du modèle informatique, la validité opérationnelle ainsi qu'un résumé global. Cette documentation, plus courte, facilite la tâche des adeptes du *Grab-and-Glue*.

V&V. Toujours dans le domaine de la simulation (générale), [Robinson, 1997] discute de l'importance de la Vérification et de la Validation (V&V), des différentes méthodes pour y parvenir, à différents niveaux.

La Vérification consiste à s'assurer que le modèle conceptuel a été transformé en modèle informatique avec une précision suffisante. Il peut s'agir d'une vérification des algorithmes comme celle du code par exemple. Elle revient à contrôler que le modèle est assez précis pour le but recherché, avec un certain degré de confiance. Cette première partie du V&V n'est pas si simple. Cette étape peut s'avérer parfois laborieuse en l'absence de méthodologie.

Figure

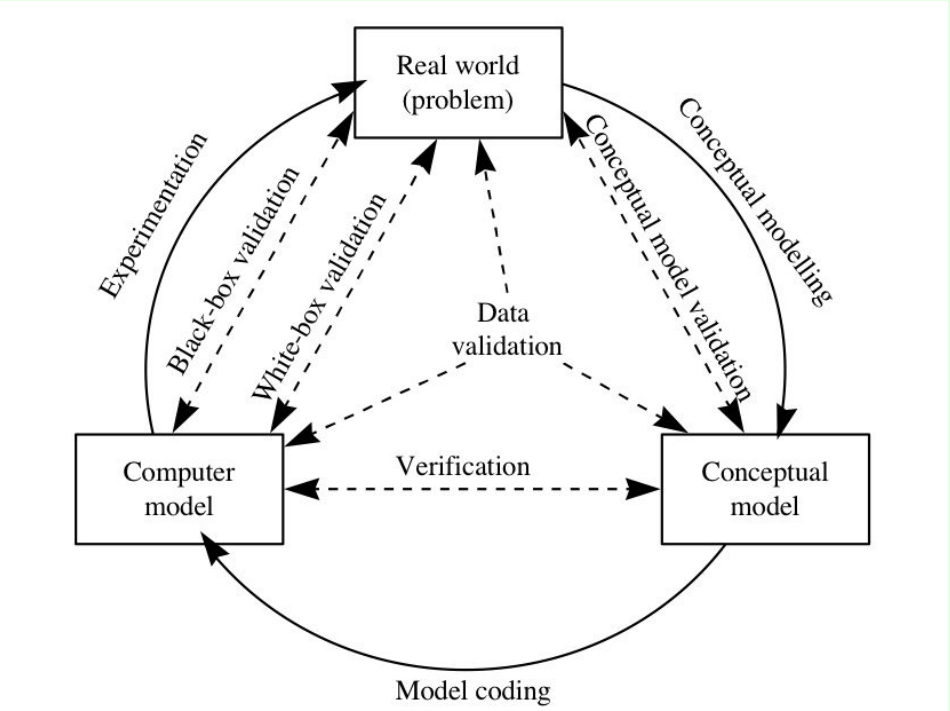


FIGURE 2.10 – Le processus de Vérification et de Validation selon [Robinson, 1997]

La seconde partie du V&V, la Validation, s'avère encore plus problématique : il est impossible d'avoir une correspondance totale par rapport au monde réel [Pidd, 2009] (ou à ce que le concepteur désire représenter). De plus, il n'existe pas de modèle valide de manière générique : il ne l'est que pour un but précis, dans un contexte précis. Faire un modèle valide générique consisterait à créer un modèle valide dans tous les cas d'utilisations possibles et probables, en tenant compte des différentes données et interactions potentielles. Un tel modèle demanderait une somme impressionnante de travail pour aboutir.

Dans son article, [Robinson, 1997] propose que le processus de validation se déroule en quatre étapes :

- **Conceptual Model Validation :** Cette validation permet de déterminer si le modèle conceptuel contient les éléments nécessaires pour atteindre les objectifs de la simulation.
- **Data Validation :** On vérifie la précision des données pour la construction du modèle, son expérimentation et sa validation. Il faut prendre en compte la présence d'un écart im-

plémentatoire entre le modèle conceptuel et le modèle informatique (voir Section 6.2.1).

- **White-box Validation :** C'est l'étape la plus intéressante, semble-t-il, dans le cas des SOA. Il s'agit de déterminer, au niveau micro, si chaque partie du modèle représente la réalité avec une précision suffisante.
- **Black-box Validation :** Ici, on vérifie, au niveau macro cette fois, si le modèle en totalité représente le monde réel correspondant avec une précision suffisante.

Le V&V ne prend pas corps lorsque le modèle a été complètement développé. Mais ce processus prend tout son sens quand il est fait périodiquement, en continu, dès le début du projet et dure tout au long de sa vie. La seule partie du processus qui attend la fin du projet est la "Black-box" validation.

Beaucoup de techniques visent à mettre en place ce processus (voir [Balci, 1994] pour de plus amples détails), qui peuvent être de type :

- Informel : dépendant fortement du raisonnement humain.
- Formel : basé uniquement sur des preuves mathématique et algorithmique.
- Statique : résultant de l'étude du code source du modèle.
- Dynamique : axé sur l'exécution du code du modèle.
- Symbolique : associé au comportement du modèle durant son exécution.
- Contraint : permettant de tester des points particuliers du modèle.

Ce processus de V&V a pour unique but d'augmenter la confiance dans le modèle simulé, car il est impossible d'obtenir un modèle absolument correct. Ces techniques appliquées lors d'une réutilisation de modèle répondent à cette question, à condition bien sûr que le modèle réutilisé ait bien été documenté. Plus tôt sera fait ce travail de V&V dans le cadre d'une réutilisation, plus rapidement nous aurons une "idée" de la crédibilité à accorder à ce modèle dans un autre but.

La validation dans la SOA

Dans un Workshop, Macal [Macal, 2005] expose le problème de la validation des SOA. La validation dépend intrinsèquement de la raison pour laquelle la modélisation a été faite. Nous modélisons et simulons pour les raisons suivantes :

- Nous pensons de manière linéaire, nous ne sommes pas capables de comprendre toutes les forces en action et la façon dont elles interagissent.
- Nous ne pouvons imaginer toutes les possibilités du monde réel.
- Nous sommes incapables de prévoir tous les effets de plusieurs événements en cascade ni un élément nouveau que notre imagination n'aurait pu concevoir.

La raison principale de la modélisation est d'avoir un aperçu des causes et effets, pour établir des arguments qui prédiront à quelle(s) condition(s) un événement se réalise ou pas. Nous modélisons donc surtout pour faire des prédictions qualitatives (et parfois quantitatives).

Selon [Sargent, 1999, Chan *et al.*, 2010], un des principaux avantages de la SOA par rapport aux autres types de simulations (notamment la simulation à événements discrets) est le processus d'observation des schémas d'interactions. Cet atout, dépendant grandement de la plateforme de simulation, conduit de plus en plus de chercheurs vers la SOA. Même si les animations produites lors d'une simulation ne sont pas toujours nécessaires au sein d'une SOA, leur présence permet de visualiser des dynamiques et interactions entre les agents, et jusqu'à un certain point, il s'agit d'un moyen puissant de vérifier, valider et expliquer le modèle.

De plus, il est mentionné dans [Davidsson, 2001] certaines différences entre SOA et d'autres types de simulation, ainsi que ses avantages, telle la facilité d'implémenter et de comprendre les interactions et comportements, de faire évoluer le modèle au sein du même projet.

Les auteurs de [Law *et al.*, 2007] expliquent que, contrairement aux autres paradigmes de simulation[18] qui ont donné lieu à un riche ensemble de théories et de pratiques sur la validation, le paradigme de la SOA en est encore à ses débuts en termes de techniques en rapport avec la modélisation. Il est nécessaire de se pencher encore plus sur le développement analytique, l'optimisation et la validation.

La validation par l'observation de la SOA. Ce sujet a fait l'objet de plusieurs études. Notre équipe a porté une thèse sur le sujet : "Observation de simulations multi-agents à grande échelle" [Ralambondrainy, 2009]. Dans son ouvrage, Ralambondrainy a établi un état de l'art de l'observation au sein de la SOA, en étudiant la visualisation d'informations, les interfaces intelligentes, les agents intelligents et enfin l'approche agent.

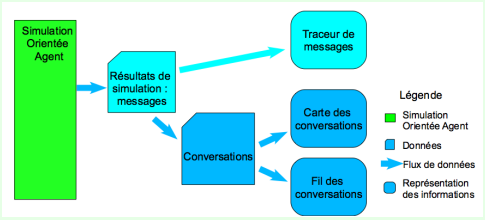
Puis il a identifié seize exigences pour le processus d'observation, classifiées comme étant : de modèle, de mode d'exécution, d'architecture et de techniques.

Il a ensuite développé une ontologie d'observation. Elle fournit une compréhension commune de la structure de l'information entre les acteurs du processus de simulation et une clarification de la construction de plateformes. Cette ontologie présente ce qui est considéré comme implicite dans le processus d'observation et ainsi donne une base réutilisable dans ce domaine. Elle peut aussi servir de base à la spécification logicielle pour le processus d'observation dans les plateformes de SOA.

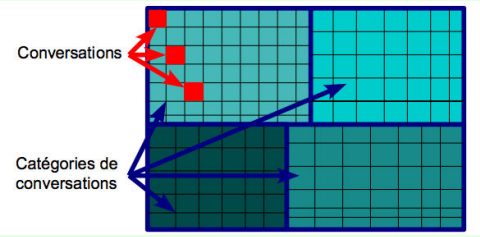
Il propose aussi une architecture de plateforme orientée "observation", afin de prendre en compte l'exigence de distribution observationnelle. Il développe un "concept de conversation" (voir Figure 2.11) et des vues génériques donnant une représentation de haut niveau observationnel par rapport aux diagrammes de séquence agent et facilitant la compréhension des interactions au sein des systèmes multi-agents en général et de la simulation en particulier. Ces interactions sont justement à la base de la dynamique du système et des phénomènes émergents que l'on souhaite étudier.

[18] La Simulation à Evénements Discrets, la Dynamique des Systèmes ou la simulation basée les éléments finis.

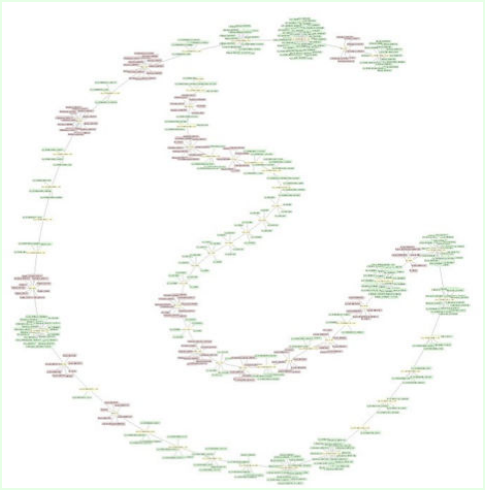
Figure



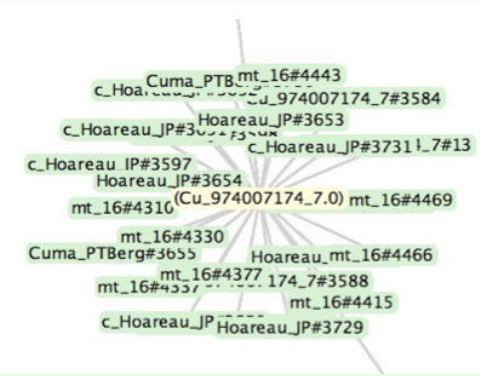
(a) Le concept de conversation est à l'origine de nouvelles représentations synthétiques.



(b) La carte des conversations présente l'ensemble des conversations d'une simulation dans un même espace. Les conversations peuvent être classées par catégories.



(c) Le fil de conversations représente un sous-ensemble de conversations reliées sémantiquement.



(d) Le zoom sur une boule fournit le détail de la conversation correspondante ainsi que des messages qui la constituent

FIGURE 2.11 – Le concept de conversation proposé par [Ralambondrainy, 2009]

Une des connaissances les plus intéressantes qu'apporte la SOA est non de savoir qu'on passe d'un événement A à un événement B, mais de répondre aux questions :

- Quelles sont les interactions qui ont entraîné le passage de A à B ?
- Qui est l'acteur de ce changement ?
- Quel est le contexte (position, dimension sociale...) ?

Dans un système complexe, de manière générale, il est quasiment impossible de tout vérifier et valider [Oreskes *et al.*, 1994]. Cependant, les systèmes multi-agents apportent

quelques propriétés intéressantes, car les agents sont les singletons du système, ayant à la base des interactions simples. De plus, les modèles de la SOA sont généralement plus faciles à entretenir : le raffinement de modèles entraîne souvent des changements très locaux avec des résultats globaux.

2.4.2 La réutilisation et la Simulation Orientée Agent

Dans le domaine de la SOA, la réutilisation est facilitée par la notion atomique de l'agent. Parmi les techniques permettant la réutilisation, nous en avons choisi quelques-unes aux objectifs semblables. Notre principale contrainte est de pouvoir réutiliser aussi bien les modèles que les agents eux-mêmes et leurs comportements.

IODA

La méthodologie **I**nteraction **O**riented **D**esign of **A**gent simulations (IODA) est issue à l'origine de travaux de Jean-Christophe Routier, Philippe Mathieu et Sébastien Picault en 2001 [Mathieu *et al.*, 2001]. Depuis, elle est en constant développement comme le prouvent les thèmes de recherches de cette équipe [Gaillard *et al.*, 2010]. Si nous nous positionnons par rapport à cette méthodologie, c'est qu'elle est fondée sur trois règles [Kubera, 2010] :

1. Toute entité pertinente du phénomène simulé est concrétisée par un agent.
2. Tout comportement d'un agent est concrétisé par une interaction.
3. Toute interaction est décrite indépendamment des spécificités des agents.

La seconde règle est celle qui nous rapproche de IODA. IODA est fondée sur un modèle formel mathématique. Il décrit les divers concepts de l'approche d'une méthodologie de conception fournissant des outils graphiques. Ceux-ci permettent de construire progressivement un modèle. Des algorithmes exploitent ces informations afin de générer une implémentation de la simulation.

Cette méthodologie est complétée par une plateforme de simulation appelée **J**ava **E**nvironment for the **D**esign of agent **I**nteractions (JEDI), un environnement de développement intégré nommé JEDI-BUILDER et un explorateur de l'espace des simulations **L**ets you **E**xplore **I**nteractions for your **A**gents (LEIA). Cet ensemble d'outils (voir Figure 2.12) permet notamment de valider le modèle IODA.

La base de l'IODA est le concept d'interaction où nous avons une *source* et une *cible*. Dans notre contexte, nous nous intéressons à la méthodologie détaillée de conception IODA, plus particulièrement aux différentes étapes de la méthodologie (voir Figure 2.13). IODA, étant une méthodologie en constant développement depuis plus de dix ans, est très détaillée. Elle se résume comme suit :

A. **La "topologie de l'environnement"** consiste à donner un sens à la notion de distance dans la simulation.

B. **L'"identificateur des familles d'entités"** est une liste d'identifiants pour celle-ci.

Figure

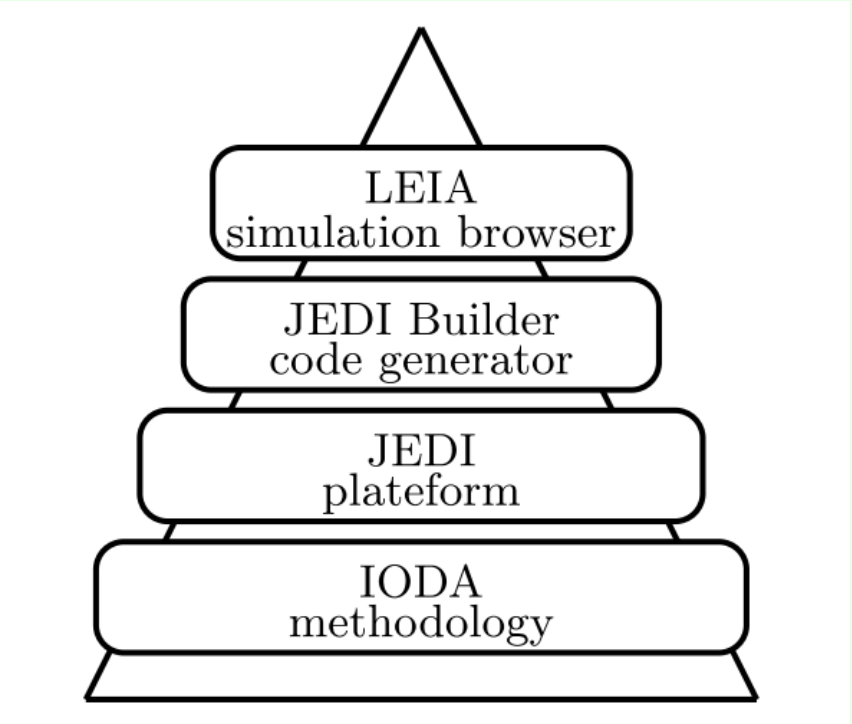


FIGURE 2.12 – Pyramide d'outils basés sur IODA [Gaillard *et al.*, 2010]

C. **L'"identificateur des interactions"** dresse une liste d'identifiants pour les interactions les concernant.

D. **L'étape "Préconditions, déclencheur et actions des interactions"** consiste à décrire les interactions par une signature complète.

E. **La "matrice d'interaction brute"** est le premier fruit de cette méthodologie : un tableau avec, en entrée, les entités *sources* en ligne et *cibles* en colonne (avec une colonne supplémentaire notée $\emptyset$ pour les interactions dégénérées, *c'est-à-dire* où la cible est l'entité elle-même). Cette matrice contient toutes les interactions possibles du système dans un premier temps.

F. **L'étape "halo des familles d'entités"** consiste à identifier le voisinage des entités.

G. **La "matrice d'interaction raffinée"** associe une priorité à chaque élément d'assignation de la matrice d'interaction brute.

H. **La "matrice de mise à jour"** permet de lister, sans aucune priorité, les interactions dégénérées. En d'autres termes, les entités mettent leurs états à jour au travers d'une

Figure

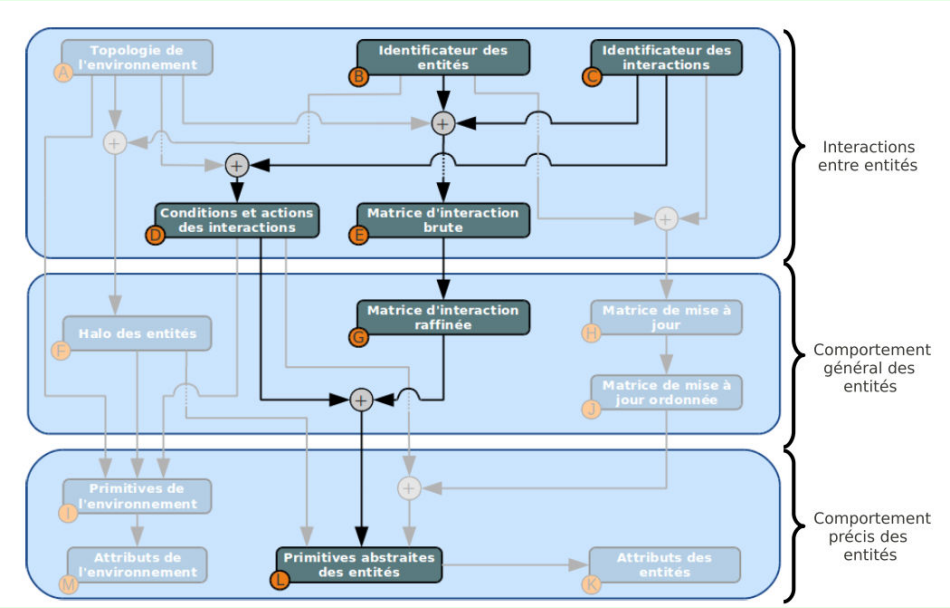


FIGURE 2.13 – Méthodologie IODA selon Kubera

interaction où l'entité est à la fois source et cible.

I. **Les "primitives de l'environnement"** : nous y trouvons par exemple celles issues des étapes A, D, F, ainsi que d'autres pour ajouter, retirer et inventorier les entités par rapport à l'environnement.

J. **La "matrice de mise à jour ordonnée"** associe une priorité à chaque élément d'assignation de la matrice de mise à jour.

K. **L'étape "attributs des familles d'entités"** consiste à terminer la spécification des familles d'entités : on identifie leurs attributs d'après l'analyse des spécifications des diverses primitives de perception, d'action décrites dans les étapes précédentes.

L. **Les "primitives d'action et de perception"** sont les descriptions des actions et perceptions en fonction des familles d'entités, grâce aux étapes antérieures (B, C, D, F, G, I et J).

M. **L'étape "attributs de l'environnement"** consiste à terminer la spécification de l'environnement en identifiant ses attributs, grâce aux étapes précédentes.

Au niveau de la réutilisation, IODA propose le concept de **spécialisation** similaire à la relation "*kind of*" (sorte de) des langages orientés objet : la spécialisation exprime une relation

sémantique entre deux familles d'agents. Lorsqu'une famille d'agents A spécialise une famille B :

- A est aussi la *cible* de toutes les interactions dont B est la cible.
- A est également la *source* de toutes les interactions dont B est la source.

Pour dépasser le concept d'héritage du langage-objet, IODA propose trois opérateurs de spécialisation :

- **L' ajout** qui permet d'ajouter à un agent la capacité d'initier une interaction avec un autre agent pour cible.
- **La modification** qui modifie la garde de distance d'une interaction que l'agent peut initier.
- **Le retrait** qui donne la possibilité de retirer à un agent la capacité d'initier une interaction avec un agent particulier pour cible.

Ainsi, IODA simplifie la conception de modèles de comportements grâce à ce système d'héritage. Cette approche est bien adaptée pour représenter simultanément plusieurs systèmes qui ont certains éléments en commun. Cet héritage suscite néanmoins quelques problèmes [Kubera, 2010] dans l'ordre des priorités d'interactions lorsque l'agent hérite de plusieurs familles d'agents mères.

Cette réutilisation repose sur l'héritage d'un ensemble d'interactions et non sur un choix de comportements spécifiques. Il est souvent utile de réutiliser un même comportement sur plusieurs agents qui ne sont pas forcément de la même famille (comme par exemple les comportements "*follow*" ou "*wander*"). De plus, IODA ne permet pas de regrouper plusieurs interactions.

AGR

Le modèle AGR [Ferber et Gutknecht, 1998, Ferber *et al.*, 2003, Michel, 2004] est axé sur l'association de trois concepts-clés : l'**A**gent, le **G**roupe et le **R**ôle. Il pose la fondation de la plateforme MadKit [Gutknecht *et al.*, 2000, Gutknecht *et al.*, 2001]. L'idée de base : un individu peut avoir de multiples rôles, dans les divers groupes auxquels il appartient.

- L'**agent** est simplement décrit comme une entité autonome communicante qui joue des rôles au sein de différents groupes. Cette définition permet de choisir la représentation interne de l'agent appropriée au domaine d'application.
- Le **groupe** est la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes.
- Le **rôle** est une représentation abstraite d'une fonction, d'un service, d'une position sociale ou d'une identification d'un agent au sein d'un groupe particulier. Ainsi, chaque agent peut avoir plusieurs rôles ; un même rôle peut être tenu par plusieurs agents.

Figure

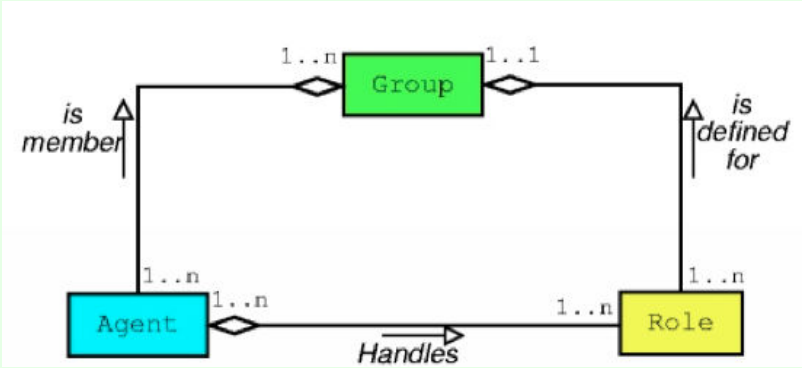


FIGURE 2.14 – Modèle organisationnel AGR

Les deux conséquences de ce modèle sont :

- Les rôles et leurs relations sont décrits en premier : niveau social.
- Les rôles sont distribués entre les agents : niveau multi-agents.

Cette méthodologie existant depuis plus de dix ans fait toujours l'objet d'affinements, comme le montrent les différents travaux de l'équipe [Amiguet, 2000, Tranier, 2007, Ferber et Stratulat, 2009]. Elle fait partie d'un ensemble plus général, le projet *Aalaadin* : analyser et exprimer divers systèmes multi-agents par l'intermédiaire des concepts organisationnels. Par exemple, le modèle AGREEN [Báez-Barranco *et al.*, 2007] est une extension du modèle AGRE (AGR + Environnement) qui lui-même est une extension du modèle AGR. Mansour présente AGRS [Mansour, 2007] : il y introduit la notion de service, description d'un ensemble de fonctionnalités que doit proposer ou demander un agent.

Le concept de rôle dans AGR permet de modéliser des systèmes complexes comme sur la Figure 2.15. Un agent peut faire partie de plusieurs groupes et avoir aussi plusieurs rôles.

Au niveau de la réutilisation, AGR, par dérivations ou agrégations de rôles, donne la possibilité de réutiliser des rôles de base. En effet, les rôles peuvent être : instancier, spécialiser, généraliser ou agréger dynamiquement. Ce sont des schémas génériques d'interactions.

Madkit propose aussi SEdit, éditeur graphique permettant de visualiser et d'animer des diagrammes de type "automates", "portes logiques" ou "réseaux de petri", qui offre des possibilités de réutilisation [Gutknecht *et al.*, 2001]. SEdit est une application construite sous forme d'un ensemble d'agents MadKit.

De la même manière que IODA, la réutilisation dans AGR au travers des rôles repose sur un ensemble d'actions dans un rôle et non sur un choix de comportements spécifiques. De

Figure

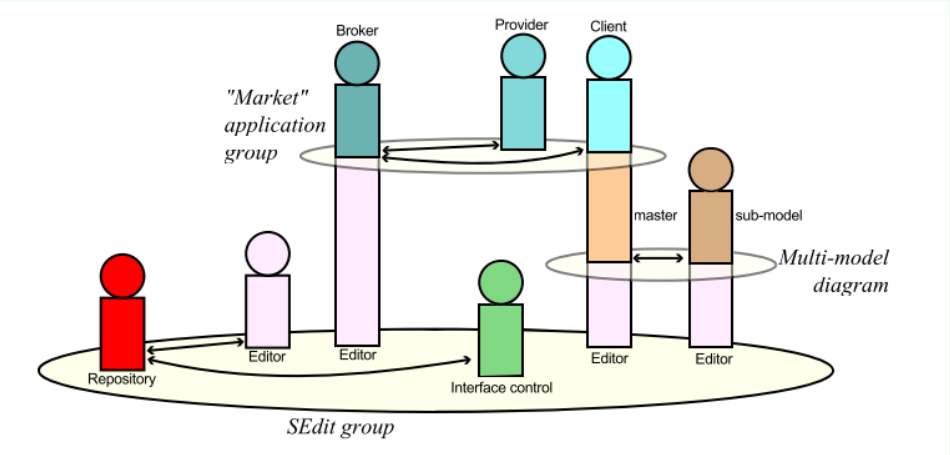


FIGURE 2.15 – Exemple de groupe dans AGR

plus, cette réutilisation de rôles semble restreinte à l'intérieur d'une simulation ou d'un projet. L'éditeur SEdit permet d'importer des agents, mais ceux-ci doivent être dans le formalisme SEdit, donc de type "automates", "portes logiques" ou "réseaux de petri". Ces agents doivent donc avoir une représentation sous cette forme.

AA4MM

AA4MM (**A**gents et **A**rtefacts pour la **M**ulti**M**odélisation) [Siebers *et al.*, 2007, Siebert, 2011] est un métamodèle pour envisager la multimodélisation[19] et le couplage de simulation que Julien Siebert a proposés dans sa thèse de doctorat. Au niveau méthodologique, AA4MM s'accompagne d'une décomposition des questions en cinq niveaux qui permet au modélisateur de spécifier chacun des aspects du multimodèle, notamment les changements d'échelles entre différents modèles :

- **Le niveau Conceptuel** porte sur les questions de représentation du multimodèle (méta-modélisation).
- **Le niveau Sémantique** est lié à la signification et au rôle du multimodèle dans le système étudié.
- **Le niveau Syntaxique** concerne la formalisation du multimodèle pour prendre en compte celle de chacun des sous-modèles.
- **Le niveau Dynamique** porte sur les questions propres à l'exécution du multimodèle et

[19] Un **multimodèle** est une composition de plusieurs sous-modèles homogènes ou hétérogènes avec différent niveau d'abstraction possible [Amblard *et al.*, 2001].

Figure

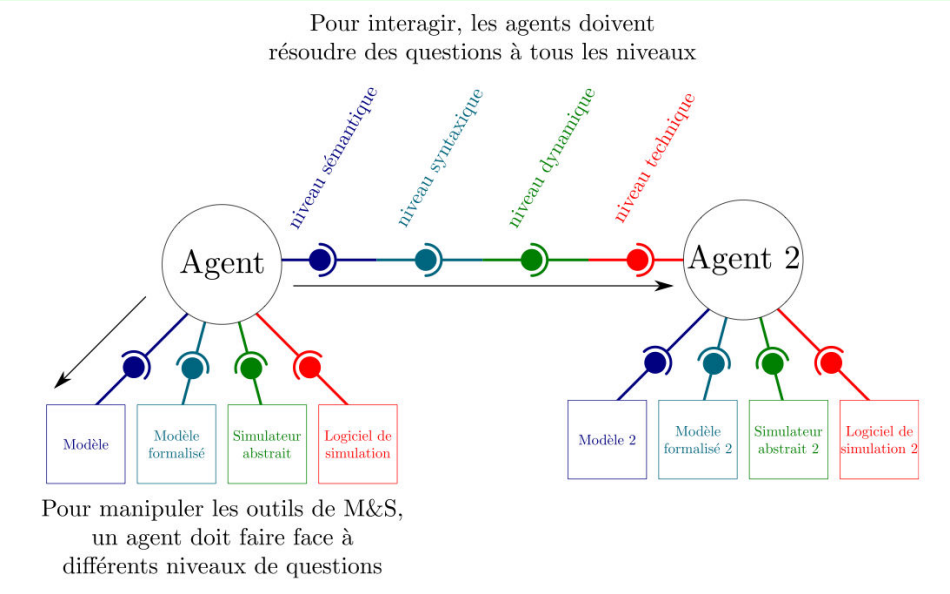


FIGURE 2.16 – Concepts mis en œuvre dans AA4MM

les problématiques classiques de la simulation distribuée.

- **Le niveau Technique** est en rapport avec l'implantation de la simulation du multimodèle, sa distribution, sa fiabilité…

Ce métamodèle est centré sur l'utilisation de *m-agents* (voir Figure 2.16), associés à un *bloc MSL* : **M**odèle (formalisé ou pas), **S**imulateur et **L**ogiciel. Les *m-agents* interagissent entre eux au travers d'artefacts (objets passifs) de l'environnement. Chacune des interactions entre *m-agents* est réifiée par un **artefact de couplage**. Les *m-agents* manipulent des modèles, des simulateurs abstraits et des logiciels de simulation afin de gérer les problèmes liés à la multimodélisation. Les interactions entre un *m-agent* et son *bloc MSL* sont réifiées par l'**artefact d'interface**.

Parmi les hypothèses posées, nous constatons qu'il est nécessaire que le *m-agent* puisse s'interfacer avec le logiciel simulateur, contrôler le simulateur abstrait, avoir des entrées/sorties sur les modèles.

Ainsi, AA4MM facilite le changement de modèles, d'échelle temporelle et de couplage entre niveaux. AA4MM offre ainsi la possibilité de prendre en compte plusieurs niveaux d'abstraction et surtout de réutiliser les modèles existants, dans des simulateurs différents. Cependant, il s'agit de réutilisation de modèles complets, non des agents et de leurs comportements. AA4MM ne satisfait donc pas totalement notre contrainte de réutilisation.

2.4.3 Synthèse

Bien qu'il existe un lien entre la réutilisation du modèle et celle des logiciels, les motivations de modélisation et de développement de logiciels sont tout à fait différentes. Celles-ci doivent être prises en compte lors du transfert des idées d'un champ à l'autre. Un spectre de réutilisation va de l'utilisation de petites portions de code, en passant par les grands composants, pour arriver au modèle complet. Chaque point de ce spectre présente divers avantages et difficultés. De plus, les avantages de la réutilisation ne sont perçus que dans les cas où on note réduction conséquente du temps et du coût pour l'élaboration du modèle.

Les obstacles à une réutilisation optimale sont nombreux. Tout d'abord, la faible motivation des développeurs de modèles à adopter des procédures permettant la réutilisation de leurs créations. Cette démarche augmente le coût de développement du modèle sans qu'ils en perçoivent eux-mêmes les avantages : ces derniers seraient acquis par d'autres. Ensuite, une des principales préoccupations est le niveau de la confiance à accorder à un modèle ou un code obtenu dans un autre contexte. Enfin demeurent le temps et le coût de la familiarisation avec le modèle ou le code de quelqu'un d'autre, qui peuvent excéder le temps et les coûts-avantages de la réutilisation.

Pour que la réutilisation soit efficace, les auteurs de [Robinson *et al.*, 2004] proposent de travailler minutieusement à une stratégie bien planifiée pour une architecture bien conçue, agréée par tous. Pour atteindre cet objectif, les auteurs suggèrent la réutilisation basée sur les composants : l'outil de simulation doit être préparé pour permettre la réutilisation, par l'intermédiaire de composants génériques pouvant être facilement ajoutés ou extraits.

Les méthodologies en SOA dans ce domaine existent. Cependant, dans le spectre de réutilisation, elles sont souvent limitées au niveau du modèle complet ou de l'agent avec l'impossibilité de "descendre" davantage. Notre objectif est de promouvoir la réutilisation aussi bien des modèles que des agents et de leurs comportements.

Comme l'observent [Sullivan et Knight, 1996] dans le domaine du Génie Logiciel, si nous voulons que les composants soient "composables", ils doivent être conçus dans cette optique. La réutilisation doit être pensée dès le départ car, comme dans le domaine du Génie Logiciel [Reese, 1987], l'ajustement *a posteriori* demandera plus d'efforts que le développement d'un composant, si celui-ci n'a pas été pensé dans ce but. Nous estimons que la même observation s'applique au domaine de la SOA.

Sans aller jusqu'à fournir une librairie de composants, il nous faut décrire une manière d'articuler notre modélisation agent pour faciliter la composabilité au niveau des agents mêmes. Si nous repensons l'agent dès le départ dans cette optique, nous pouvons proposer une manière efficace de faciliter la réutilisation de modèle dans la SOA.

2.5 Conclusion

Nous avons présenté dans ce chapitre les éléments qui nous ont guidés durant notre thèse : la co-construction des modèles, leurs réutilisations. Ces deux problématiques font l'objet de plusieurs travaux dans le domaine général de la modélisation et parfois aussi dans

le domaine de la simulation. Cependant, dans le domaine plus spécifique de la SOA, nous pouvons constater un manque de méthodologie sur ces deux points malgré un besoin fort de la part des thématiciens.

Les méthodologies existantes pour la co-construction ne permettent qu'un faible niveau de collaboration notamment entre thématiciens durant la phase initiale de modélisation ni celle des thématiciens avec les modélisateurs ou les modélisateurs-informaticiens. Pour faciliter cette co-construction, nous proposons de suivre une méthodologie de conception favorisant cette collaboration. De la même manière, pour réduire les obstacles à une réutilisation optimale, nous proposons de repenser certains concepts de base des agents.

Ces deux objectifs constituent les lignes directrices de nos travaux pour proposer une méthodologie et une architecture d'agent permettant l'intégration et la réutilisation de connaissances pluridisciplinaires pour la SOA.

3 L'approche DOM et l'expérience EDMMAS

Plan du chapitre

3.1 Introduction

Après la création et la validation d'une nouvelle application, les experts veulent parfois la réutiliser, totalement ou partiellement. Comme nous l'avons expliqué dans le Chapitre 2, le réemploi un modèle dans un nouveau contexte permet de l'améliorer (en l'affinant de plus en plus), mais aussi de réduire le temps (et les coûts) de conception de nouvelles applications. L'idée de la méthodologie nommée "**D**ynamic-**O**riented **M**odeling" ou encore **M**odélisation **O**rientée **D**ynamique (DOM) [Payet *et al.*, 2006] est née de ce besoin de la réutilisation de modèles dans le contexte de la **S**imulation **O**rientée **A**gent (SOA). Un besoin devenu aujourd'hui urgent face à la complexité de plus en plus importante des systèmes étudiés. Une complexité qui rend l'élaboration des modèles longue et difficile. Les résultats obtenus sont donc de plus en plus précieux et leurs réutilisations indispensables.

Dans ce chapitre, nous présenterons tout d'abord DOM. Ensuite, pour illustrer notre expérience dans la réutilisation de modèle, nous évoquerons le projet **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation (EDMMAS). C'est à travers de telles expériences, dans un contexte fortement pluridisciplinaire, que les limites de la méthodologie DOM commencent à être posées. EDMMAS constitue ainsi notre première contribution en tant que mise en oeuvre d'un cas concret mettant en évidence la problématique à laquelle la suite de nos travaux répond.

3.2 Présentation de la Modélisation Orientée Dynamique

DOM est une méthodologie de modélisation structurant les modèles afin de rendre leurs composants autonomes, ce qui facilite leur réutilisation et leur intégration dans la conception de futurs projets. Elle est axée sur un principe qui permet de différencier les composants du modèle et sur l'utilisation de l'environnement en tant qu'élément de couplage de ces composants. Par le principe de la séparation des dynamiques, nous pouvons réduire la complexité du système global en le "cassant" en plusieurs sous-systèmes plus simples.

Définition 9

Une dynamique est l'association d'un ensemble d'activités qui participent à une caractéristique majeure d'un système complexe.

Par exemple, la *distribution de courant* par les réseaux de distribution électrique, *la production* par les centrales ou même les panneaux solaires individuels ou *la consommation d'énergie* par les usines et les secteurs résidentiels sont des activités qui participent à la dynamique de l'évolution de l'énergie (électrique).

En identifiant isolément chacune des dynamiques, nous pouvons :

- Concevoir plus aisément un modèle complexe dans lequel un grand nombre de dynamiques intervient.

- Faciliter l'intégration et la réutilisation de ces dynamiques dans un autre modèle.

Ainsi, la méthodologie DOM décompose un système complexe en sous-systèmes sur la base d'un ensemble de dynamiques pré-identifiées. L'environnement est ensuite utilisé comme milieu de couplage des dynamiques, ce qui permet au modèle global de former un tout. L'introduction de l'expression de ces dynamiques dans le modèle conceptuel facilite la réutilisation dans les projets de modélisation de système complexe.

DOM se focalise donc sur les dynamiques qui entrent en jeu dans les systèmes à modéliser et simuler. Cette méthodologie est basée sur l'intégration de plusieurs couches d' "environnements", chaque couche étant appelée Modèle mono-dynamique (MDM [1]). Chacun de ces sous-modèles MDM contient exclusivement la connaissance nécessaire à son ensemble d'activités, ce qui permet au concepteur d'être pleinement concentré sur une seule dynamique à la fois. Il en va de même pour les entités qui composent le système, car au niveau pratique, l'utilisation de DOM a l'avantage de faciliter l'isolation (partielle), au sein même du code, du traitement propre à chaque dynamique. Ce qui simplifie le travail futur des (mêmes ou d'autres) modélisateurs pour l'affinement de ces dynamiques.

Ensuite, DOM interconnecte ces couches MDM au travers de l'environnement, ainsi DOM est un Multi-Mono Dynamic Model (Multi-MDM). Il est donc facile de choisir et réutiliser les différentes mono-dynamiques élaborées dans d'autres modèles.

Par l'intermédiaire de ces dynamiques, DOM permet une approche multiniveaux. Comme nous le verrons dans la Section 3.3.3, DOM facilite l'organisation sur plusieurs niveaux, avec, par exemple, des agents à un niveau macroscopique et d'autres au niveau microscopique, ce qui est bien adapté dans le cas de notre modèle EDMMAS.

La Figure 3.1 illustre les principes de base de cette modélisation :

- En haut à gauche, il s'agit de la représentation de l'agent selon les dynamiques A, B et C, en tenant compte des comportements et états : chaque agent, en effet, possède un ensemble d'états et de comportements. Ces derniers peuvent modifier les états de l'agent, qui eux-mêmes peuvent influencer les comportements. Ces deux ensembles sont décomposés selon les dynamiques auxquelles participe l'agent.
- En haut à droite est représenté le découpage des dynamiques : chaque dynamique est représentée par un modèle MDM. Un agent qui participe à plusieurs dynamiques voit ses états et comportements répartis dans ces dynamiques.
- En bas, l'environnement est l'élément de couplage entre les différentes MDM.

3.3 L'expérience EDMMAS

L'expérience EDMMAS [Gangat *et al.*, 2009a, Gangat *et al.*, 2009b] est un cas concret de réutilisation de modèle dans un contexte pluridisciplinaire. Nous allons ici successivement présenter : le contexte général puis celui, plus particulier, du cadre pluridisciplinaire ; les trois

[1] **M**ono **D**ynamical **M**odel

Figure

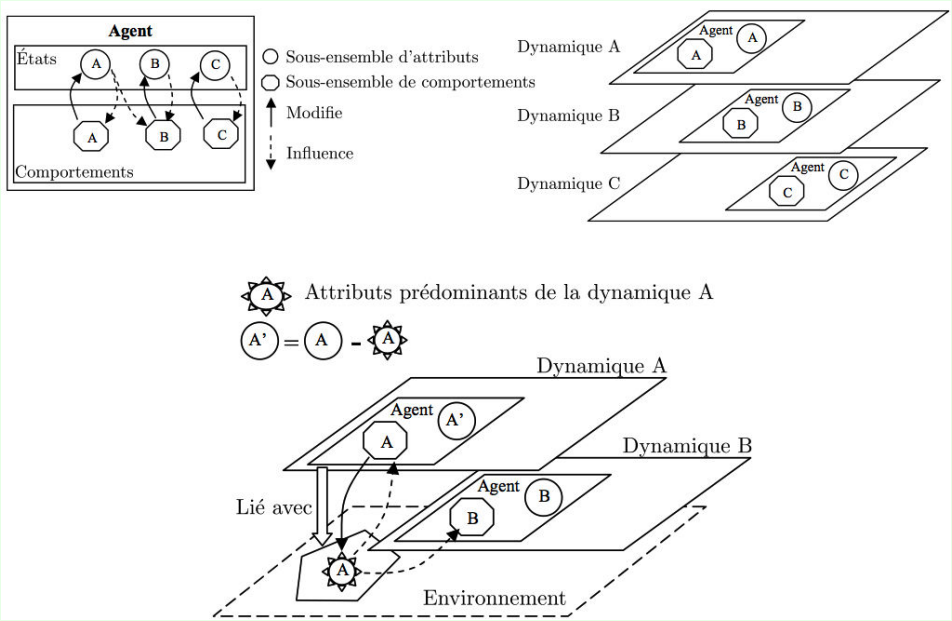


FIGURE 3.1 – La Modélisation Orientée Dynamique (extrait de [Payet *et al.*, 2006])

itérations de réutilisation de modèle qui ont conduit à l'aboutissement de ce projet ; et enfin, le modèle final obtenu.

3.3.1 Contexte général

L'île de La Réunion se situe au centre du projet GERRI[2], catalyseur pour la recherche et l'innovation. Ce programme novateur se fixe pour objectif, d'ici 2030, de faire de l'île un espace de démonstration de toutes les technologies liées au développement durable. Conduite par l'État, la Région, le Département et d'autres acteurs économiques, cette réalisation se veut une expérience-pilote pour faire de La Réunion un territoire d'expérimentation grandeur nature. Le projet GERRI se déclinait en 5 thématiques complémentaires, dont la première est la **Maîtrise D**e l'**É**nergie (MDE). La MDE est une réponse aux défis énergétiques et au souci de protection de l'environnement des pays industrialisés et en voie de développement. Elle regroupe les techniques pour diminuer la consommation d'énergie, dans un souci d'économie financière, et pour réduire l'empreinte écologique. L'amélioration de la MDE peut se réaliser à

[2]GERRI était à la fois l'acronyme de "**G**reen **E**nergy **R**evolution **R**eunion **I**sland" et de "**G**renelle de l'**E**nvironnement à La **R**éunion : **R**éussir l'**I**nnovation". Depuis le mois d'avril 2013, le groupement d'intérêt public GERRI (ainsi que l'Agence Régionale de l'Énergie Réunion) a été remplacé par la Société Publique Locale "*Énergies Réunion*". Cette agence devra, avec les mêmes objectifs, gérer 148 projets en lien avec les énergies renouvelables.

plusieurs niveaux selon la portée de son impact :

- Au niveau individuel et familial, par exemple en innovant en matière de gestion du chauffage et de la climatisation...
- Au niveau local ou communal, en améliorant les transports en commun...
- Au niveau national, par l'établissement de lois antipollution, en encourageant une agriculture moins polluante...
- Au niveau international, par l'organisation de congrès et de conférences sur le sujet...

L'île de La Réunion fournit un terrain favorable à la recherche dans ce domaine de par son isolement, pour deux raisons. Premièrement, la faible superficie de son territoire, 2 500 km^2 , favorise la recherche de nouvelles formes d'aménagement énergétique. Deuxièmement, à la différence de la métropole, on ne peut pas implanter sur notre île une production électrique d'origine nucléaire : d'abord, la tranche unitaire est surdimensionnée par rapport aux besoins de l'île ; ensuite, la maintenance d'une telle structure y est impossible.

De plus, malgré sa superficie peu étendue, La Réunion présente les mêmes problèmes énergétiques complexes que dans de nombreux pays du monde, notamment :

- La présence de nombreux acteurs de la production, autant au niveau individuel que collectif.
- Un nombre important de consommateurs répartis de façon non homogène.
- Un grand nombre de microclimats différents.
- Des sources d'énergie de différents types.

Ces paramètres font de cette île un lieu idéal pour générer recherche et innovation dans le domaine de l'utilisation rationnelle de l'énergie. La problématique abordée dans notre projet EDMMAS est l'aménagement énergétique d'un territoire en utilisant une modélisation reproductible.

3.3.2 Contexte pluridisciplinaire du projet EDMMAS

Dans le domaine des **S**ystème **M**ulti-**A**gents (SMA) les chercheurs proposent des modèles pour la MDE : par exemple, la conception d'un système domotique de gestion de l'énergie dans l'habitat reposant sur le paradigme multi-agents. Certains travaux [Boman *et al.*, 1998] utilisent quatre types d'agents représentant le confort personnel individuel, sa position dans un bâtiment, les paramètres de l'environnement et chaque pièce de l'habitat. Plus récemment, [Abras, 2009] traite de la conception d'un système domotique de gestion de l'énergie dans l'habitat sur la base d'un paradigme multi-agents. D'autres études [Kamphuis *et al.*, 2004] dépassent le niveau individuel et vont jusqu'à la gestion de l'énergie renouvelable ou encore l'évolution du prix de l'énergie.

Ainsi, la plupart des travaux sont centrés sur l'énergie aux niveaux individuel et familial,

afin d'obtenir une répartition optimale vis-à-vis d'agents consommateurs. Ces travaux s'accompagnent également d'un processus autonome de distribution d'une énergie donnée (par exemple la définition de mécanismes de négociation entre agents pour la réduction de la consommation électrique au sein d'un bâtiment). Mais peu d'études traitent de la MDE au niveau supérieur, à l'échelle d'un territoire ou d'une région.

Les SOA existantes n'offrent pas la possibilité, dans leur modélisation et dans la définition de leur support d'exécution, de mettre en œuvre des mécanismes permettant l'intégration au sein d'une même simulation de plusieurs sources d'énergie de nature différente dont les interactions les unes avec les autres sont possibles (par exemple lors de transferts d'énergies). Par ailleurs dans le cadre général de la maîtrise des énergies, l'impact de l'évolution d'une énergie particulière peut être étudié à travers des échelles temporelles et spatiales fixées. Ces échelles peuvent varier pour un autre type d'énergie.

C'est pour répondre à ce contexte que nous proposons un modèle de gestion de l'énergie que nous avons appelé : EDMMAS. L'objectif est d'aider à la prise de décision dans l'aménagement énergétique de l'île de La Réunion. L'outil issu du modèle peut supporter l'exécution simultanée de plusieurs environnements qui évoluent chacun dans une échelle temporelle qui peut leur être propre. L'idée consiste à initialiser automatiquement des agents élémentaires à partir de données extraites des cartes géoréférencées [David *et al.*, 2009], découpées en un ou plusieurs maillages de taille fixe ou variable et à associer des logiques comportementales aux agents corrélés à ces mailles. Ces cartes ne se limitent pas toutes à véhiculer de la donnée, certaines permettent aussi d'établir des scénarios de simulation, fondés sur des expertises. Ces cartes représentent les centres de production et de consommation d'énergie sur l'île, les réseaux physiques de distribution, la dynamique de l'étalement urbain de population...

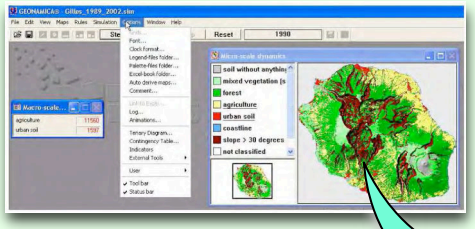
Nous sommes ici clairement dans un contexte pluridisciplinaire où, par exemple,les spécialistes en énergie, ayant une connaissance pointue de la production et consommation d'énergie, doivent pouvoir se concentrer sur la modélisation de la dynamique énergétique, sans avoir à répondre aux questions des géographes sur l'évolution démographique de la population. De plus, lorsque le domaine de l'énergie est mis en corrélation avec ceux de l'écologie et de l'économie, avec l'évolution de la population et de son étalement, nous nous trouvons confrontés à une augmentation de la quantité de données et à une complexification des flux d'informations.

Afin de pallier cette complexité, nous proposons de réutiliser des modèles partiels déjà produits dans d'autres réalisations. Pour l'élaboration de notre modèle EDMMAS, nous sommes passés par trois réutilisations successives de modèles.

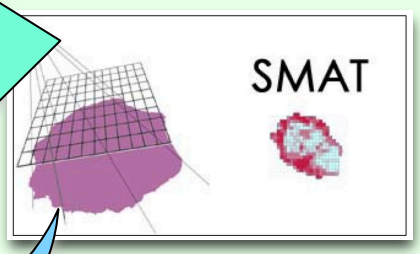
3.3.3 Trois réutilisations successives de modèle

EDMMAS est un cas de réutilisation de modèles issus d'un précédent projet de notre équipe : **D**OMINO **S**MAT (DS), lui-même s'inscrivant dans la prolongation d'un autre projet : SMAT, qui à son tour découle d'inspiration issue d'un ancien prototype (Métronamica Réunion), comme illustré sur la Figure 3.2.

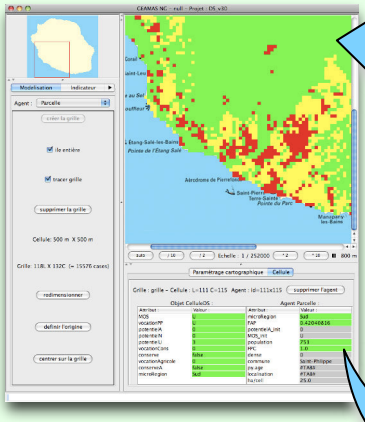
Figure



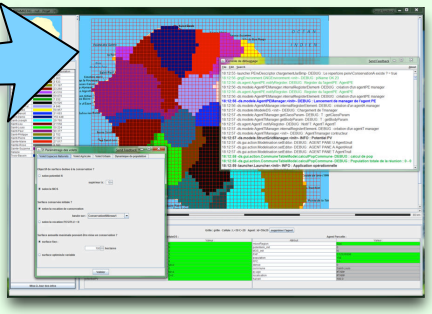
Metronamica
Simulation par automates cellulaires
de l'étalement urbain
à La Réunion



SMAT
Simulation Multi-Agents Territoires
de la dynamique de population
de La Réunion



DS
Domino-Smat
Simulation Multi-Agents
de la dynamique de population
et d'évolution du sol
de La Réunion



EDMMAS
Energy Demand Management
by Multi-Agent Simulation
Simulation Multi-Agent
de la dynamique de population,
d'évolution du sol et
d'évolution énergétique
de La Réunion

Inspiration Evolution

FIGURE 3.2 – Réutilisations successives de modèles pour arriver à EDMMAS

Métronamica Réunion : Modèle initial

Le modèle Métronamica Réunion [Lajoie et Hagen-Zanker, 2007] est un automate cellulaire développé par Gilles Lajoie, sur la plateforme Geonamica[3], pour explorer la dynamique de l'occupation des sols à La Réunion. Les auteurs reprennent un constat établi lors d'un forum-débat du Plan Urbain organisé à La Réunion en 1993 : "L'île de La Réunion connaît actuellement des transformations spectaculaires : elle passe en quelques années d'un paysage rural à une urbanisation massive, d'une économie paysanne à une économie tertiaire, d'un mode de vie campagnard à un mode de vie moderne. Le développement de la civilisation urbaine est aujourd'hui une donnée incontournable de cette situation nouvelle : la ville et l'urbanité s'emparent du territoire et le modèlent tandis que les rapports sociaux se complexifient en même temps que se transforme la société réunionnaise."

Dans les années 1950, l'île de La Réunion comptait 275 000 habitants. En 50 ans, sa population a triplé et le million d'habitants est prévu pour 2030, d'après les projections démographiques. Cet accroissement implique de prévoir les besoins en matière d'affectation et d'usage du sol à La Réunion.

Cette problématique est essentielle dans les réflexions sur l'aménagement. En effet, sur les 2 500 km^2 du territoire, seuls 1 500 km^2 peuvent être utilisés pour l'activité humaine, à cause de la géographie de La Réunion (volcan, cirques et remparts, forêts d'altitude...). Les choix d'affectation ou de préservation du sol sont traduits dans les documents réglementaires : Plans Locaux d'Urbanisme (PLU), Schéma d'Aménagement Régional (SAR), Schémas de Cohérence Territoriaux (SCOT)[4]...

Métronamica Réunion a été conçu pour étudier cette problématique. Ce modèle repose sur un mécanisme stochastique de transition des cellules qui changent d'état en fonction d'un potentiel de transition qui leur est propre. Bien que ce prototype comporte des avantages indéniables pour ouvrir le dialogue dans le cadre d'un groupe de réflexion sur la prospective territoriale, il reste limité au niveau des interactions et ne prend pas en compte les niveaux d'organisations superposés.

SMAT : Première réutilisation

Pour dépasser les limites de Métronamica Réunion, le modèle **S**ystème **M**ulti-**A**gents **T**erritoires (SMAT) a vu le jour. Ce modèle n'est pas une réutilisation directe du projet précédent, mais il y trouve son origine au niveau du modèle de domaine. Il est le fruit de la collaboration de chercheurs informaticiens de notre équipe SMART et de chercheurs géographes de l'équipe du CREGUR de l'université de La Réunion [Lajoie, 2007].

Il se focalise sur l'évolution et la répartition de la population sur le territoire réunionnais. Le but est de considérer le phénomène d'étalement urbain que l'on a pu constater grâce à Métronamica Réunion, en modélisant cette fois les mouvements de l'ensemble de la population, en prenant en compte pyramide des âges, natalité, mortalité, immigration...Cette évolution

[3] http://www.riks.nl/products/geonamica (Page consultée le 1er février 2013).

[4] Plus d'informations sur : www.developpement-durable.gouv.fr/IMG/pdf/F10_MEDDTL_Fiches_Guide_Ev_Env_Doc_Urba_BD_nov2011.pdf (Page consultée le 1er février 2013).

est constatée en fonction de critères démographiques et de critères liés à l'attractivité du sol (par exemple, les zones situées sur le littoral de l'île sont plus favorables à l'urbanisation que les zones montagneuses).

Les agents de SMAT sont spatialisés. Il s'agit d'agents **Parcelles** qui abritent une population dont on connaît initialement, via un ensemble de cartes d'initialisations, la pyramide des âges, les taux de natalité et de mortalité ainsi que le comportement en termes de mobilité résidentielle à l'échelle de la commune. Ces mouvements de population résultent de l'évolution de la population et du pouvoir d'attractivité de la parcelle. Le prototype SMAT a été implémenté sur notre plateforme **GEAMAS N**ew-**G**eneration (GEAMAS-NG) utilisant l'approche DOM (voir Section 3.2).

L'objectif de SMAT a par la suite rejoint celui d'une Action Thématique Programmée (ATP) du CIRAD intitulée DOMINO[5].

DS : Deuxième réutilisation

Le modèle DS [David *et al.*, 2007] est la contraction de deux acronymes : **D**OMINO et **S**MAT. Ce second développement mené par notre équipe est le fruit d'une collaboration de trois années avec l'ATP DOMINO pilotée par le CIRAD, en collaboration avec l'université de La Réunion, le comité de pilotage de la canne, l'association pour la promotion en milieu rural et la chambre d'agriculture de La Réunion. L'ATP dans lequel il s'inscrit vise à développer des outils d'aide à la décision sur les choix de règles d'affectation des terres. Notre équipe, dans la continuation du projet SMAT, est ainsi intervenue pour apporter son appui au développement de ces outils et formuler les concepts informatiques nécessaires à l'implémentation des modèles.

DS est un modèle de SOA servant à explorer différents scénarios d'utilisation des terres et de planifications de conservation pour l'île de La Réunion. Il simule, sur l'ensemble de l'île, les interactions entre les trois types d'utilisation du sol (espaces naturels, agricoles et urbains), dans le but de permettre l'observation des évolutions induites. Il est constitué d'un environnement composé de **cellules** et de plusieurs agents (voir Figure 3.3). L'organisation hiérarchique de ces agents se décompose comme suit :

- **Parcelle :** Le territoire est découpé en unités élémentaires spatiales (appelées parcelles), découlant d'une coupe en grille. Chaque unité est un agent d'échelle micro **Parcelle** géolocalisé sur une **cellule** de l'environnement.
- **Microrégion :** Les agents **Parcelles** sont regroupés selon les quatre microrégions Nord, Sud, Est et Ouest (conformément à la hiérarchie administrative de l'île) et sont gérés par des agents **Microrégions**. Ces dernières agissent comme intermédiaires entre l'agent **Région** et les agents **Parcelles**, gérant la répartition de population et interagissant avec les agents superviseurs qui seront décrits ci-après.

[5] **D**émarche **O**bjet **M**ultisite pour l'étude des **I**nteractions entre **N**iveaux d'**O**rganisation

- **Région :** Les quatre **Microrégions** composent l'agent **Région**. Ce dernier permet l'intégration de la population migrante interne à l'île (depuis les **Parcelles**), mais aussi externe par une redistribution selon les **Microrégions**.

Parallèlement à cette hiérarchie, ce modèle contient aussi trois agents superviseurs d'échelle macro qui s'occupent de l'affection des terres en agissant directement sur les cellules de l'environnement : les agents superviseurs **Agricole**, **Naturel** et **Urbain**, que nous détaillerons plus loin.

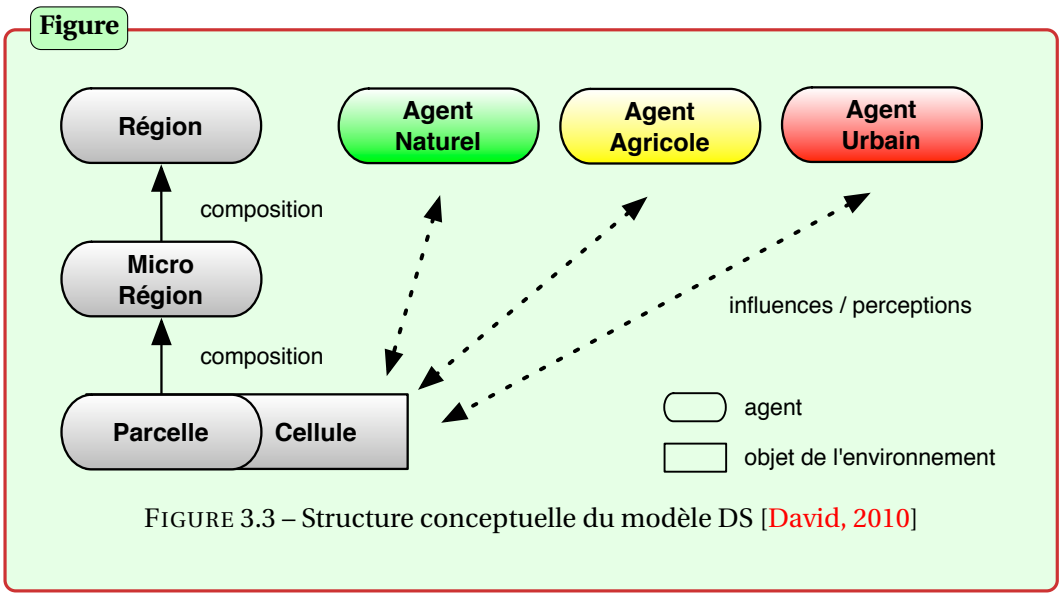


FIGURE 3.3 – Structure conceptuelle du modèle DS [David, 2010]

Ce modèle est axé sur deux dynamiques comportementales couplées grâce à la méthodologie DOM :

1. Une première dynamique, dite d'évolution de la population, repose sur l'évolution des agents **Parcelles** qui couvrent, tous ensemble, la totalité du territoire.
2. Une seconde dynamique, dite d'évolution du Mode d'Occupation des Sols (MOS), présente à l'échelle globale de l'île, repose sur les agents superviseurs. Elle implique donc les changements d'utilisation des terres (naturelles, agricoles ou urbaines).

Chaque dynamique possède ses propres formes d'interactions avec l'environnement, et plus particulièrement au niveau des **cellules**. Les deux dynamiques précitées coexistent en inter-influence l'une avec l'autre. Le couplage des dynamiques d'évolution de la population et de l'évolution du MOS a pour principal but de loger la population dans les espaces urbains en faisant passer, si nécessaire, des cellules en MOS Urbain.

Dynamique de Population : La première et plus importante dynamique concerne l'évolution de la population. Cette dynamique est une réutilisation directe de celle modélisée au sein

de SMAT. Le territoire est découpé en un grand nombre d'agents réactifs (les **Parcelles**), sur une granularité modulable en fonction du degré de finesse recherché. Cette dynamique se situe à un niveau micro.

Chacun des agents **Parcelle** possède une population initiale. Cette population évolue tous les ans selon :

- Un calcul des migrations, naissances, et morts de l'année.
- Une mise à jour de la pyramide des âges de la **Parcelle**.
- Une mise à jour de la population totale de la **Parcelle**.

Cette nouvelle population totale (ainsi que la population migrante) de chaque Parcelle est communiquée successivement aux agents **Microrégions** et à l'agent **Région**, qui se chargent de la répartition.

Dynamique du MOS : La seconde dynamique du cœur du système de DS est celle du mode d'occupation du sol. Elle a pour moteurs les trois agents superviseurs (voir Figure 3.3), qui ont une vision globale de l'environnement ; elle est à l'échelle macro (contrairement à la dynamique précédente). Ces agents sont :

- **L'agent Naturel**, dont l'objectif est d'obtenir une surface d'espaces naturels à conserver sur l'île en mettant des cellules en conservation (prioritairement les cellules de plus fort potentiel naturel).
- **L'agent Agricole**, qui vise à atteindre une surface de terre agricole donnée sur l'île dans son ensemble en faisant passer des cellules (prioritairement les cellules de plus fort potentiel agricole) en MOS Agricole.
- **L'agent Urbain**, dont le but est d'assurer l'étalement urbain nécessaire à l'installation de la population selon certaines règles.

Scénarios : DS peut être initialisé avec divers scénarios pour la politique d'utilisation des terres et de l'évolution de la population au travers d'une interface utilisateur.

À la fin de chaque simulation, DS donne différents indicateurs. Par exemple, la Figure 3.4 est une capture d'écran de l'application dont le centre est une carte, vue principale, montrant l'efficacité d'utilisation des terres de chaque parcelle de l'île à chaque étape de simulation. DS fournit aussi des graphiques présentant l'évolution globale du type d'utilisation des terres naturelle, agricole ou urbaine.

EDMMAS : Troisième réutilisation

Pour répondre aux besoins du contexte GERRI et dans le même temps éprouver le potentiel de réutilisabilité conféré par DOM, EDMMAS a vu le jour. Dans notre précédent modèle DS, les agents sont localisés dans l'espace, cette base nous permet donc de prendre en compte la localisation de la production d'énergie, ainsi que la consommation de la population de

Figure

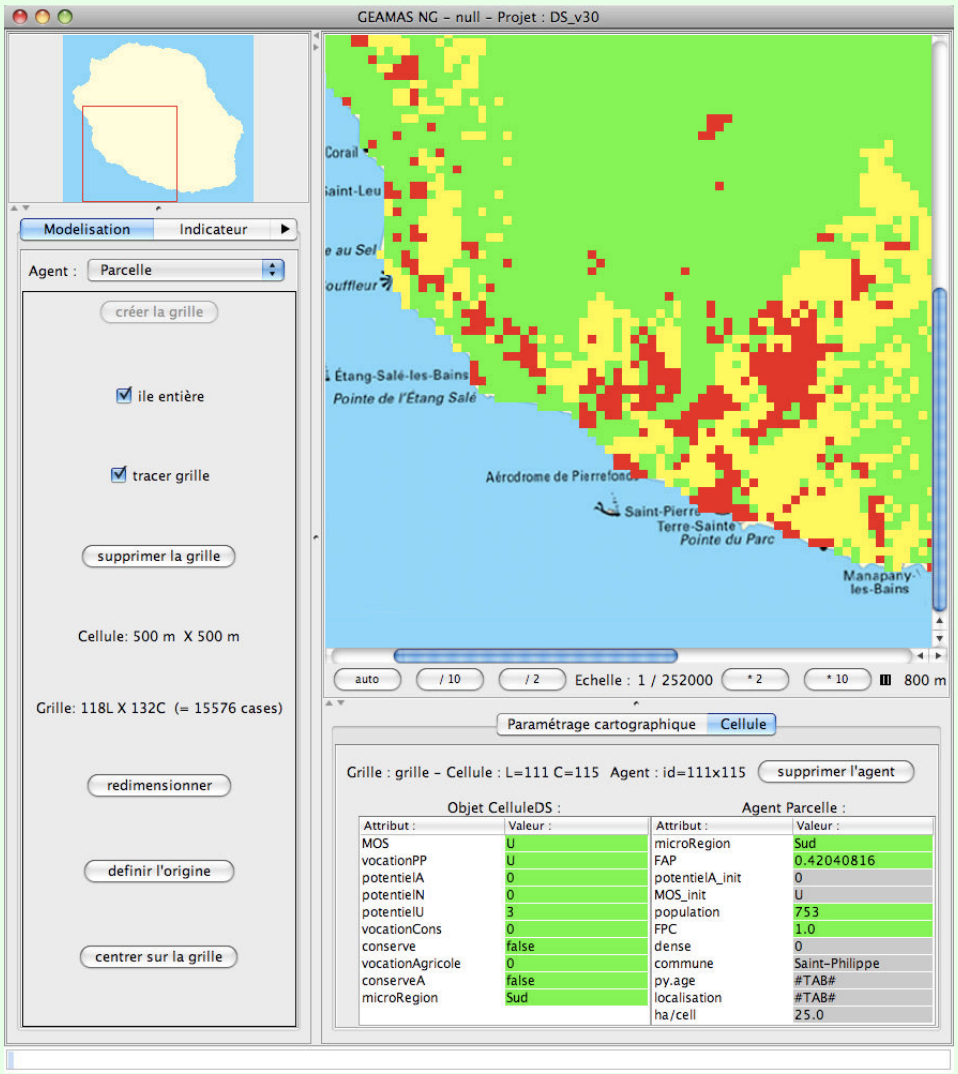


FIGURE 3.4 – La principale interface utilisateur de DS (extrait de [David, 2010])

chaque parcelle. Nous avons donc naturellement envisagé d'utiliser cette base pour générer des simulations basées sur l'espace géographique en considérant les besoins en distribution d'énergie dans l'île à court et moyen termes. Cela nous a conduits à la réalisation du modèle présenté en détail dans la section suivante.

3.3.4 Le modèle EDMMAS

Ce modèle est une réutilisation du précédent modèle DS, validé et utilisé par les experts. Nous en avons repris les bases auquel nous avons adjoint une nouvelle dynamique où les agents doivent gérer le type et la quantité d'énergie produite par les centrales et les unités de production individuelles photovoltaïques.

Dynamique de l'énergie

La population ne consomme pas de la même manière selon sa position géographique, son activité, la météo, *etc.* Aussi, dans notre modèle, cette population donne lieu à des agents **Parcelles** consommateurs. Les principales informations que ces derniers doivent gérer sont :

- L'évolution de leur nombre d'habitants dans le temps.
- Le type d'activité lié au terrain (naturel, agricole ou urbain).
- Le type d'activité lié à la zone (pour particuliers et industrielle).
- Le niveau de satisfaction de l'agent selon certains critères : confort thermique, coût, empreinte écologique...

Utiliser DS comme base nous permet de nous affranchir des deux premiers points, car il gère déjà l'évolution de la population et celle du MOS. L'approche choisie pour ce projet a donc été d'apporter une nouvelle dynamique à DS pour modéliser :

- La production et la distribution d'énergie.
- La gestion d'énergie multiniveaux.

Production et Distribution : Le premier volet du projet consiste à modéliser par la dynamique de l'énergie (ou d'évolution énergétique) : la production, sa distribution et sa consommation. Au niveau de la production, des agents cognitifs forment le cœur du système. Ils gèrent les informations suivantes :

- Le type de production d'énergie : éolien, fuel, charbon, hydroélectrique, photovoltaïque...
- La capacité de production suivant la période. La Réunion possède des centrales mixtes qui fonctionnent au charbon et à la bagasse[6].
- L'impact des contraintes extérieures. Ainsi, la météo n'a que très peu d'influence sur les moteurs à gasoil, mais beaucoup plus sur le solaire et l'éolien.
- Le coût de la production (également appelé coût opératoire) : les moyens nécessaires pour maintenir en production.

[6] La bagasse est le résidu fibreux de la canne à sucre, passée par le moulin pour en tirer le suc. En 2006, la culture de canne à sucre occupe à elle seule près de 55 % du territoire agricole réunionnais. Ce résidu est une énergie renouvelable disponible seulement pendant la période de récolte de la canne à sucre.

- Le contenu CO_2[7] des différentes filières de production d'électricité.
- La modularité de la production : certaines centrales peuvent être activées partiellement ; pour d'autres, il est impossible de régler le débit.
- La production locale d'énergie solaire des usagers.

Au niveau de la distribution, des agents distributeurs prennent en compte les contraintes suivantes :

- La gestion des flux d'énergie : il faut tenir compte de la perte d'énergie (en pourcentage) par rapport au type de la ligne (haute tension, moyenne tension, basse tension et les différentes classes de câbles) et des transformateurs.
- La distribution sur petites distances : il est nécessaire de favoriser la distribution de l'énergie sur de courtes distances, donc de conseiller la mise en marche de certaines centrales aux dépens d'autres.

Au niveau de la consommation, des agents réactifs consommateurs (la population de La Réunion) sont présents. Le système tient compte de certains points particuliers :

- La position des transformateurs haute-basse tension entre les producteurs et les maisons des particuliers.
- La différenciation des types de consommation (naturel, agricole ou urbain).

Dans cette modélisation (voir la Figure 3.5), les agents ont pour mission de distribuer l'énergie le mieux possible en se basant sur les différents producteurs et consommateurs d'énergie, les transformateurs et les lignes à haute tension.

La gestion d'énergie multiniveaux : Le second volet du projet repose sur les travaux de Shadi Abras [Abras, 2009] qui traitent de la conception d'un système domotique de gestion de l'énergie dans l'habitat.

Il introduit le modèle générique de représentation de notions utilisées pour la conception de systèmes multi-agents de l'énergie dans l'habitat. La modélisation proposée repose sur la notion de service. Celui-ci peut être **interruptible**, **décalable** ou **modifiable** et appartient à l'une des deux catégories : **permanent** ou **temporaire**, selon sa nature.

Cette notion de service est utilisée ensuite dans la caractérisation des agents du SMA proposé, appelé MAHAS (**M**ulti-**A**gents **H**ome **A**utomation **S**ystem). Son originalité réside dans la gestion d'horizons temporels multiniveaux avec un cycle de pilotage en deux phases :

- **Un mécanisme réactif** qui contrôle le système à un horizon temporel court de l'ordre de la minute, reposant notamment sur une fonction centrale d'évaluation de niveau de satisfaction vis-à-vis des équipements. Abras propose un mécanisme assez élaboré

[7] Les chiffres des émissions de CO_2 par kWh sont issus d'une démarche scientifique pour analyser les impacts environnementaux d'un produit, d'un procédé, d'une filière.

Figure

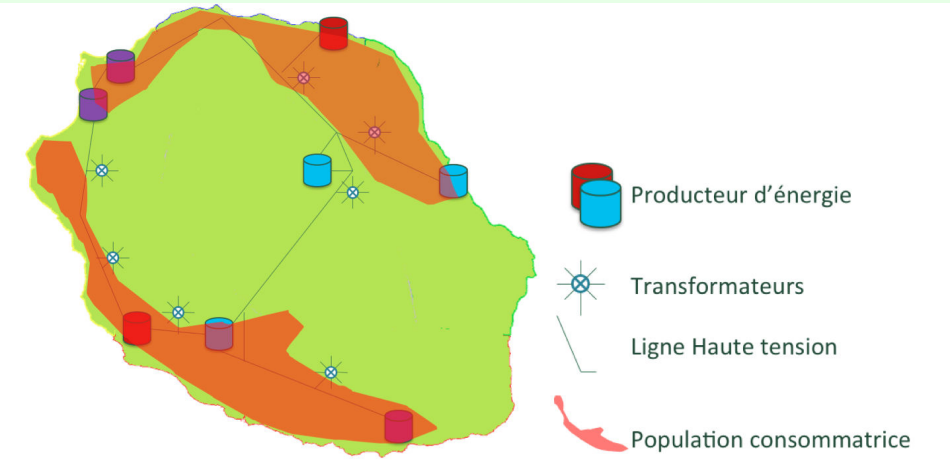


FIGURE 3.5 – Production et Distribution de l'énergie. Adapté des documents *EDF*

prenant en compte l'ensemble des aspects de la notion de service. Un protocole de négociation s'appuyant sur le niveau de satisfaction des agents pour négocier de la ressource énergétique a été défini.

- **Un mécanisme d'anticipation** capable de calculer un plan global de consommation d'énergie d'un habitat fonctionnant sur un horizon temporel de l'ordre de l'heure. L'auteur propose une méthode de résolution générale pour ce mécanisme de calcul de plans d'affectation de l'énergie en prenant en compte les prévisions de consommation/production des différents équipements.

La solution proposée définit alors un système adapté à différentes échelles de temps : le système réactif ajuste le plan global au système réel en réagissant aux événements imprévus.

Dans EDMMAS, l'intégration de cette contribution nous amènerait à utiliser les mêmes concepts à des niveaux supérieurs (voir la Figure 3.5). Ainsi, au lieu d'appliquer ces concepts au niveau d'un bâtiment et de ses équipements (comme dans la thèse de Shadi Abras), ils seront généralisés au niveau des microrégions (en y ajoutant une microrégion Centre, étant donné que sa population, moins dense, nécessite moins de demande énergétique). Dans cette partie du projet, nous sortons donc du cadre de la simulation d'un système complexe *réel*, pour arriver à une simulation d'un système complexe non réel, ce qui pourrait être réalisé dans le futur. En effet, une telle gestion implique la présence d'un nouveau type de transformateur "intelligent".

Figure

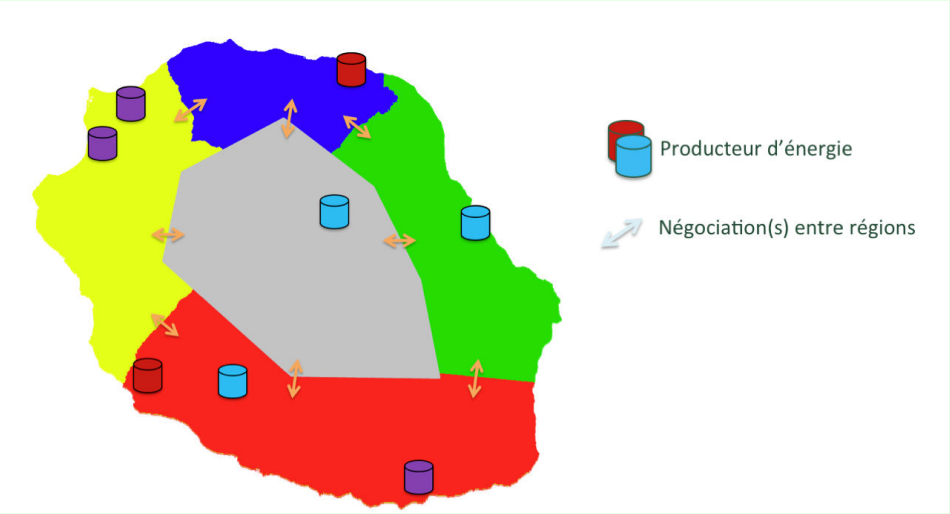


FIGURE 3.6 – Gestion d'énergie multiniveaux. Adapté des travaux de Abras [Abras, 2009]

Figure

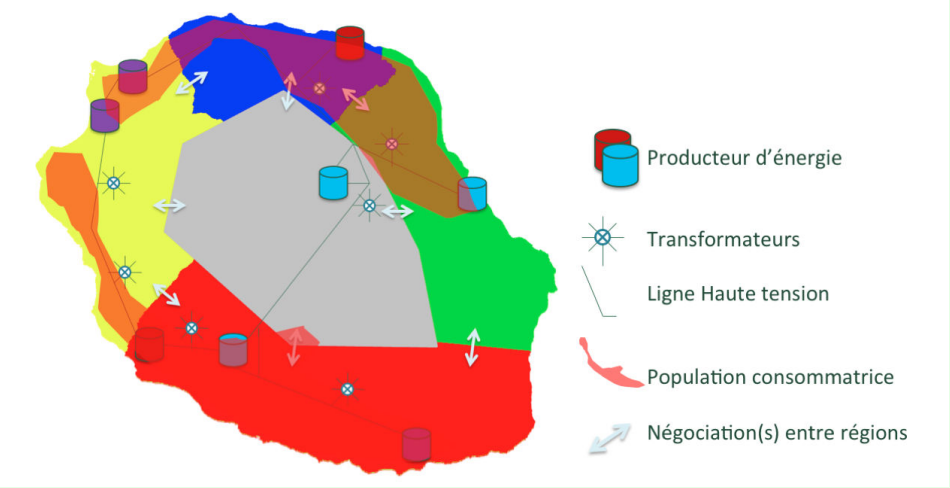


FIGURE 3.7 – Production et Distribution de l'énergie avec négociations entre régions

Synthèse des deux volets : Les deux volets du projet EDMMAS (voir la Figure 3.7) permettront de mieux produire et distribuer l'énergie électrique. Par cette distribution plus intelligente de l'énergie, nous en économiserons une grande quantité. En effet, des pertes électriques naturelles apparaissent dans les réseaux de transport et de distribution, liées au type de la ligne, et représentent un coût. Par exemple, selon l'*Observatoire Énergie Réunion*[8], la production électrique en 2006 a été de 2 365 GWh pour une consommation de 2 152 GWh. La différence entre ces deux valeurs s'explique par ces pertes électriques naturelles dans les réseaux. Ajouter cette donnée dans le système permet donc de prévoir, par exemple, de nouvelles lignes, de nouvelles installations ainsi que leur localisation optimale et leur dimensionnement, de manière à limiter les coûts, aux niveaux économique et écologique.

3.4 Le prototype EDMMAS

Inspiré du modèle décrit précédemment, le premier volet de EDMMAS a été implémenté sur la plateforme GEAMAS-NG.

3.4.1 Implémentation de la Dynamique de l'énergie

Figure

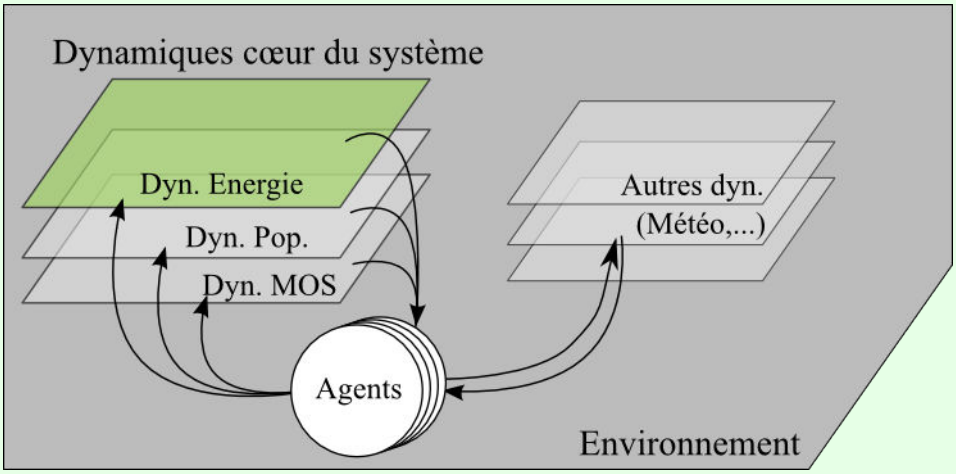


FIGURE 3.8 – Les dynamiques de EDMMAS

La Figure 3.8 montre sur la gauche les dynamiques présentes dans EDMMAS et sur la droite celles qui pourraient être ajoutées par la suite. La dynamique de l'énergie (en vert) a donc été créée, complétant ainsi le cœur du précédent système DS. Cette dynamique, sur laquelle nous avons consacré la plupart de nos efforts dans EDMMAS, a pour moteur des

[8] www.arer.org/IMG/pdf/175_OER-Bilan-energetique-grand-public.pdf (Document consulté le 1er février 2013).

agents cognitifs localisés sur les centres de production d'énergie.

Implémentation de la carte

Le niveau d'ensoleillement est aussi intégré au moyen d'une carte (voir Figure 3.9) pour permettre aux agents **Parcelles** de participer à la dynamique de production d'électricité. Étant donné le niveau de précision souhaité, nous avons utilisé des données de cumul moyen annuel ; mais il est possible d'utiliser un ensemble de cartes pour tenir compte, par exemple, de l'évolution du climat de manière mensuelle.

Figure

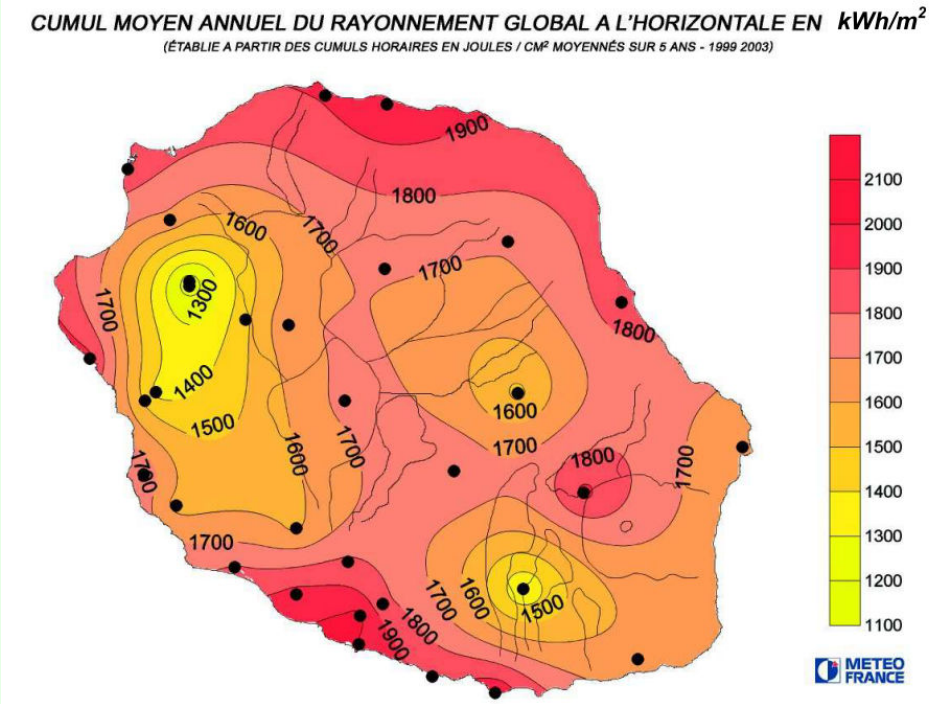


FIGURE 3.9 – Carte du cumul moyen annuel du rayonnement global à l'horizontale à La Réunion (*Source : Météo-France*)

Pour faciliter l'initialisation des dynamiques et des agents, dans un précédent article [David *et al.*, 2009], nous avons présenté le langage e**X**tensible **E**diting **L**anguage **O**f **C**onfiguration (XELOC). Il s'agit d'un langage hybride basé sur le langage XML et enrichi par une sémantique de langage script. Sa nature extensible et accessible en fait un support générique utilisable dans de nombreux contextes d'implémentation. Au sein de notre plateforme de simulation GEAMAS-NG, nous avons développé un interpréteur XELOC . Parmi les avantages présentés dans [David, 2010], l'un d'entre eux est d'offrir une technique d'initialisation utilisant des cartes sémantiques fournies par des thématiciens, comme celle de la Figure 3.9.

Par exemple, mettons en place les informations nécessaires pour initialiser le potentiel photovoltaïque via la carte de la Figure 3.9, composée de 11 couleurs différentes :

```
<?xml version="1.0" encoding="UTF-8"?>
<MODULE type="PEnv" name="potentielPV" >
    <version num="0.01ds" statut="stable" date="04/03/2009" />
    <author> Yassine Gangat </author>
    <info> Carte des potentiels en Photo Voltaique </info>
    <entry img="potentielPV.bmp" imgToReal="253250" pixelToReal="63.5" >
        <operation name="Potentiel Photo Voltaique"
            class="ds.ressource.map.potentielPV.PPVOperator"
            legend="Détermine le potentiel photovoltaïque de la zone interrogée
                selon le degré d'ensoleillement" />
        <legend class="ds.ressource.map.potentielPV.LegendPPV" >
<!-- Ces entrées définissent en fait la légende de la carte.
Par exemple, la composante couleur avec Rouge=252, Vert=245 et
Bleu=36 correspond à la zone " 3 ". Dans notre exemple, cela
correspond à du jaune. -->
            <entry name="11" r="245" v="9"   b="64"  />
            <entry name="10" r="250" v="10"  b="64"  />
            <entry name="9"  r="250" v="30"  b="77"  />
            <entry name="8"  r="250" v="77"  b="100" />
            <entry name="7"  r="249" v="128" b="120" />
            <entry name="6"  r="250" v="154" b="87"  />
            <entry name="5"  r="250" v="180" b="62"  />
            <entry name="4"  r="250" v="211" b="42"  />
            <entry name="3"  r="250" v="245" b="36"  />
            <entry name="2"  r="233" v="255" b="150" />
            <entry name="1"  r="227" v="255" b="25"  />
        </legend>
    </entry>
</MODULE>
```

À partir de cette description XML et du fichier image correspondant, l'interpréteur doit être aussi configuré pour utiliser ces informations et les restituer correctement.

Implémentation des agents consommateurs

Nous allons ici ouvrir une petite parenthèse technique sur l'un des concepts génie-logiciel clés de notre plateforme GEAMAS-NG : celui des objets **MultiFaces**. Il est en effet utilisé dans l'implémentation de EDMMAS, en particulier au niveau des agents **Parcelles** pour supporter les modifications nécessaires à la prise en compte de la consommation d'énergie (en rapport avec leur population, leur type de terrain et de consommation). L'éclairage de ce concept illustre en partie comment notre plateforme offre un support à l'approche de modélisation

DOM.

En **P**rogrammation **O**rientée **O**bjet (POO), un objet est associé à une unique instance de classe. Le concept d'objet MultiFace propose une extension de cette définition. Un objet MultiFace peut être associé à plusieurs instances de classe simultanément. Chaque instance constitue une "face" particulière de l'objet MultiFace. Les méthodes invoquées sur un objet MultiFace sont dirigées sur l'instance correspondant à la face où l'objet MultiFace est positionné.

Ce concept offre une solution technique au contexte de multienvironnement sur lequel est fondée notre plateforme GEAMAS-NG : un corps MultiFace pointe sur plusieurs instances de `Body` (`package gng.gfc.agent`) qui sont autant d'extensions de l'agent dans des environnements distincts. Cela permet de signifier à l'agent qu'il possède un corps dans un environnement et d'indiquer celui dans lequel il doit agir lors de ses prises de décision.

Ainsi, les agents multifaces possèdent plusieurs "faces". À l'image d'un dé polyédrique, à chaque position de l'agent sur une "face" précise, l'information perçue par l'agent est différente. Pour permettre à un agent de participer à une dynamique, il faut donc lui attribuer une face. Cette attribution se fait par une méthode `bodyForAgent(String agentFamily, String agentId, ReadableMultiValue bodyParam)` qui associe à un agent, identifié par son `agentId`, un corps au sein de cette dynamique, créé à partir des paramètres d'instanciation `bodyParam`[9].

Implémentation des agents producteurs et distributeurs

Les agents producteurs gèrent les informations de production d'énergie décrites dans la Section 3.3.4 et tiennent compte des contraintes énoncées comme la perte d'énergie par rapport au type de la ligne et favorisent la distribution sur petites distances. Ils ont été placés sur l'île en suivant leurs positions sur la Figure 3.10. Cette création, plus complexe, a été présentée dans un rapport technique [Gangat, 2009].

3.5 Évaluation et Conclusion

La réutilisation est une demande concrète de la part des thématiciens. Le modèle EDMMAS est issu de trois réutilisations successives, dont deux utilisant DOM. Ces réutilisations ont permis autant un gain de temps qu'une amélioration progressive du modèle original, grâce aux dynamiques de DOM.

La synergie obtenue par ces différentes dynamiques produit des résultats pertinents. Les agents producteurs et consommateurs, géolocalisés, peuvent aussi prendre en compte dans leurs transactions une donnée souvent négligée dans les autres modèles existants au niveau de la distribution : la gestion des flux d'énergie.

Outre les résultats de la simulation et l'apport de EDMMAS à la thématique de la **M**aîtrise **D**e l'**É**nergie, l'expérience de cette mise en oeuvre nous montre que plus nous étendons les modèles à d'autres dynamiques, plus les agents deviennent considérablement complexes.

[9] Plus d'informations technique sont disponibles sur le site de l'équipe, dans la rubrique de GEAMAS-NG, https://projets-lim.univ-reunion.fr/projects/sma/wiki.

Figure

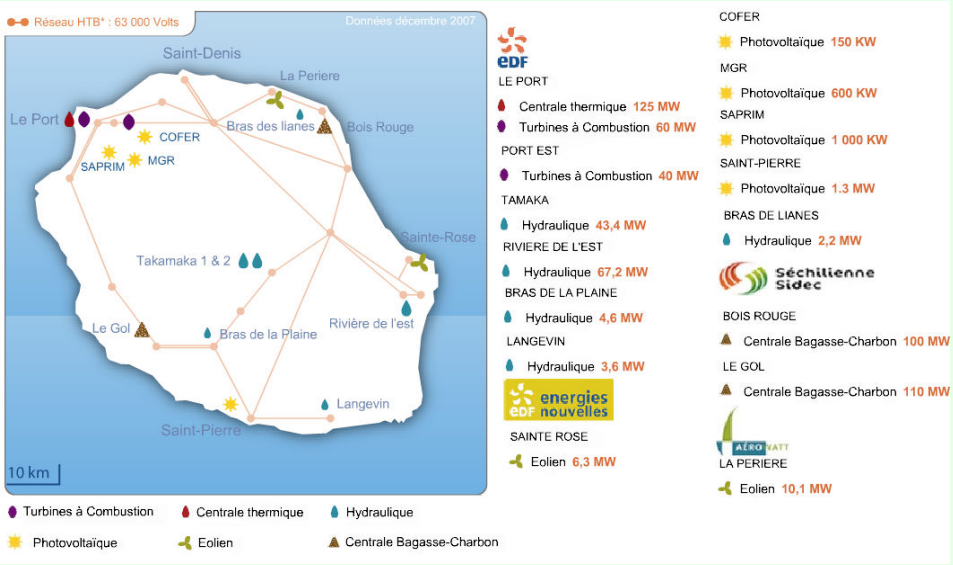


FIGURE 3.10 – Production d'énergie à La Réunion (*Source : EDF*)

En effet, si ce projet a prouvé que l'utilisation de DOM est un bon choix, car il facilite réellement le réemploi d' une "ancienne" simulation pour en construire une nouvelle, il s'avère néanmoins que le prix à payer est une montée en complexité forte au niveau de l'unité agent. Ce contrecoup fut si fort que nous en sommes arrivé à l'incapacité de pouvoir entreprendre l'implémentation du deuxième volet sur la seule base méthodologique de DOM.

L'approche DOM est efficace au niveau conceptuel et particulièrement pour la gestion des environnements. Elle répond ainsi parfaitement aux problèmes liés aux dynamiques et à la gestion de multienvironnement, au niveau conceptuel.

Les limites décelées dans DOM sont à un autre niveau : cette méthodologie n'offre aucun support à la bonne gestion de l'accumulation des traits comportementaux qu'elle induit progressivement au niveau des agents. Le constat est le suivant : plus le niveau de complexité augmente, plus le nombre de dynamiques à appréhender devient important et plus les déclinements comportementaux des agents sont nombreux. La réutilisation des modèles finit par redevenir compliquée. Cette nouvelle complexité découle de la démultiplication des comportements des agents et crée un décalage conséquent entre le modèle opérationnel et le modèle conceptuel (voir Section 2.2.1). Ce décalage nécessitera ensuite un travail important lors de l'implémentation des agents, notamment pour leurs comportements impliquant plusieurs dynamiques. Plus les agents sont corrélés à des dynamiques, plus ils deviennent délicats à appréhender sur le plan implémentatoire en induisant une plus grande complexité dans les

modèles opérationnel et informatique.

Notre deuxième contribution dans cette thèse est de limiter ce problème et rendre ainsi l'approche plus pertinente pour les cycles successifs de réutilisation de modèle. Il s'agit de la proposition d'un modèle d'architecture interne d'agent facilitant la gestion du multicomportement.

4 Cadre Théorique : Depuis MVC dans le monde Objet à aMVC dans le monde Agent

Plan du chapitre

4.1 Introduction

Nous avons vu dans le chapitre précédent que, pour la gestion des dynamiques et des environnements, nous avons une grande aisance pour la réutilisation grâce à la **M**odélisation **O**rientée **D**ynamique (DOM). Le contrecoup de cette démarche est une augmentation de la complexité des agents, étant donné qu'ils ont un corps (ou une face) pour chaque dynamique. La faiblesse que nous pouvons ressentir se trouve actuellement au niveau du moyen d'expression de l'agent lui-même. La réutilisation n'est que partiellement résolue. Notre objectif est de faciliter plus encore cette réutilisation et d'aller plus loin que les modèles, en permettant de réutiliser non seulement les agents, mais également les comportements qui les constituent.

Dans la Section 2.4, nous avons vu que pour simplifier la réutilisation, il faut utiliser des composants génériques qui pourraient être facilement ajoutés et extraits. Et pour que ces composants soient "composables", ils doivent être conçus dans cette optique. Aussi, la réutilisation doit être pensée dès le départ, au niveau même de l'architecture de l'agent.

La majorité des travaux sur les architectures agents ne proposent pas une décomposition fine de l'agent. La littérature offre plusieurs définitions d'architecture agent, mais celle que nous adopterons est celle proposée par Maes [Maes, 1991] :

Définition 10

Architecture agent : Ensemble des techniques et algorithmes visant à proposer une méthodologie pour construire un agent. Elle spécifie comment l'agent peut être décomposé en un ensemble de modules et comment celles-ci peuvent interagir. Cette spécification permet de répondre à la question : "Comment, à partir des données de ses capteurs et de son état interne courant, l'agent détermine ses actions et son état interne futur ?"

Bien que le problème initial de réutilisation soit à un haut niveau, nous devons descendre au niveau architectural pour proposer une solution efficace. Dans ce chapitre, nous faisons une parenthèse pour présenter les Design Patterns dans le monde Objet et dans le monde Agent, puis le modèle **M**odèle **V**ue **C**ontrôleur (MVC). Enfin nous proposons l'adaptation de ce pattern dans le monde Agent. Cette adaptation offre une décomposition architecturale de l'agent très fine, car au lieu de proposer des sur-couches à l'agent, elle le décompose en sous-modèles.

4.2 Les Design Patterns

Alexander, Ishikawa et Silverstein ont introduit la notion de *patterns* dans la conception architecturale. Ils décrivent dans [Alexander *et al.*, 1977] un "pattern" comme suit : "*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*"

Un "pattern", ou "patron", est donc la description d'un problème qui se produit très fréquemment dans notre environnement, puis de l'architecture associée à l'expression d'une solution formulée de façon à pouvoir l'utiliser plusieurs fois sans jamais l'adapter deux fois de la même manière. Il s'agit donc de décrire une **solution** générale pour un **problème** récurrent dans un **contexte** donné. Ces trois mots en gras résument la notion de "pattern", qui a été adaptée à l'informatique sous la forme de "Design Pattern" en 1987 [Smith, 1987]. Cette notion a pour but de capitaliser la connaissance née du savoir-faire d'experts, pour la rendre disponible ensuite à des non-spécialistes. Ce qui sert aussi de garde-fou contre certaines erreurs classiques, par exemple le fait de privilégier la composition et la délégation sur l'héritage...

4.2.1 Design Patterns du monde Objet

Définition

Kent Beck et Ward Cunningham dans un article [Beck et Cunningham, 1987] présentent une adaptation de Langage de Pattern à la **P**rogrammation **O**rientée **O**bjet (POO) . Ils proposent ainsi cinq modèles utilisés, avec succès, pour concevoir des interfaces utilisateurs à base de fenêtres. Depuis, de nombreux articles et présentations relatifs aux Design Patterns ont été publiés par divers chercheurs du monde orienté objet.

Ce concept a gagné en popularité en 1994, après l'ouvrage collectif [Gamma *et al.*, 1995] de Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, connus sous le pseudonyme "Gang of Four" ou simplement "GoF". Les modèles de conception encouragent la réutilisation et peuvent servir comme des "blocs de construction" pour les logiciels complexes. Le but était de rendre disponibles et explicites des pratiques de bonne conception pour capturer et diffuser le savoir-faire des concepteurs, en nommant et explicitant une structure de haut niveau avec un vocabulaire commun aux développeurs et concepteurs.

La raison de cet engouement ? Les systèmes devenant de plus en plus complexes avec un nombre de classes et d'instances croissant, ce concept permet une plus large réutilisation de classes par l'intermédiaire d'ensembles de classes collaboratrices.

Un Design Pattern est un "patron de conception"[1] : selon "GoF", c'est la description d'objets et de classes communicants. Ils permettent de résoudre un problème général de conception dans un contexte particulier. Chacun d'eux fait que certains aspects de la structure du système varient indépendamment des autres, ce qui rend le système plus robuste à un type particulier de changement.

Voici le vocabulaire utilisé pour la description d'un Design Pattern[2] :

- **Nom* :** Mot simple ou petite phrase servant référence au pattern, évoquant l'esprit dans lequel il a été conçu. Cette précision facilite la recherche et l'association de patterns.
- **Problème* :** Définition du problème, avec les intentions ou les sorties souhaitées, et les symptômes montrant l'existence de ce problème.

[1] En français, on l'appelle aussi forme/modèle/motif/schéma de conception.
[2] Les éléments suivis d'une étoile (*) sont ceux qui sont obligatoires dans la majorité des cas.

- **Contexte initial :** Préconditions à l'existence du problème, appelées parfois la *situation*. En cas de forces en conflit, la solution passe par les contextes.
- **Forces :** Description des forces ou des contraintes, ainsi que leurs corrélations. Parfois, les forces sont opposées ou contradictoires, par exemple être minutieux et rigoureux est en conflit avec les contraintes de temps et d'argent.
- **Solution* :** Instructions, parfois avec des variantes. On peut leur ajouter images, diagrammes (souvent de type OMT – Object Modeling Technique) et autres médias pour plus de clarté.
- **Exemples :** Exemples d'applications et de solutions, analogies, exemples visuels,... La proposition d'exemples d'utilisations "connus" est particulièrement utile pour la compréhension du contexte.
- **Contexte résultant (Conséquences)* :** Résultats après l'application du pattern, incluant les postconditions et les effets de bord. Le cas échéant, on mentionne les nouveaux problèmes surgis de la résolution du problème original.
- **Rationnel :** Le rationnel (ou la logique) qualifie le processus de réflexion dans l'utilisation de ce pattern. On y mentionne aussi une explication des raisons de son bon fonctionnement, et comment les conflits des forces et contraintes aboutissent à la sortie souhaitée.
- **Patterns reliés :** Ce sont les différences et relations avec les autres patterns, si possible les prédécesseurs, antécédents ou alternatives pour résoudre un problème similaire.

On utilise parfois d'autres termes comme **Participants** qui désigne les objets qui participent au Design Pattern et leurs responsabilités, **Collaborations** qui explique comment les participants collaborent pour accomplir leurs responsabilités...

Ces patterns sont conçus pour imiter le comportement d'un expert face à une situation pour proposer des modèles génériques de résolution pour un type de problème particulier : " *Si je suis placé dans un **contexte initial** comme dans ces **exemples**, que je traite ce **problème** avec ces **forces** et contraintes en jeu, mais que ma situation est différente de celle des **patterns reliés**, alors il faut réfléchir selon ce **rationnel**. Si je veux obtenir ce **contexte résultant**, je dois mettre en œuvre cette **solution**. Et voilà le **nom** pour me souvenir de ce scénario.*"

On dénombre plusieurs types de Design Patterns. La catégorisation d'origine propose les patrons de création, de structure et de comportement. D'autres catégories (et classifications) ont été créées par la suite. Les Design Patterns de cette catégorisation sont généralement associés à des problèmes communs au niveau du code. Ils prévoient différents schémas visant à affiner et construire des sous-systèmes. Les modèles de conception sont des solutions de moyenne envergure qui précisent certaines structures et comportements des entités et de leurs relations.

Au-dessus des Design patterns, on distingue les **Architectural (Design) Patterns** ou les modèles architecturaux. Ces derniers, d'un niveau d'abstraction plus élevé que les modèles de

conception, proposent des stratégies de haut niveau concernant les composants de grande échelle, les propriétés globales et les mécanismes d'un système. Ce sont des schémas d'organisation structurelle de logiciels. Ces modèles architecturaux sont généralement composés d'un ensemble de Design Patterns, comme le modèle MVC.

De la même manière, les patterns se généralisent à d'autres niveaux que nous ne détaillerons pas ici : les patterns de programmation (appelés aussi idiomes ou coding patterns), d'analyse et de process, d'organisation, les antipatterns...

Avantages & Inconvénients

Les Design Patterns permettent une plus grande réutilisabilité grâce au vocabulaire commun utilisé. Ils facilitent ainsi la capitalisation de l'expérience des concepteurs sous forme de guide, présentant ces patrons avec un niveau d'abstraction élevé, encourageant les bonnes pratiques de conception. Ce qui permet l'élaboration de logiciels de plus en plus complexes et de meilleure qualité, car le patron de conception est une technique d'architecture logicielle efficace. De plus, le vocabulaire créé enrichit et simplifie la communication des développeurs. Les patterns améliorent aussi l'efficacité, la robustesse, le temps de développement, la lisibilité du code. Ils en permettent une meilleure documentation.

On retrouve ces patterns dans les frameworks et les librairies logicielles, ce qui facilite leurs utilisations et augmente leur potentiel.

Cependant, la création (ainsi que l'utilisation) d'un Design Pattern requiert un effort[3] de synthèse non négligeable. Elle nécessite un certain apprentissage et de l'expérience, notamment pour réussir à trouver les patrons adéquats, car ce ne sont pas des abstractions "naturelles". D'autre part, il faut choisir la bonne granularité et savoir ce qui doit être regroupé ou décomposé. Et surtout, sa mise en œuvre nécessite souvent des adaptations aux spécificités du logiciel.

4.2.2 Design Patterns du monde Agent

Les Design Patterns sont aussi utilisés dans le monde Agent, bien que rarement présentés sous cette forme. En effet, nous utilisons tous des motifs pour représenter certaines actions, architectures ou autres. Dans cette partie, nous présentons quelques travaux qui ont exposé cette présence dans le monde Agent.

Classification des Design Patterns existants

Klügl et Karlsson résument dans [Klügl et Karlsson, 2009] les différentes utilisations des Design Patterns dans le monde Agent. Ils discutent de l'application des patterns pour les simulations orientées agents et donnent quelques exemples illustrant des situations particulières. Pour eux, les utilisations des patterns dans la **S**imulation **O**rientée **A**gent (SOA) pouvaient être différentes de celles proposées pour les **S**ystème **M**ulti-**A**gents (SMA).

Dans le monde Agent, les Design Patterns sont utilisés, même s'ils ne sont pas connus [Cruz

[3] Il est à noter qu'il faut toujours utiliser les Design Patterns lorsque l'effort apporte un avantage, car un pattern peut introduire de la complexité parasite et inutile dans de petits projets.

Torres *et al.*, 2011]. Les auteurs expliquent que, pour maximiser les avantages des Design Patterns, on doit les appliquer uniformément dans l'ensemble de la communauté de recherche SMA. Ce qui entraînerait la diffusion de solutions SMA, et fournirait ainsi des commentaires et retours d'expérience précieux à la communauté de recherche. Cependant, ils soulèvent aussi les problèmes résultant d'une absence de vocabulaire commun.

Sauvage propose dans [Sauvage, 2003, Sauvage, 2004b, Sauvage, 2004a] un thésaurus de motifs orientés agents. Suite à une étude approfondie de la littérature, il décrit deux métamotifs (*schémas d'organisation* et *protocoles*) qui expriment les principes du paradigme agent. Il présente ensuite une série de motifs SMA que l'on retrouve dans les plateformes ou méthodologies actuelles, notamment le motif architectural BDI, mise en œuvre du modèle *Belief Desire Intention*. Cependant, il ne propose pas de nouveaux motifs architecturaux.

Figure

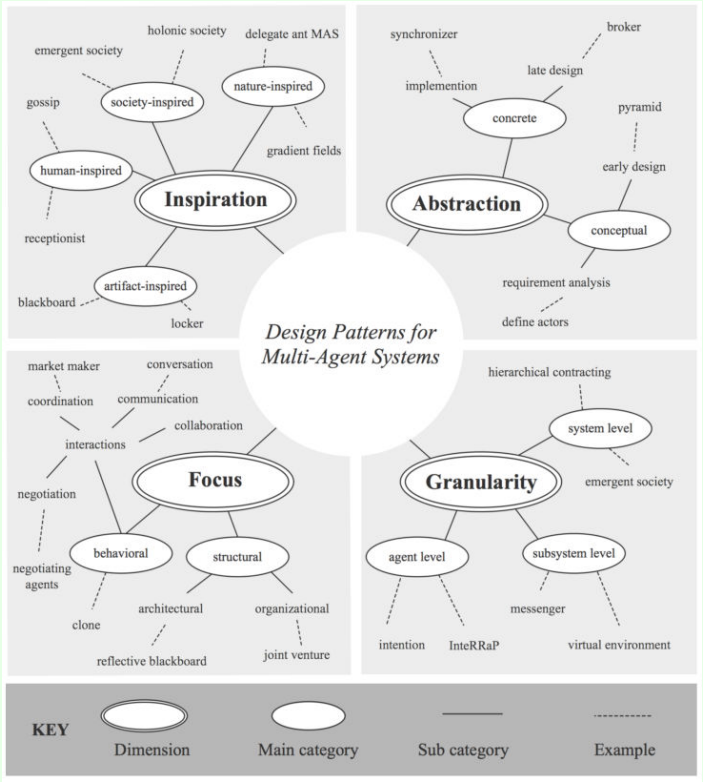


FIGURE 4.1 – Classification des patterns utilisés en SMA, avec exemples selon [Juziuk *et al.*, 2012]

Les auteurs de [Juziuk *et al.*, 2012] proposent une revue systématique des différents Design Patterns utilisés dans les SMA. Leurs recherches ont dénombré un total de 206 patterns,

regroupés en trois grands "clusters", en rapport avec leurs origines :

- Les patterns inspirés du monde Objet
- Les patterns bio-inspirés
- Les patterns en relation avec les agents mobiles

Ils expliquent aussi que 59 % d'entre eux ne sont pas dépendants d'un domaine particulier. Ils proposent une classification générique (voir Figure 4.1) :

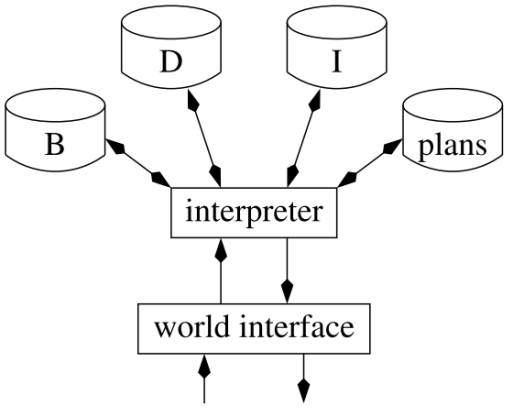
- **Inspiration** : Cette catégorie de patterns permet la compréhension des métaphores et analogies.
- **Abstraction** : Cette dimension classifie les patterns en concrets ou conceptuels.
- **Focus** : Il s'agit des patterns qui participent à la décomposition d'un système (ils sont dits structurels) ou aux interactions (ils sont de type comportements).
- **Granularity** : Cette catégorie regroupe les différentes échelles de patterns : le système, le sous-système ou l'agent.

L'analyse des récents travaux [Sauvage, 2003, Klügl et Karlsson, 2009, Juziuk *et al.*, 2012] sur les Design Patterns dans les SMA fait ressortir les grandes tendances d'utilisations suivantes :

Les Design Patterns d'architecture : Sauvage présente l'architecture BDI sous la forme d'un pattern avec une description que nous adaptons et résumons comme suit :

- **Nom :** Architecture BDI (*Belief Desire Intention*)
- **Problème :** Comment séparer les croyances, désirs, intentions et plans pour créer un agent décisionnel ?
- **Contexte initial :** Il faut mettre en adéquation l'architecture interne de l'agent et son modèle conceptuel lorsqu'il est basé sur le modèle BDI ou un de ses dérivés.
- **Forces :** Les agents que l'on utilise sont cognitifs et fonctionnent sur le modèle BDI. Ils doivent être capables de manipuler des connaissances relativement abstraites.
- **Solution :** L'architecture proposée est l'utilisation de cinq modules :
  - Module principal Interprète qui fait le lien entre le modèle interne de l'agent et l'environnement.
  - Quatre bases de connaissances connectées à l'Interprète :
    - Une base qui contient ses croyances.
    - Une base avec ses désirs.
    - Une base qui recele ses intentions (actions).

* Une base qui contient ses plans.



- **Contexte résultant :** Le modèle BDI est un modèle de délibération qui permet de modéliser le comportement décisionnel d'un agent rationnel, il n'est pas applicable à des agents aux réactions rapides.
- **Rationnel :** Séparer les croyances, désirs, intentions et plans permet de mieux définir le comportement décisionnel de l'agent.
- **Patterns reliés :** Architecture verticale, horizontale, récursive.

L'auteur présente trois autres patterns architecturaux et d'autres types de patterns. Le modèle BDI ainsi présenté nous permet de mieux comprendre les raisons pour lesquelles il a été mis en œuvre et les avantages qui en découlent.

Les Design Patterns de niveau plateforme agent : La majorité des travaux a été axée sur des patterns pour les applications ou plateformes elles-mêmes comme dans [Campos et Hill, 1998, Amblard *et al.*, 2001, Nutaro et Hammonds, 2004, Nguyen *et al.*, 2008]. En effet, ils ont utilisé MVC d'un point de vue Programmation Orientée Objet ou de Génie Logiciel, pour séparer le moteur de la plateforme des fenêtres de l'application.

Certains travaux offrent des méthodologies : Mahmoud et Maamar [Mahmoud et Maamar, 2006] proposent une approche qui permet au développeur d'utiliser le pattern MVC pour répartir les agents nécessaires à la construction du système multi-agents. Ils recommandent de diviser le software en trois parties, chacune avec ses propres agents :

- **Le Modèle :** le sous-système d'informations, non-accessible à l'utilisateur, contient les informations nécessaires pour accomplir les tâches.
- **La Vue :** le sous-système d'interface représente les interactions entre l'utilisateur et le reste du système.
- **Le Contrôleur :** le sous-système de dialogue, composé d'agents réactifs, répond aux stimuli externes et offre des services aux deux autres sous-systèmes.

Les Design Patterns qui expriment les actions : Une grande partie des travaux est centrée sur la manière de capitaliser la connaissance des experts par rapport aux actions des agents. Aridor et Lange [Aridor et Lange, 1998], pionniers dans l'application des Design Patterns au domaine des SMA, proposent dans leur article une classification ainsi que certains patterns que l'on retrouve dans le monde agent mobile. Les trois catégories qu'ils proposent sont :

- **Travelling patterns :** Ces modèles sont, pour les agents mobiles, les plus importants. Ils gèrent le déplacement des agents. Exemples : *itinary, forwarding et ticket.*
- **Task patterns :** Ces patrons sont en rapport avec la répartition des tâches et leur délégation à un ou plusieurs agents (pouvant les effectuer coopérativement ou en parallèle). Exemples : *master-slave et plan.*
- **Interactions patterns :** Le pouvoir des agents de communiquer entre eux étant vital pour leur coopération, ces patrons facilitent la localisation des agents et leurs interactions. Exemples : *meeting, locker, messenger...*

D'autres idées ont émergé comme PASSI (Process for Agent Societies Specification and Implementation) [Cossentino *et al.*, 2003]. Ici, les auteurs proposent une autre classification : action pattern, behaviour pattern, component pattern et service pattern. PASSI est une méthodologie qui permet de passer de l'analyse du problème jusqu'au code grâce à des outils comme UML (Unified Modeling Language) et AUML (agent UML) et de créer une ontologie de domaine. Les patterns utilisés dans PASSI représentent aussi une ou plusieurs actions possibles des agents.

Certains travaux sont plus centrés sur l'interaction de l'agent. Les auteurs de [Boronea *et al.*, 2009] discutent de certains patterns d'Aridor et Lange comme *Itinerary pattern, Master-Slave pattern, Meeting pattern, Messenger Pattern...* et proposent une implémentation concrète dans JADE.

Les auteurs de [Hayden *et al.*, 1999] axent plus leurs recherches dans le domaine particulier de la coordination de l'agent avec certains patterns comme *Broker, Embassy, Mediator...* selon quatre catégories : *hierarchical*, *federated*, *peer-to-peer* et *agent-pair*.

Les Design Patterns de modélisation : Il existe d'autres travaux, comme ceux de Couturier qui propose une autre vision [Couturier *et al.*, 2012] : des patterns qui couvrent toutes les phases de conception pour les systèmes d'information coopératifs, où des agents sont utilisés. Ici, les patterns proposés sont spécifiques à un domaine particulier.

Synthèse

Ce petit tour d'horizon montre que les travaux restent souvent au niveau de la plateforme (Génie Logiciel) elle-même ou des interactions entre agents. Il existe aussi des retranscriptions de patterns architecturaux à partir d'architectures d'agents existantes.

Notre proposition ici ne consiste pas à adapter une architecture existante sous forme de pattern, mais plutôt s'inspire d'un pattern existant, MVC, pour l'adapter afin de formuler une nouvelle architecture agent.

4.3 Présentation du pattern MVC

Le concept original MVC a été élaboré en 1979 au Xerox Parc par Trygve Reenskaug [Reenskaug, 2003] sous le nom de *Thing-Model-View-Editor*, puis formalisé par Steve Burbeck [Burbeck, 1987] et mis en œuvre sur Smalltalk [Krasner et Pope, 1988]. Ce modèle repose sur une séparation des concepts : logique du domaine, présentation et interactions. D'où ces trois distinctions :

- **Le Modèle** est un objet représentant le *fond*. Il s'intéresse à la représentation des données de la couche métier (*business logic*).
- **La Vue** concerne la *forme*. C'est une certaine forme de visualisation/représentation de l'état du modèle.
- **Le Contrôleur** propose des méthodes pour changer l'état du modèle.

Burbeck explique qu'on rencontre deux variantes du modèle dans MVC : l'une passive, l'autre active. Dans le modèle passif, le Contrôleur manipule exclusivement le Modèle et gère la synchronisation Vue-Modèle. Le Modèle actif est indépendant du Contrôleur, car celui-ci n'est présent que pour avertir la Vue des changements produits.

Figure

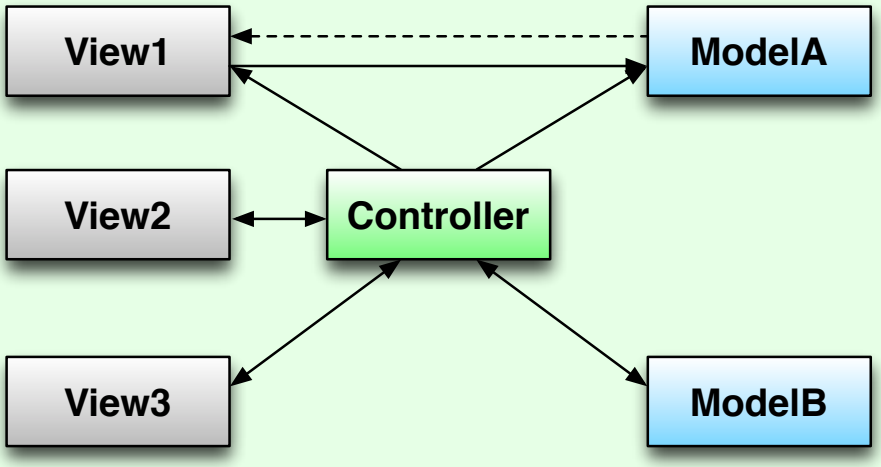


FIGURE 4.2 – Exemple d'utilisation de MVC

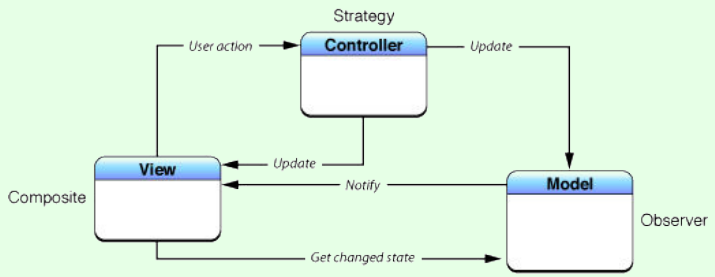
La Figure 4.2 présente un exemple de mise en œuvre du modèle MVC. De manière générale, dans nos représentations de MVC et ses variantes, les flèches en pointillés marquent la notion de "update" de manière spécifique, *c'est-à-dire* que l'entité source prévient l'entité cible

d'un changement "en lui". Les flèches pleines représentent tous les autres types de relations (modifications, envois d'une action utilisateur...).

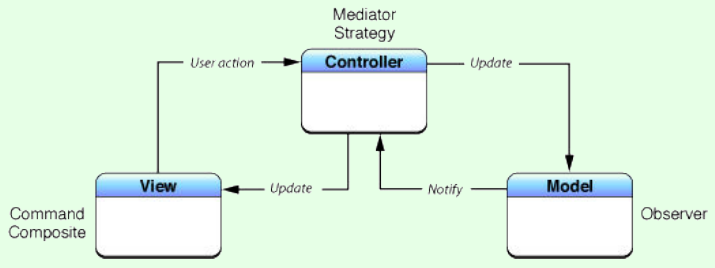
Le concept Modèle-Vue-Contrôleur n'est pas un Design Pattern simple, mais un Architectural Pattern, composé de plusieurs Design Patterns proposés dans l'ouvrage de "GoF". On les utilise dans les relations entre les composants :

- **Vue-Vue :** Le pattern **Composite** est utilisé ici pour représenter la structure potentiellement complexe et hiérarchisée, notamment lorsque la vue principale est composée de sous-vues imbriquées, coordonnées entre elles.
- **Vue-Contrôleur** : On utilise pour cette relation le pattern **Strategy** qui définit une famille d'algorithmes, encapsule chacun d'entre eux et les rend interchangeables. Un objet Contrôleur met en œuvre *Strategy* pour un ou plusieurs objets Vue.
- **Vue-Modèle** et **Contrôleur-Modèle** : Ces deux relations sont basées sur le patron **Observer** qui définit une interdépendance (one-to-many) entre les objets pour informer tous les dépendants d'un objet de ses changements d'état et permet la mise à jour automatique. Un objet Modèle laisse savoir aux objets Vue ses changements dans le cas du modèle actif. En outre, le Contrôleur doit également être notifié de tout changement du modèle.

Figure



(a) MVC Classique



(b) MVC Cocoa avec Mediator

FIGURE 4.3 – Différence entre MVC Classique et MVC Cocoa, selon Apple

Suivant les différentes variantes de MVC que nous découvrirons dans la section suivante, d'autres patterns sont parfois employés pour les relations. Par exemple :

- **Vue-Vue** : Le pattern **Decorator** qui attache dynamiquement des responsabilités supplémentaires à un objet permet ici d'ajouter une Vue (par exemple un scrolling) à une autre Vue.
- **Contrôleur-Modèle** : Le pattern **Command** est souvent utilisé, car il encapsule une requête comme un objet. Ce qui permet de paramétrer des clients aux demandes différentes, des files d'attente ou des demandes de logs (journaux) et autorise les opérations "annulables".
- **Contrôleur-Modèle** et **Contrôleur-Vue** : Ces deux relations utilisent le **Mediator** dans la version MVC dans Cocoa d'Apple[4] (voir Figure 4.3). Les objets ne communiquent plus directement entre eux, mais la communication est encapsulée par le mediator. Cette protection réduit les dépendances entre objets communicants et donc leurs couplages.

Au premier abord, le modèle MVC peut être perçu comme une *usine à gaz*, principalement parce qu'il met en œuvre beaucoup de classes ou d'objets supplémentaires, quand il semble qu'un seul puisse faire l'affaire. Mais la puissance de ce modèle se révèle non pas en codant seul, mais plutôt lorsqu'il faut maintenir le code et permettre à d'autres d'en modifier une partie sans changer le reste.

4.3.1 Variantes de MVC

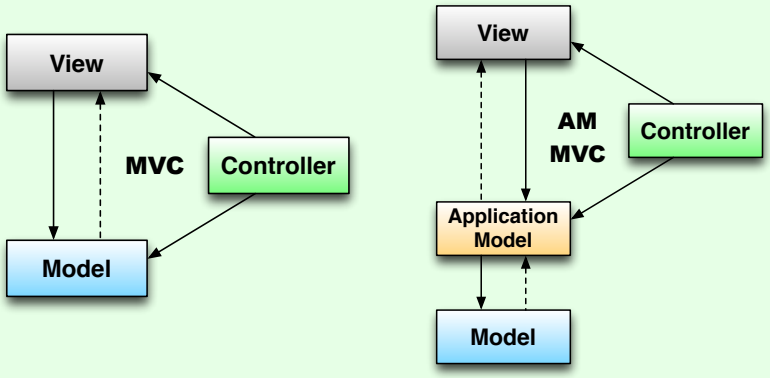
Le concept originel de MVC (voir Figure 4.4a) a connu beaucoup de variantes au cours du temps. Après plusieurs années d'utilisation, de tests et d'affinements, l'application du modèle MVC s'est développée au-delà de ses objectifs initiaux. Pourquoi tous ces changements ? Parce que la version originale présente des dépendances entre Vue et Modèle, le Modèle mettant à jour la Vue directement, ce qui n'est pas tout le temps souhaitable. Aujourd'hui, les variantes les plus utilisées en Génie Logiciel sont les suivantes :

- **AM-MVC :** L'**A**pplication **M**odel **MVC** (voir Figure 4.4b) est la variante la plus directe de MVC. Il introduit une nouvelle entité appelée Application Model[5]. Celle-ci est l'intermédiaire entre le Modèle et les Vues ou Contrôleurs afin de libérer le Modèle de tout ce qui concerne la présentation. Dans ce schéma, le Contrôleur n'a plus d'accès direct au Modèle et la Vue ne s'enregistre plus comme Observable du modèle, mais à l'Application Model. Par exemple, dans un jeu, quand la vie d'un personnage atteint '0', l'Application Model se chargera de changer la couleur de l'écran, et non le Modèle.
- **MVP :** Le **M**odel **V**iew **P**resenter [Potel, 1996] a été introduit en 1996 pour le `C++` par Mike

[4] Voir http://developer.apple.com/library/mac/#documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html pour plus d'informations (Page consultée le 1er février 2013).

[5] Voir "VisualWorks Application Model" sur http://martinfowler.com/eaaDev/uiArchs.html (Page consultée le 1er février 2013).

Figure



(a) MVC Classique (b) Application Model MVC

FIGURE 4.4 – MVC et sa variation la plus directe AM-MVC

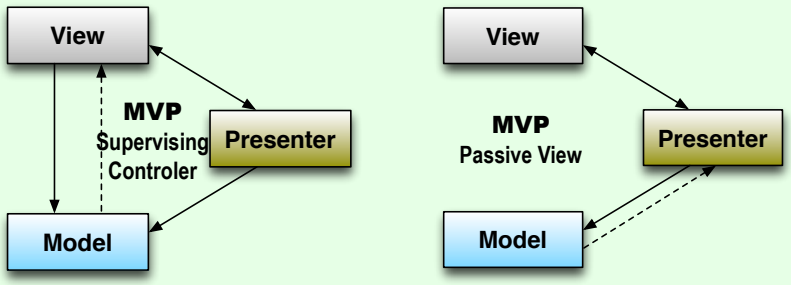
Potel. Ce patron est fondé sur MVC et AM-MVC, mais il délègue plus de travail à la Vue et supprime le Contrôleur. Le Presenter est en charge de la logique de présentation, et en tant que Contrôleur, il commande le Modèle. Il modifie également la présentation selon les règles de l'application. Étroitement lié à la Vue, il a conscience de la Vue (alors que la Vue n'a pas connaissance du Presenter). Ce modèle est plus adapté aux architectures n-tiers. Martin Fowler[6] divise MVP en deux déclinaisons : Supervising Controller (voir Figure 4.5a) et Passive View (voir Figure 4.5b).

- *Le Supervising Controller* est plus axé sur MVC : la Vue gère les événements, demande les données pertinentes au Modèle et change son affichage. Ici, la Vue est consciente de la présence du Modèle. Il y a un databinding (liaison de données) entre le Modèle et la Vue.
- *Le Passive View* est principalement basé sur AM-MVC : le Presenter gère les événements, demande les données pertinentes à partir du Modèle et modifie l'affichage de la Vue. Cette dernière n'est pas consciente de la présence du Modèle.

- **PM :** Le **P**resentation **M**odel (Figure 4.6a) est similaire à MVP, mais sépare la Vue du Presenter. La Vue est consciente du Presenter, mais pas le contraire. Le Presenter, appelé aussi Presentation Model, est une représentation logique de l'interface utilisateur (UI) qui ne s'appuie pas sur les éléments visuels. Il agit comme une version abstraite de

[6] Se référer à http://www.martinfowler.com/eaaDev/ModelViewPresenter.html (Page consultée le 1er février 2013).

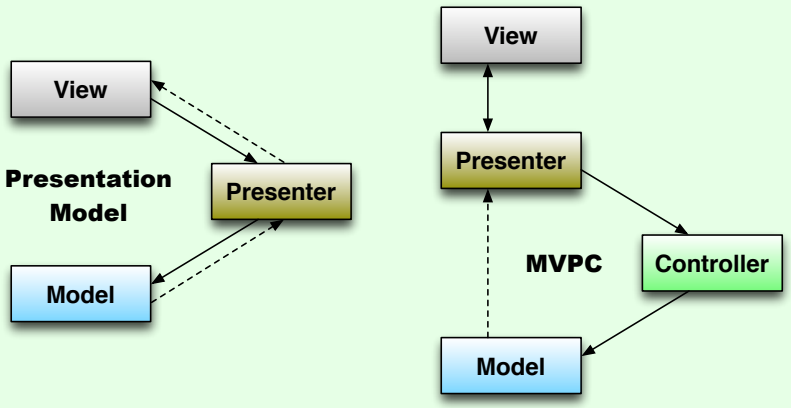
Figure



(a) MVP Supervising Controller (b) MVP Passive View

FIGURE 4.5 – Les deux variantes de MVP

Figure



(a) Presentation Model (b) MVPC

FIGURE 4.6 – Autres variantes de MVC

la Vue. Il existe aussi une autre variante appelée **M**odel **V**iew **V**iew**M**odel, également connue sous le sigle **MVVM** ou tout simplement pattern ViewModel, où on trouve un

databinding, généralement WPF et Silverlight de Microsoft, entre le Presenter, appelé ici ViewModel, et la Vue. Cette architecture facilite la gestion des modèles et des interactions complexes.

- **MVPC :** Le but du **M**odel **V**iew **P**resenter **C**ontroller (Figure 4.6b) est de séparer la logique et la présentation dans le modèle PM, qui sont étroitement liées. Ce pattern, assez complexe, permet cette séparation parfois nécessaire.

Figure

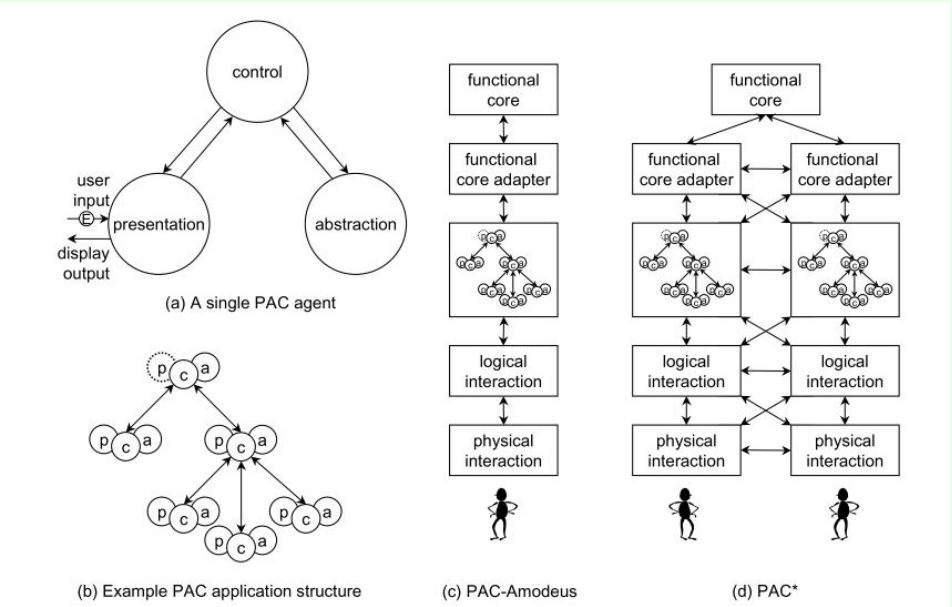


FIGURE 4.7 – PAC selon [Phillips, 1999]

- **PAC :** Le modèle **P**resentation **A**bstraction **C**ontrol décompose une application en une hiérarchie d'abstractions. Selon Coutaz [Coutaz *et al.*, 1995], créateur de ce modèle, il est plus adapté à l'organisation cognitive de l'être humain. Les modèles de types PAC et PAC-Amodeus [Coutaz *et al.*, 1996] sont orientés agents et garantissent une séparation forte entre l'UI et l'application proprement dite. PAC-Amodeus est un modèle d'architecture logicielle, qui reprend le découpage en couches de Arch [Arc, 1992], basé sur cinq composants, tout en exprimant le "contrôleur de dialogue" sous la forme d'agents PAC. Un agent PAC comprend :

  – **La Présentation** qui définit le comportement perceptible de l'agent par un agent humain (V+C de MVC)

- **L'Abstraction**, avec ses fonctions et ses attributs internes, précisant la compétence de l'agent (M de MVC).
- **Le Contrôle** a un double rôle : servir de pont entre la Présentation et l'Abstraction de l'agent, et gérer des relations avec d'autres agents PAC (pas d'équivalent dans MVC).

Dans cette méthodologie où on utilise des agents pour une IHM [Hussey et Carrington, 1997], les interactions se font entre les Contrôles des agents PAC. De plus, par rapport à MVC, une distinction claire est établie entre le modèle (Abstraction) et la communication Vue-Modèle (Contrôle). Cependant, selon les auteurs de [Tarpin-Bernard et David, 1999], cette décomposition Abstraction/Présentation est généralement insuffisante dans les cas complexes et les formalismes de modélisations des composants Contrôle sont rares et généralement limités.

Ainsi, ces patterns sont similaires à plusieurs égards, mais chacun a évolué de manière différente, pour répondre à des besoins et préoccupations distincts.

4.3.2 Performances de MVC

Le concept MVC est utilisé afin de créer des ensembles de composants systèmes. Il permet d'isoler les composants les uns des autres, autant que possible. Ce qui facilite la tâche du concepteur de l'application pour comprendre et modifier une unité particulière, sans l'obligation de tout savoir sur les autres.

Afin de rendre les performances du pattern MVC plus concrètes, nous allons par la suite en présenter une utilisation particulière : soutenir le développement de logiciels graphiques hautement interactifs. L'exemple du GUI[7] a été choisi car c'est le plus démonstratif des exemples, mais aussi parce que dans notre contexte d'usage, nous parlons d'éléments architecturaux et programmatiques, et les avantages espérés de notre proposition seront proches de ceux conférés par ce cadre d'utilisation du pattern MVC.

Réduire la complexité des GUI par MVC

Le paradigme MVC est utilisé comme suit dans les GUI :

- **Le Modèle** gère le comportement et les données du domaine de l'application, répondant aux demandes d'informations sur son état (généralement à partir de la Vue) et à des instructions pour changer d'état (habituellement de la part du Contrôleur). Il ne contient aucune information concernant l'état du GUI et ne propose aucun service spécifique à des Vues particulières et à des Contrôleurs : il est indépendant.
- **La Vue** gère l'affichage graphique et/ou textuel de la partie de l'écran allouée à son application. Elle affiche les données du Modèle et lui demande des mises à jour.

[7] **G**raphical **U**ser **I**nterface, *c'est-à-dire* un environnement graphique qui fournit un cadre de travail simplifiant la manipulation des appareils informatiques à l'aide d'interfaces graphiques.

- **Le Contrôleur** définit le comportement de l'application. Il interprète les entrées de la souris et du clavier par l'utilisateur, commandant le Modèle et/ou la Vue pour modification le cas échéant : il contrôle la synchronisation.

Le Design Pattern MVC a été décrit de plusieurs manières. Nous proposons ici celle de North et Macal dans [North et Macal, 2011]. Ces auteurs ont exposé le modèle MVC en suivant la convention de Gamma [Gamma *et al.*, 1995] dans ce cadre d'utilisation :

- **Nom :** Le nom du pattern est *Modèle-Vue-Contrôleur*.
- **Problème :** Comment séparer la logique spécifique à l'application du code de l'interface utilisateur ?
- **Contexte initial :** Une interface utilisateur avec une logique spécifique à l'application est nécessaire.
- **Forces :** Un framework générique d'interface utilisateur a besoin de travailler avec le code de domaine.
- **Solution :** Créer un modèle spécifique au domaine, connecté à un affichage d'interface utilisateur. Le point de vue est alors géré par un contrôleur. Le pattern MVC résout ce problème de création en divisant l'application en trois parties :
  - Le **Modèle**, qui contient les données et la logique du domaine.
  - La **Vue**, interface entre l'utilisateur et le système.
  - Le **Contrôleur**, gérant les interactions entre l'interface (Vue) et les données (Modèle).
- **Contexte résultant :** On a une interface utilisateur avec un modèle clairement séparé, spécifique au domaine, la vue de l'interface utilisateur et le contrôleur.
- **Rationnel :** Il est indispensable de séparer le code de l'interface utilisateur du code spécifique au domaine.

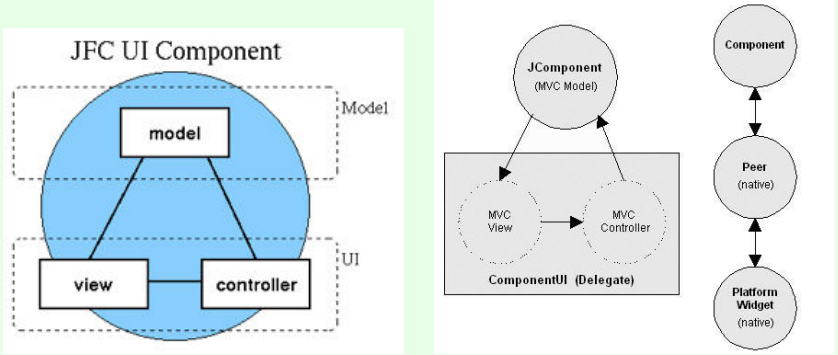
L'implémentation de MVC en Smalltalk [Burbeck, 1987] a inspiré beaucoup d'autres frameworks GUI. Si on considère la classe `Button` en Swing de Java [Stelting et Maassen, 2002] utilisée pour représenter un bouton simple, Swing utilise une adaptation de MVC dans laquelle la Vue et le Contrôleur sont combinés en un même objet appelé "*Delegate*" (Figure 4.8, d'après Oracle[8] et DFKI[9]). Cette adaptation classique permet de simplifier la communication entre Vue et Contrôleur. La classe `Button` (Tableau 4.1) est associée à un "*implementor*" `ButtonModel` pour le **Modèle**. Il encapsule les états internes d'un bouton et définit son com-

[8] Oracle Corporation, http://www.oracle.com/technetwork/java/architecture-142923.html (Page consultée le 1er février 2013).

[9] German Research Center for Artificial Intelligence, DFKI GmbH, http://www.dfki.uni-kl.de/km/java/ (Page consultée le 1er février 2013).

portement. La classe `Button` est aussi associée à `ButtonUI` pour sa **Vue**, et éventuellement à un ou plus "event handlers" pour son **Contrôleur**.

Figure



(a) MVC dans Swing (b) Différence entre Swing et AWT

FIGURE 4.8 – MVC dans Swing

Tableau

TABLE 4.1 – Exemple de MVC dans un "component" GUI

Désignation	`Button` en Swing
Modèle	`ButtonModel`
Vue	La représentation visuelle de `ButtonUI`
Contrôleur	Les "handlers" de `ButtonUI`

Presque tous les éléments complexes d'un GUI dans Swing utilisent le modèle MVC au niveau du composant pour d'excellentes raisons, mais la plus importante est qu'il est hautement réutilisable, "personnalisable" et "connectable" à d'autres composants.

Avantages & Inconvénients de MVC

La liste des avantages de MVC est longue et dépend souvent du contexte. Nous en proposons une sélection :

- **Séparation des tâches :** L'utilisation de MVC permet de séparer la logique métier, l'interface utilisateur et l'entrée du système.
- **Spécialisation et focus :** En raison de ces distinctions, les développeurs de l'UI peuvent se concentrer exclusivement sur l'interface, sans être gênés par les entrées systèmes ou la logique métier. De même pour les développeurs de la logique métier ou l'entrée du système.
- **Développement et mises à jour en parallèle par des équipes distinctes :** Un des avantages les plus intéressants est la possibilité, pour les développeurs, de travailler en parallèle. Les développeurs de la Vue peuvent s'impliquer dans la conception d'interfaces utilisateurs, tandis que ceux du Modèle construisent leurs classes en même temps. Les UI peuvent être mises à jour et la logique métier peut être modifiée, sans ralentir l'une ou l'autre équipe.
- **Vues et Contrôleurs substituables :** Différents Contrôleurs et Vues peuvent être utilisés pour le même Modèle afin de fournir une autre interface. Par exemple, les mêmes données seront vues comme un camembert ou un graphique à barres. En raison de cet avantage, MVC, très flexible, constitue un outil puissant pour la représentation des connaissances.
- **Vues multiples et synchronisées du même modèle :** Différents Contrôleurs et Vues sont utilisables pour un unique Modèle en même temps, en raison de la séparation du Modèle de la Vue. Chaque Vue présente simultanément, indépendamment et de façon synchronisée les mêmes informations à partir du Modèle.
- **Vues pluggables**[10] **:** MVC permet de fabriquer des composants UI, structurés en une hiérarchie complexe dans une GUI. Ces composants encouragent la réutilisation et réduisent le nombre de sous-classes spéciales.

Parmi les autres avantages que nous ne détaillerons pas, signalons : Le "*Look and Feel*" modifiable facilement, la possibilité de mettre en place un framework automatisant le processus, la capacité des MVC de se trouver à la fois au niveau micro (widgets, composants unitaires...) et macro (application...)...

MVC apporte les bénéfices qu'apportent les bonnes pratiques de la POO. Elle décourage les répétitions inutiles de code et améliore sa réutilisation et sa maintenabilité.

Cependant, il est important de savoir que cette architecture présente aussi les mêmes inconvénients que les Design Patterns de manière générale. En outre, des difficultés de conception peuvent se présenter au moment de l'implémentation lorsque les différentes équipes ne sont pas coordonnées (par rapport à l'interfaçage). Parfois il est nécessaire de faire évoluer le pattern pour qu'il s'applique au cas précis d'utilisation.

[10] Des vues qui sont adaptables et que l'on peut facilement connecter.

4.3.3 Synthèse

Les utilisations de MVC (et de ses variantes) montrent que ce pattern peut gérer la complexité, la flexibilité et la capacité d'évolution d'une application riche. De plus, le mécanisme de MVC se révèle d'une grande souplesse malgré l'énorme quantité de données, la réutilisation et l'adaptation des précédents composants et le développement parallèle coopératif.

Ce sont des fonctionnalités très utiles pendant et après le développement des applications. Ce pattern a été testé avec succès dans plusieurs grands projets logiciels et techniques de développement. Il existe même une adaptation des agents dans l'IHM avec les modèles PAC et AMF. Aussi, notre démarche part du questionnement suivant : Pourquoi ne pas s'inspirer de ces technologies IHM pour faciliter la formalisation des agents ?

Si nous nous tenions à la présentation de la Section 4.3.2, le paradigme MVC ne serait utilisé que pour isoler l'interface utilisateur (UI) – en général, le GUI – de la logique de domaine de l'application, comme sur l'architecture logicielle de certains SMA, dans la programmation événementielle.

La définition originale [Bodker, 1991] de l'interface utilisateur montre qu'une UI est une **surface d'interaction** entre l'utilisateur et la machine. En extrapolant cette définition, en l'adaptant au monde agent, nous pouvons convenir que la machine est un agent particulier et que "les utilisateurs" sont en fait des agents qui interagissent entre eux : la surface d'interaction sur un tel agent sera donc sa partie "visible", accessible aux autres agents !

Le plus souvent, à l'heure actuelle, construire un agent ressemble à développer une application multifenêtre multivariable dans un processus monolithique, sans outils véritablement efficaces. Notre proposition consiste à adapter MVC de la POO au SMA, en prenant en compte les caractéristiques particulières des agents. Nous avons appelé le résultat de cette adaptation : le pattern **agent MVC (aMVC)** (agent MVC).

4.4 Proposition du pattern aMVC

Les raisons pour lesquelles nous avons choisi MVC comme base de notre proposition sont les suivantes : d'abord, ce pattern a permis la création d'applications très complexes. Cette complexité se retrouve aussi dans les SMA (et plus spécifiquement les SOA) où sont modélisés et simulés des systèmes complexes multidynamiques, multienvironnements. De plus, MVC est un concept connu de tout informaticien, qu'il soit dans le domaine de la recherche ou du développement. Il a été utilisé, éprouvé et adapté de plusieurs manières. Enfin, la décomposition de l'agent que nous verrons plus loin est très proche de la triade formée par ce concept.

4.4.1 Spécificités du monde Agent

Le monde Agent et le monde Objet présentent certaines similitudes : les états internes, les unités de comportements modulaires, la communication par envoi de messages. Cependant, les SMA (et les SOA) présentent des spécificités absentes de la POO [Gangat *et al.*, 2012a].

En programmation orientée objet, la plupart des interactions sont prédéterminées. Une

action entraîne forcément une réaction de l'application. Le monde Objet est déterministe. L'objet est une unité d'exécution ; c'est celui qui appelle, qui décide d'exécuter une action. D'autre part, les SMA ont une tendance non-déterministe : les agents exercent une influence sur l'environnement, sur un autre agent directement (ou à travers l'environnement, selon certains points de vue). Cette influence peut être couronnée de succès ou non : l'agent est une unité de comportement ; c'est celui qui reçoit, qui décide d'exécuter (ou non) une action. Il filtre les messages émis en fonction de son propre intérêt. C'est sa première spécificité majeure : l'autonomie de contrôle. Les objets n'ont pas de contrôle sur leurs comportements ; les agents les contrôlent afin d'atteindre leurs objectifs.

La deuxième spécificité majeure des SMA est la notion du collectif : en POO, aucune notion de collectivité ou de dimension sociale n'existe. Par exemple, dans une interface graphique, il serait inutile, voire dysfonctionnel, de disposer de deux boutons "OK" ou plus. En SMA, c'est cette multiplicité de la même entité qui est voulue ! Nous gérons des milliers de fourmis ou de termites, ce qui ajoute une composante sociale : les agents peuvent s'engager dans des interactions complexes, par exemple la coopération, la compétition ou la négociation avec d'autres agents ; ce n'est pas le cas des objets...

Parmi d'autres spécificités : les agents sont proactifs, dans la mesure où leur exécution est une boucle infinie dans laquelle ils observent leur environnement, mettent à jour leur état et sélectionnent les actions à effectuer en saisissant des opportunités. Au contraire, les objets ne deviennent actifs que lorsqu'un autre objet "invoque une méthode" sur lui. De plus, les agents sont "capables" de prendre de mauvaises décisions, et d'apprendre de ces erreurs. Les objets ne peuvent pas prendre de décisions erronées : les erreurs commises ne sont que des erreurs de programmation et de conception, les objets ne peuvent pas en tirer de leçon.

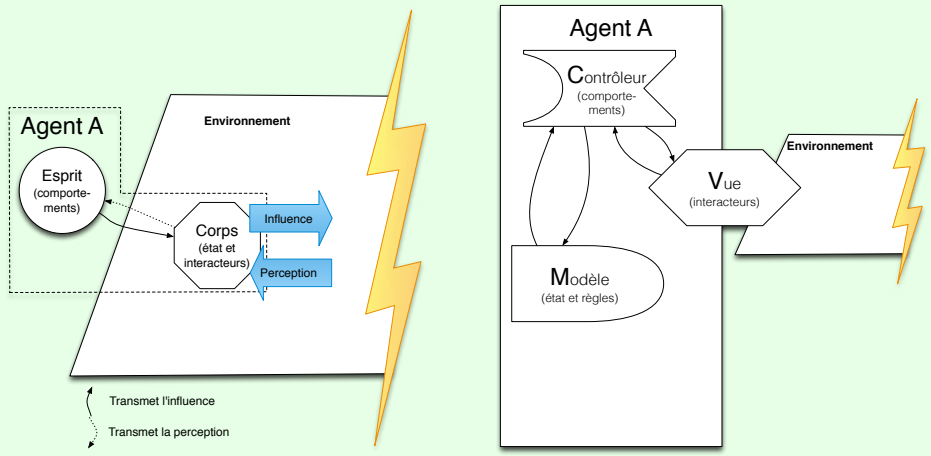
Ces spécificités nous poussent à adapter le modèle MVC pour pouvoir l'utiliser correctement.

4.4.2 Définition

Dans un contexte de simulation, les agents sont souvent considérés comme des entités possédant un "corps" et un "esprit". On retrouve cette idée de séparation explicite entre ces deux composantes d'un agent notamment dans les travaux de Soulié [Soulié, 2001] avec la notion de séparation du corps et de l'esprit d'un agent comme sur la Figure 4.9a. Dans les simulations multienvironnements comme DOM, l'agent doit pouvoir interagir avec plusieurs environnements à la fois. Pour représenter le lien entre les agents et le (ou les) environnement(s), ces deux notions sont utilisées :

- **Les corps** sont des représentations des agents au sein des environnements et sont constitués notamment de capteurs et effecteurs.
- **L'esprit** représente sa dynamique interne dans laquelle se déroulent les processus décisionnels de l'agent.

Figure



(a) Agent Trivial (b) Agent basé sur aMVC

FIGURE 4.9 – Application de aMVC

Ce principe de séparation entre le(s) corps et l'esprit de l'agent permet à un agent d'être situé dans plusieurs environnements, en démultipliant les corps associés à un même esprit. Dans cette séparation, l'état de l'agent est souvent "placé" dans le corps de l'agent (car il s'agit de son état par rapport à son environnement [Michel, 2007]), et parfois dans son esprit (car il prend des décisions en se basant notamment sur ses états).

Notre proposition s'appuie sur une première décomposition où nous caractérisons un agent selon trois composantes :

- **L'état** de l'agent, qui contient l'ensemble de ses caractéristiques internes.
- **Les comportements** de l'agent, qui organise toutes les actions qu'il peut entreprendre : c'est le processus décisionnel.
- **Les interacteurs** de l'agent (voir Définition 11).

Définition 11

Les interacteurs d'un agent représentent les outils et canaux que possède cet agent pour transmettre des influences et recevoir des perceptions dans un environnement. Ce sont les effecteurs et récepteurs (capteurs) de l'agent.

La raison principale de l'introduction de cette définition est notre volonté de séparer les états et les interacteurs du corps de l'agent. La définition usuelle d'un agent utilise cette séparation corps/esprit. Mais, dans celle-ci, le corps d'un agent représente à la fois ses états et sa capacité d'interagir avec l'environnement. Nous avons séparons ici ce corps en deux autres parties : états et interacteurs.

Cette décomposition sous forme de triade, assez naturelle, peut être facilement mise en parallèle avec MVC. L'utilisation de ces trois composantes permet d'identifier les modules M, V et C (voir Figure 4.9b) :

- **Modèle :** Dans la décomposition de cet agent, les attributs de son **état** forment le modèle. En effet, l'ensemble des valeurs de ses caractéristiques internes (l'âge, la santé, l'endurance…) constitue une base de données utilisée par les comportements. Dans certains cas, il faut y adjoindre certaines règles et lois pour la cohérence et la consistance de ces attributs. Par exemple, le vieillissement de l'agent fait partie du modèle en tant que loi inertielle. Si un attribut du modèle est utilisé dans un autre modèle du même agent, les règles de cohérence permettent de répercuter les modifications. Le chapitre suivant montrera l'utilité de ces règles [11].
- **Vue :** La Vue agit comme une surface d'interaction entre l'utilisateur et le programme dans une interface graphique. Dans le monde Agent, la Vue est une surface d'interaction entre l'agent et l'extérieur. Il s'agit donc des **interacteurs**.
- **Contrôleur :** Chez un agent, les **comportements** parfois consultent les états afin de prendre une décision et parfois les modifient. En outre, les comportements gèrent l'interaction de l'agent avec l'environnement.

Les identifications du **Modèle** et du **Contrôleur** sont évidentes. Par contre celle de la **Vue** est moins triviale : pour faciliter la compréhension de la vue, le plus simple est d'utiliser des analogies par rapport aux GUI. L'agent serait le GUI d'une application et l'environnement, l'utilisateur.

Pour que le GUI puisse percevoir ce que l'utilisateur veut, il lui propose un bouton ou une case à cocher. En cliquant sur ce bouton, l'utilisateur envoie un message qui est perçu par le GUI. En d'autres termes, les capteurs (récepteurs) d'un agent sont semblables à ce bouton ou cette case à cocher, lui permettant de percevoir des informations externes. De la même manière, le GUI transmet une information à l'utilisateur en utilisant une zone d'affichage texte ou une jauge colorée. Ces moyens lui permettent d'influencer l'utilisateur. Les effecteurs d'un agent sont comme cette zone de texte ou cette jauge colorée, lui permettant d'exercer une influence sur l'extérieur. La Vue d'un agent peut donc être considérée comme étant les interacteurs de l'agent avec l'environnement, grâce auxquels il peut exercer une influence ou une perception.

[11] Lorsque nous utilisons dans la suite l'un des deux termes sans distinction (cohérence ou consistance), cela concerne les deux termes.

Tableau

TABLE 4.2 – Similarité MVC et aMVC

Désignation	`Button` en Swing	Agent aMVC
Modèle	`ButtonModel`	Les attributs (l'état) et règles
Vue	La représentation visuelle de `ButtonUI`	Les interacteurs
Contrôleur	Les "handlers" de `ButtonUI`	Les comportements

Il est important de noter que, dans le tableau 4.2, si nous avons présenté un élément de l'interface utilisateur en parallèle avec un agent, c'est uniquement pour montrer les similarités entre MVC et aMVC. En aucun cas nous ne limitons notre définition à cette comparaison par rapport à un bouton.

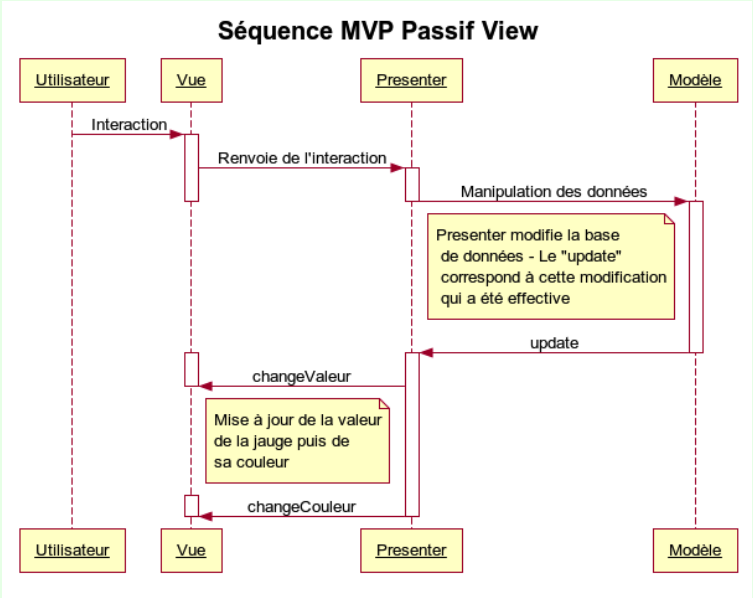
En raison des spécificités du monde Agent mentionnées dans la Section 4.4.1, nous devons choisir correctement notre variante de MVC (voir Section 4.3.1). En effet, la Vue ne doit pas être connectée directement au Modèle comme dans le MVC classique, car toutes les actions de l'agent se font au travers du Contrôleur. En réponse à la première spécificité majeure, nous devrions alors utiliser le **MVP Passif View** ou le **Presentation Model**. Comme nous le voyons sur les diagrammes UML de séquences[12] de la Figure 4.10a et 4.10b, sur un exemple de bouton qui change la valeur d'une jauge ainsi que sa couleur, le Modèle est isolé de la Vue. Cependant, le modèle **Presentation Model** est une forme de **MVP** permettant un découplage plus fort du présentateur et de la vue. La différence dans ce diagramme se situe sur la partie en italique :

- **MVP Passif View :** Lorsque nous cliquons sur le bouton, la Vue renvoie notre entrée au Présentateur. Celui-ci demande au Modèle de changer la valeur de la jauge. Le Modèle effectue l'opération en mettant à jour cette valeur et déclenche un "update"[13]. *Le Présentateur gère cet "update" en modifiant la valeur de la Vue grâce à celle obtenue puis modifie la couleur.*
- **Presentation Model :** Lorsque nous cliquons sur le bouton, la Vue renvoie notre entrée au PresentationModel. Celui-ci demande au Modèle de changer la valeur de la jauge. Le Modèle effectue l'opération en mettant à jour cette valeur et déclenche un "update". *Le PresentationModel gère cet "update" puis déclenche un autre "update" à destination de*

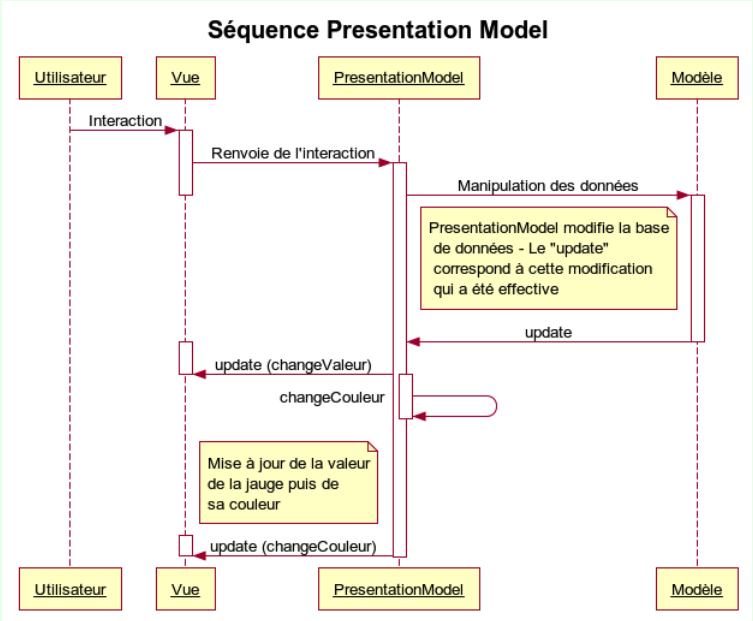
[12] Un diagramme de séquence (Sequence Diagram) est une représentation graphique séquentielle du déroulement des traitements et des interactions entre les éléments du système en utilisant la formulation Unified Modeling Language (UML).

[13] Il s'agit d'un événement pour prévenir qu'un changement

Figure



(a) Diagramme de séquence du MVP Passif View



(b) Diagramme de séquence du Presentation Model

FIGURE 4.10 – Diagrammes de séquence de MVP et PM

la Vue. Celle-ci met alors à jour la valeur de la jauge. Puis le PresentationModel grâce à la valeur de la jauge, décide de modifier la couleur et déclenche un "update", la Vue met alors à jour la couleur de la jauge.

Ainsi, le **Presentation Model** résout le problème de couplage entre la vue et la logique de présentation en utilisant des "updates" plutôt qu'en effectuant directement l'action. Ce qui augmente la testabilité de la logique, étant donné qu'il n'est pas nécessaire de disposer des composants Vues.

Figure

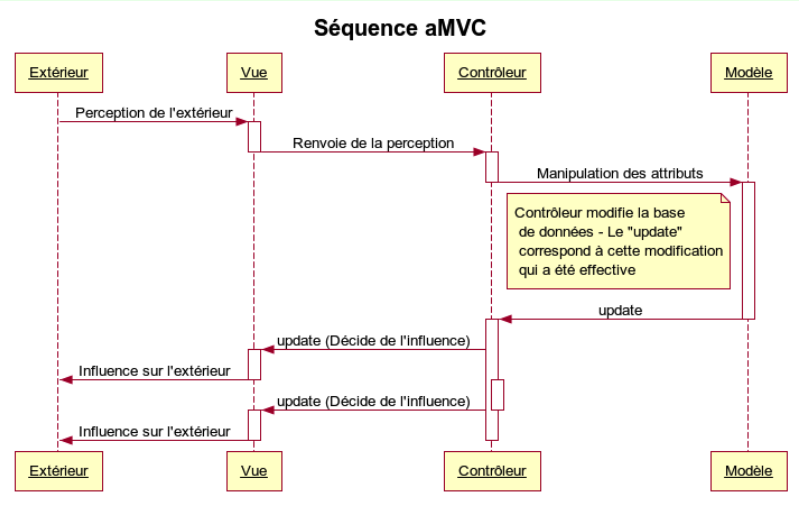


FIGURE 4.11 – Diagramme de séquence de aMVC

Comme l'explique Burbeck [Burbeck, 1987], il existe deux variantes de MVC : un modèle passif et un modèle actif. Dans le modèle passif, le contrôleur manipule exclusivement le modèle et gère la synchronisation entre la vue et le modèle. Dans le modèle actif, le modèle est partiellement indépendant du contrôleur, qui est présent pour avertir la(les) vue(s) que des changements se sont produits. Dans notre contexte, le modèle actif comme celui du **Presentation Model** est préférable. Un des avantages est que nous pourrions mettre en œuvre plus d'un seul contrôleur pour un même modèle.

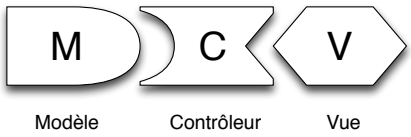
En nous basant sur la variante **Presentation Model** pour notre structure aMVC, nous permettons un faible couplage entre les différents composants. La séquence pour un agent aMVC est représentée sur la Figure 4.11 : lorsque l'extérieur effectue une influence, la Vue délègue cette perception au Contrôleur. Il manipule les données du Modèle pour modifier les attributs de l'agent. Le modèle effectue l'opération, met à jour son état si besoin est, puis déclenche un événement. Le Contrôleur reçoit la réponse et décide de l'influence adéquate.

Il envoie cette influence sous forme d'événement à la Vue qui tente ensuite de l'exécuter. Si nécessaire, il décide d'une seconde influence.

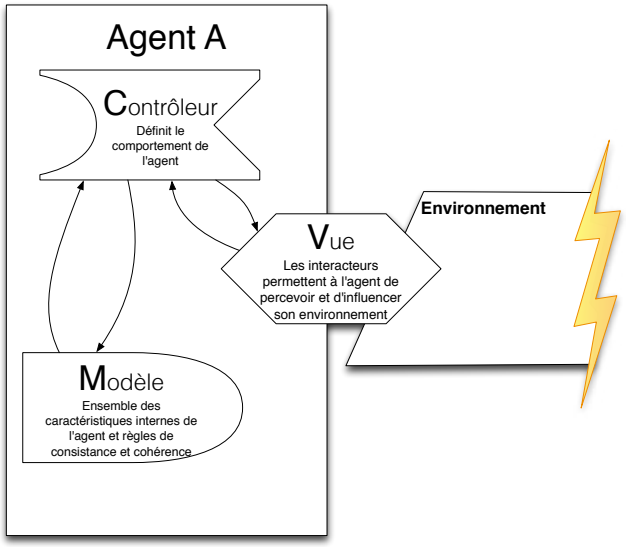
Le vocabulaire de la Section 4.2.1 permet de décrire le Design Pattern aMVC comme suit :

- **Nom :** Le nom du pattern est *Agent Model-View-Controller*, noté **aMVC**.
- **Problème :** Comment séparer le comportement des agents de leurs interacteurs et états, pour faciliter la réutilisation ?
- **Contexte initial :** Un agent doit avoir des interactions avec l'"extérieur" (l'environnement ou un autre agent) ainsi qu'avec son "intérieur" (ses états).
- **Forces :** Les agents que l'on utilise sont cognitifs ou réactifs. Ils ont des comportements complexes réutilisables. C'est pourquoi séparer les comportements des interacteurs et des états est nécessaire, tout en gardant un liant interne.
- **Solution :** L'architecture proposée est la mise en place d'un Modèle et d'une Vue, gérés par un Contrôleur. Le pattern aMVC résout ce problème en décomposant ainsi l'agent.
- **Participants :** Les participants de ce pattern sont décrits comme suit :
  - **Le Modèle** M, l'état de l'agent (et les règles de consistance et cohérence).
  - **La Vue** V, interface entre l'agent et l'"extérieur".
  - **Le Contrôleur** C, constitué du comportement de l'agent, gérant les interactions entre "intérieur" et "extérieur" à travers la Vue.

Nous utiliserons le formalisme de représentation graphique suivant pour représenter les modules M, V et C dans nos schémas :



Ce pattern s'organise comme sur la Figure suivante avec un agent A :



- **Collaborations :** Les relations existantes entre les participants sont les suivantes :
  - **Modèle-Contrôleur :** En cas de modifications dans le Modèle (l'état de l'agent), ce dernier le notifie au Contrôleur par un "update".
  - **Contrôleur-Modèle :** Le Contrôleur peut demander la lecture ou la modification des valeurs du Modèle.
  - **Vue-Contrôleur :** La Vue notifie au Contrôleur les perceptions provenant de l'environnement (extérieur).
  - **Contrôleur-Vue :** Le Contrôleur décide de l'influence qu'exercera la Vue sur l'environnement (extérieur) par l'envoi d'un "update".
- **Contexte résultant :** L'agent est clairement divisé en Modèle, Vue et Contrôleur.
- **Rationnel :** Séparer le comportement des interacteurs et des états s'avère nécessaire pour faciliter la réutilisation des différents composants.
- **Patterns reliés :**
  - Par rapport aux autres patterns similaires du monde Objet : MVC, PM, MVP Passif View.
  - Par rapport aux autres patterns similaires du monde Agent : BDI, Architecture récursive.

À partir de cette définition "de base" du Design Pattern aMVC, nous proposons un formalisme plus général :

Définition 12

(Agent aMVC) Un agent aMVC A est un ensemble non vide de couples $\langle M^A, C^A \rangle$ et $\langle C^A, V^A \rangle$ où :

- M^A est un sous-ensemble de l'ensemble $\mathscr{E}$ des caractéristiques internes formant l'état interne de l'agent, ainsi que les règles de cohérence et consistance.
- V^A est un sous-ensemble de l'ensemble $\mathscr{I}$ des récepteurs et effecteurs.
- C^A est un sous-ensemble de l'ensemble $\mathscr{C}$ des comportements.

$Agent^A = \{\langle M^A, C^A \rangle^*; \langle C^A, V^A \rangle^*\}$

Notons que, par construction, les composantes C^A servent de point de jonction entre les différents couples. En partageant des parties C communes, les couples créent globalement un graphe connexe : l'agent .

Quelques propriétés immédiates :

- Si un agent aMVC possède n Modèles, alors $\bigcup_{i \in [\![1;n]\!]} M_i^A = \mathscr{E}$
- Le graphe formé par les couples $\langle M^A, C^A \rangle$ et $\langle C^A, V^A \rangle$ d'un agent aMVC est connexe, *c'est-à-dire* que pour tout couple de sommets du graphe formé par les M^A, V^A et C^A , il existe une "chaîne" allant de l'un à l'autre.
  En effet, si le graphe n'est pas connexe, c'est qu'il est composé de deux ou plusieurs composantes connexes. Par exemple, pour un couple dissocié du reste de l'agent. Cela signifierait que cet agent pourrait être scindé en deux ou plusieurs agents, car il n'y a aucune relation entre ces sous-parties.

La définition 12 est celle d'un agent aMVC générique. Elle peut être adaptée à des cas particuliers, par exemple en posant les simplifications que pour un agent :

  - M est l'ensemble $\mathscr{E}$ des caractéristiques internes formant son état interne (et les règles).
  - V est l'ensemble $\mathscr{I}$ de ses récepteurs et effecteurs.
  - C est l'ensemble $\mathscr{C}$ de ses comportements

- Dans le cas d'un agent purement réactif, nous aurons $Agent^{reactif} = \{\langle C, V \rangle\}$. Cet agent n'ayant pas besoin de consulter son état pour réagir, la composante M a été supprimée.
- Un agent sans interacteurs se réduit à $Agent^{incomplet} = \{\langle M, C \rangle\}$ (par exemple lorsque sa Vue a été retirée dynamiquement). Cette forme peut servir à un agent qui aurait déjà deux couples $\langle M, C \rangle$ et $\langle C, V \rangle$ auquel nous ajoutons un couple $\langle M, C_c \rangle$ qui permettrait de modifier le modèle M sans interaction externe immédiate, un peu comme une conscience.

- Un agent cognitif de base est défini par $Agent^{cognitif} = \{\langle M, C\rangle; \langle C, V\rangle\}$.
- Le contenu des composantes M, V et C peut être adapté au contexte. Par exemple, dans le cas d'un agent possédant des connaissances $\mathscr{K}$ qu'il peut avoir sur son environnement : le modèle sera défini par $M = \mathscr{E} \cup \mathscr{K}$.
- Le nombre de relations MVC peut être modifié selon le besoin. Par exemple, dans le cas d'un agent B où l'on désire séparer les comportements en relation avec la communication $\mathscr{C}_c$ de ceux du mouvement $\mathscr{C}_m$, avec les interacteurs respectifs $\mathscr{I}_c$ et $\mathscr{I}_m$, nous aurons l'agent représenté dans la Figure 4.12, avec les composantes suivantes :

  - $M = \mathscr{E}$
  - $V_c = \mathscr{I}_c$
  - $C_c = \mathscr{C}_c$
  - $V_m = \mathscr{I}_m$
  - $C_m = \mathscr{C}_m$

  et donc : $Agent^B = \{\langle M, C_c\rangle; \langle C_c, V_c\rangle; \langle M, C_m\rangle; \langle C_m, V_m\rangle\}$

Figure

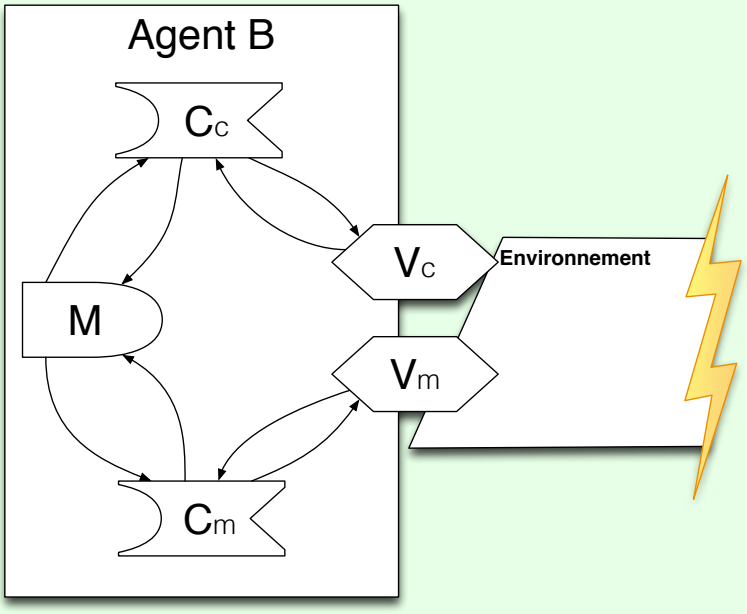


FIGURE 4.12 – Agent aMVC avec deux Vues et deux Contrôleurs

- Dans le cas d'un agent qui n'a aucun comportement mais qui peut subir des influences extérieures, nous utilisons un Contrôleur que l'on nommera "direct" . Sa tâche consiste juste à modifier l'état de l'agent selon les informations reçues par les capteurs.

4.4.3 Avantages & Inconvénients de aMVC

La liste des avantages de aMVC peut être mise en parallèle avec celle de MVC présentée en Section 4.3.2 :

- **Séparation des tâches :** En utilisant aMVC, nous pouvons scinder un agent selon son état, ses comportements et ses interacteurs, et ainsi mener des réflexions et des conceptions isolements sur ces trois aspects.
- **Spécialisation et focus :** En raison de cette séparation, un expert responsable de tout composant (M, V ou C) de l'agent peut se concentrer exclusivement sur celui-ci sans être gêné par autre chose.
- **Développement et mises à jour en parallèle par des équipes distinctes :** Un des avantages les plus intéressants est la possibilité pour chacun de travailler en parallèle grâce aux capacités héritées du pattern MVC sur la limitation des conflits et des inter-blocages.
- **Vues et Contrôleurs substituables :** Différents Contrôleurs et Vues peuvent être utilisés pour le même Modèle afin d'obtenir des effets de substitution. Par exemple, le comportement "*Déplacer*" peut être relié à l'interacteur générique "*Ailes*" ou "*Jambes*". Cet avantage de aMVC augmente la flexibilité de la composition des agents.
- **Vues multiples et synchronisées du même modèle :** Différents Contrôleurs et Vues sont utilisables pour un unique Modèle en même temps, en raison de la séparation du Modèle de la Vue. Chaque Vue présente simultanément, indépendamment et de façon synchronisée certaines informations identiques, filtrées par des comportements différents, issues d'un même Modèle.
- **Vues et Contrôleurs pluggables :** aMVC permet de créer des composants spécialisés dans le comportement et d'autres dans les interacteurs. Ces composants encouragent la réutilisation car ils regroupent au plus près les éléments concernés par un même contexte d'usage.

Cependant, il est important de garder à l'esprit que cette architecture aMVC présente aussi les mêmes inconvénients que MVC. Comme la plupart des Design Patterns, sa mise en place peut être laborieuse sans méthodologie. Pour ces raisons, au chapitre 6, nous présenterons une contribution complémentaire : une méthodologie facilitant la mise en œuvre du pattern aMVC dans le cadre d'une démarche de co-construction pluridisciplinaire de SOA.

4.5 Validation de l'agent aMVC par comparaison avec les définitions de références

Notre but étant de construire un système multi-agents à partir du formalisme aMVC, nous devons vérifier qu'il correspond aux définitions de référence des agents des SMA. Même si aucune définition n'est acceptée unanimement [Nwana, 1996], et que les plus 'reconnues' datent de la fin des années 90, nous allons, avec un certain recul, nous baser sur trois de ces définitions.

Dans un premier temps, nous nous appuierons sur le livre de Jacques Ferber :"Les Systèmes Multi Agents : vers une intelligence collective" [Ferber, 1995] datant de 1995, qui nous servira de base pour notre comparaison (définition 13). Ensuite nous prendrons la définition proposée par Jennings, Sycara et Wooldridge [Jennings *et al.*, 1998] (définition 14) émise en 1994, puis reprise en 1998 et 2002 par Wooldridge. La dernière que nous utiliserons est celle de Davidsson et Johansson [Davidsson et Johansson, 2005] (définition 15), qui, dans un article de 2005, propose une amélioration de la définition d'agent, se basant sur toutes celles qui existent et qui pourrait englober la majorité d'entre elles.

4.5.1 Définition de Ferber

Définition 13

On appelle agent une entité physique ou virtuelle...

a. qui est capable d'agir dans un environnement,

b. qui peut communiquer directement avec d'autres agents,

c. 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),

d. qui possède des ressources propres,

e. qui est capable de percevoir (mais de manière limitée) son environnement,

f. qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),

g. qui possède des compétences et offre des services,

h. qui peut éventuellement se reproduire,

i. 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.

Dans cette définition (13) nous distinguons plusieurs caractéristiques d'un agent, que nous allons analyser et comparer avec l'agent aMVC.

Le **Modèle** répond à la définition 13(d). La **Vue** correspond aux définitions 13(a)(b)(e)(f)(g). La Vue étant les interacteurs de l'agent (qui permettent l'influence et la perception de l'environnement), elle lui permettra donc (a,g), de percevoir (e,f). L'existence d'un ensemble de comportements et interacteurs dédié à la communication lui permettra de communiquer avec d'autres agents (b). Le **Contrôleur** de l'agent peut être mis en parallèle avec les définitions 13(c)(i). Le contrôleur contient tous les comportements de l'agent. Ce sont eux qui lui permettront d'atteindre son objectif (c) en tenant compte de ses ressources personnelles et de ce qu'il perçoit de l'extérieur (i).

Le seul élément de la définition non pris en compte est la capacité éventuelle de reproduction (h). Cette capacité pourrait être un comportement (Contrôleur) basé sur ses ressources internes (Modèle) et sa perception (Vue) pour influencer une reproduction (Vue).

4.5.2 Définition de Jennings *et al.*

Définition 14

Un agent est un système informatique, **situé** dans un environnement, qui agit d'une façon **autonome** et **flexible** pour atteindre les objectifs pour lesquels il a été conçu.

Dans cette définition (14), nous isolons trois mots-clés : situé, autonome et flexible :

- *Situé* signifie que l'agent peut recevoir des messages sensoriels provenant de son environnement et qu'il peut réaliser des actions susceptibles de changer cet environnement. Ce qui correspond dans notre modélisation aux interacteurs, c'est-à-dire à la **Vue**.
- *Autonome* est un concept difficile à appréhender de manière générale. Ici l'agent est capable d'agir sans intervention directe d'êtres humains (ou d'autres agents) et il contrôle ses actions et son état interne. C'est ce que font le **Contrôleur** et le **Modèle**.
- La définition de *flexible* peut être divisée en 3 parties [Jennings *et al.*, 1998] :
  - Le caractère "responsif" de l'agent : il peut percevoir son environnement et répondre rapidement aux changements qui s'y produisent (gestion par la **Vue** et le **Contrôleur**).
  - Le caractère proactif de l'agent : il n'agit pas seulement en réponse à son environnement, mais il est capable d'adopter un comportement opportuniste, dirigé par ses objectifs et ses comportements et prendre des initiatives au moment approprié (gestion par le **Contrôleur** et le **Modèle**).
  - Le caractère social de l'agent : il est capable d'interagir, si nécessaire, avec les autres agents ou humains, pour compléter ses propres tâches ou aider les autres à les compléter (gestion par le **Contrôleur** et le **Modèle**).

4.5.3 Définition de Davidsson *et al.*

Définition 15

Un agent peut être défini comme une entité qui **possède des états**, qui **est située (capable de percevoir et d'agir) dans un environnement**, qui est **rationnelle**, et qui **est au moins, du point de vue réactif, autonome**.

Cette définition (15) de Davidsson et Johansson [Davidsson et Johansson, 2005] rassemble différents points de vue :

- Celui de Jacques Ferber [Ferber, 1995],
- Celui de Jennings, Sycara et Wooldridge [Jennings *et al.*, 1998],
- Celui de Luck, McBurney et Preist, tiré du livre "*A roadmap for Agent Based Computing*" (2003) [Luck *et al.*, 2003]
- Celui de Weiss, extrait de :"*A modern approach to distributed artificial intelligence*" (1999) [Weiss, 1999]
- Celui de Russell et Norvig, "*Artificial Intelligence : A Modern Approach*" (2003) [Russell et Norvig, 2003]

Elle définit un agent "basique" et peut être facilement mise en parallèle avec notre définition aMVC, notamment par la correspondance suivante de trois grands axes :

- *Une entité qui possède des états* : Ce qui correspond dans notre modélisation au **Modèle**, qui contient l'ensemble des états.
- *Une entité qui est située (capable de percevoir et d'agir) dans un environnement* : Le caractère "situé" signifie ici aussi que l'agent peut percevoir et influencer son environnement. Ce qui correspond dans notre modélisation aux interacteurs, c'est-à-dire à la **Vue**.
- *Une entité qui est rationelle* : La rationnalité est la capacité d'agir de manière à obtenir le plus de succès possible dans la réalisation de l'objectif. C'est le rôle du **Contrôleur**.
- *Une entité qui est au moins, du point de vue réactif, autonome* : Dans cette définition, l'autonomie veut dire au minimum que l'agent est capable d'agir sans intervention extérieure directe. C'est ce que fait le **Contrôleur** suivant les "updates" du **Modèle**.

4.6 Conclusion

Après une petite parenthèse pour donner un aperçu des Design Patterns utilisés en POO et SMA, nous avons présenté le pattern bien connu MVC. Nous avons également discuté de ses variantes et de ses avantages.

Dans ce chapitre, nous avons introduit tous les concepts sous-jacents pour proposer une architecture agent aMVC et sa formalisation. Il s'agit du concept initial aMVC, dans lequel nous avons identifié :

- M comme étant une partie de l'état interne de l'agent.
- V, une partie des interacteurs de l'agent.
- C, une partie de ses comportements.

De manière générale, dans la suite du manuscrit, lorsque nous faisons référence au Modèle, à la Vue ou au Contrôleur, c'est à ceux de aMVC. Cette décomposition de l'agent avec un faible couplage nous donne bien la possibilité de réutiliser l'agent lui-même, mais aussi tous sous-constituants. Dans le chapitre suivant, nous allons développer ce concept dans la continuité de DOM, *c'est-à-dire* montrer son utilité dans un cadre multidynamique. Ce qui nous permet de proposer une architecture agent capable de supporter aussi la réutilisabilité au niveau des dynamiques, et donc au niveau des sous-modèles et modèles.

5 Cadre Opérationnel : Intégration de aMVC à DOM

Plan du chapitre

5.1 Introduction

Dans le Chapitre 4, nous avons introduit la notion d'**a**gent **MVC** (aMVC) comme étant une transposition du modèle **M**odèle **V**ue **C**ontrôleur (MVC) du monde orienté objet vers le monde des agents. En appliquant ce modèle au **S**ystème **M**ulti-**A**gents (SMA) d'un système complexe, nous allons bénéficier de la majorité des avantages du modèle MVC. Pour ce faire, dans notre nouvelle approche ascendante, nous proposons une décomposition de l'agent pour satisfaire ce modèle de conception [Gangat *et al.*, 2012b].

Dans ce chapitre, nous définirons le cadre opérationnel de ce pattern dans un contexte précis, au centre de notre problématique : le multidynamique. Nous allons donc intégrer aMVC à la **M**odélisation **O**rientée **D**ynamique (DOM). Nous proposerons ensuite les éléments pour mettre en place l'architecture d'une plateforme de **S**imulation **O**rientée **A**gent (SOA) supportant pleinement aMVC.

5.2 Intégration de l'agent aMVC au concept multidynamique

Dans la Section 4.4.2 du chapitre précédent, nous avons présenté la fondation de notre proposition de pattern aMVC. Cette section propose une approche de construction d'agent aMVC spécifiquement adapté au cadre du concept multidynamique.

5.2.1 Décomposition d'un agent trivial

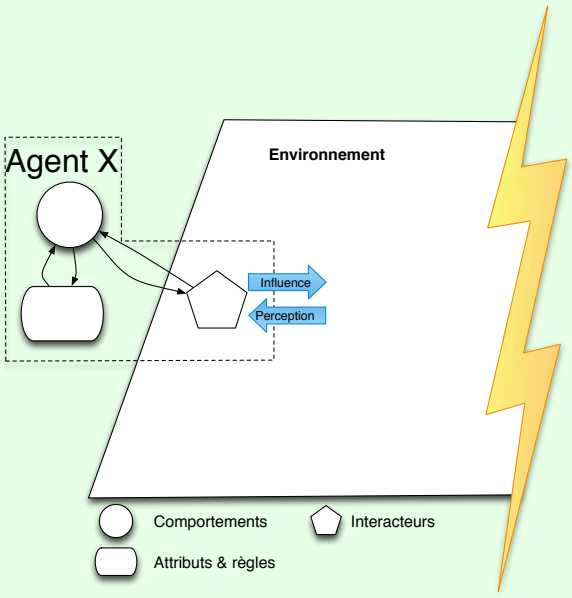
Pour présenter une formalisation de l'agent aMVC dans le contexte multidynamique, un premier découpage s'établit sur la base de trois fractionnements successifs.

Fractionnement de l'agent en fonction des dynamiques

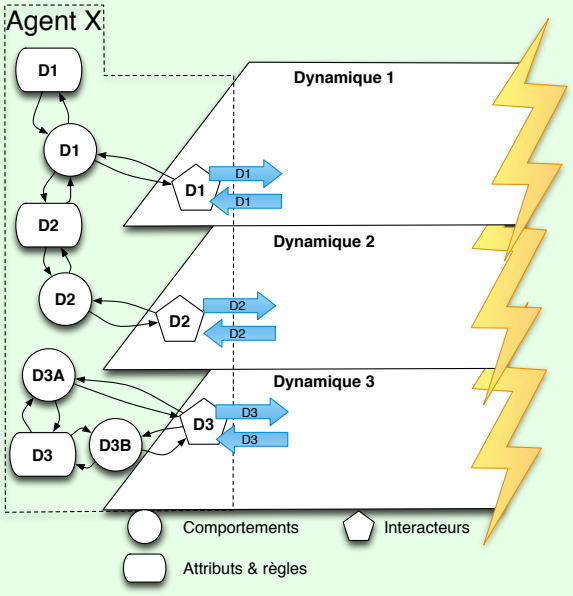
Avec un agent trivial (voir Figure 4.9a de la page 86), sans méthodologie particulière, nous avons un ensemble d'attributs internes (et les règles associées) situé dans son corps et un ensemble de comportements qui forme son esprit et constitue sa dynamique interne. Pour faciliter la compréhension des schémas, nous avons séparé ces trois composantes pour un agent X comme dans la Figure 5.1a.

Le "fractionnement de l'agent en fonction des dynamiques" signifie pour nous l'*effet que produit le fractionnement de l'environnement en plusieurs dynamiques sur un agent.* Comme nous l'avons remarqué dans la Section 3.2, l'utilisation de DOM implique de distinguer plusieurs dynamiques. Or, l'agent y est lié au travers de ses attributs et comportements en rapport respectivement avec chacune de ces dynamiques. En quelque sorte, des sous-ensembles d'attributs et de comportements de l'agent sont respectivement corrélés aux différentes dynamiques en jeu.

Figure



(a) Agent Trivial



(b) Fractionnement de l'agent en fonction des dynamiques

FIGURE 5.1 – De l'agent trivial au premier fractionnement

Exemple

Un même exemple illustratif, fictif et simple, sera développé tout au long de ce chapitre pour mieux percevoir la structuration proposée pour un agent aMVC. Il s'agira du cas d'un projet pour simuler plusieurs populations (loups et caribous) au sein d'une SOA, dans le but, par exemple, d'étudier l'évolution des deux populations, avec l'ingérence des hommes pour l'élevage. Un agent loup de base (comme les agents caribou ou homme) possède un état, un corps et des comportements.
Le premier partitionnement se fera au travers des dynamiques, où nous identifions deux caractéristiques majeures : "**la dynamique spatiale**" et la "**dynamique de communication**". La première comprend tout ce qui est en rapport avec le physique de l'individu, par exemple sa position, sa vitesse... ; la seconde est en rapport avec la faculté de communiquer avec la meute/le troupeau selon sa position sociale au sein du groupe. L'objet de l'étude étant l'évolution des deux populations, nous pouvons ajouter "**la dynamique d'évolution des loups**" et "**la dynamique d'évolution des caribous**" qui contiendront les éléments nécessaires à celle-ci (pyramide des âges, taux de natalité...).

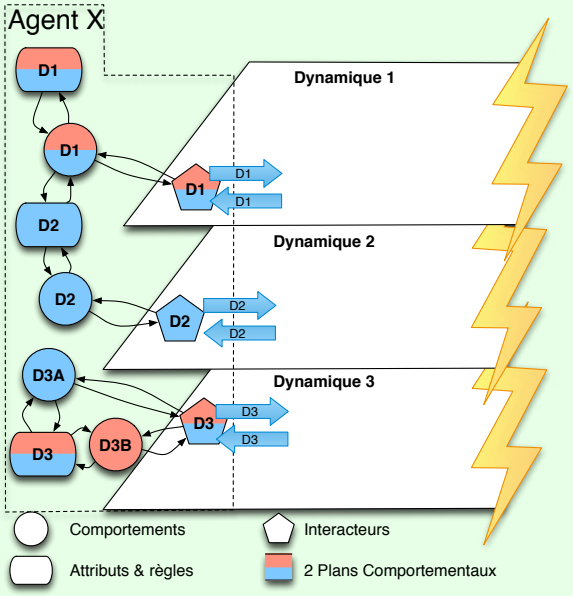
Pour appliquer DOM, il est nécessaire de séparer en sous-ensembles les attributs et les comportements, tout en sachant que certains comportements peuvent avoir des effets sur d'autres attributs en rapport avec une autre dynamique. Par exemple, le comportement D1 dans la Figure 5.1b a des effets sur les attributs D1 et D2 (l'exemple de cette Figure se veut le plus générique possible en représentant les différents cas possibles). Chacun de ces sous-ensembles est exclusivement associé à la dynamique considérée. Dans cet exemple, la scission de l'environnement se fait en trois dynamiques ($Dyn1$, $Dyn2$ et $Dyn3$) et celui des agents en sous-ensembles correspondants.

Fractionnement de l'agent en fonction des plans comportementaux

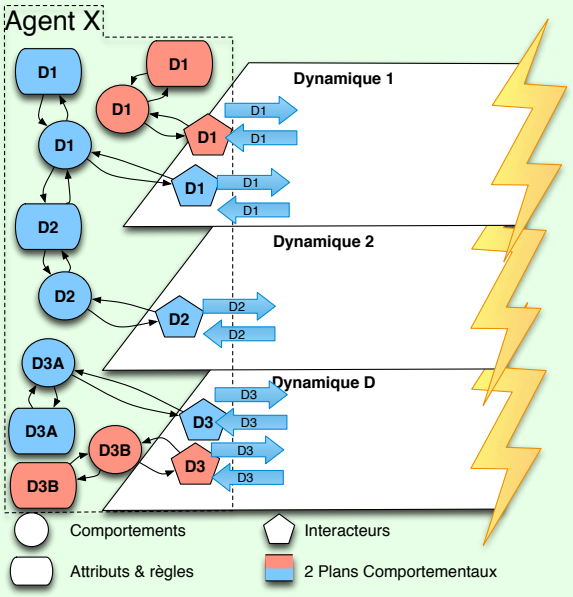
Un plan (ou dimension) comportemental correspond à un domaine d'expertise comportementale spécifique, par exemple les comportements sociaux ou les comportements de consommation d'énergie. La définition d'un plan comportemental est à un niveau différent de celui d'une dynamique. En effet, cette dernière regroupe toutes les activités qui participent à une caractéristique majeure, ce qui peut correspondre à plusieurs domaines d'expertise comportementale.

Ainsi une dynamique peut être formée d'un ou plusieurs plans comportementaux. Par exemple, les plans comportementaux "social" et "position" font partie de la dynamique de "communication". Un autre exemple, la dynamique de "l'évolution de l'énergie" de **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation (EDMMAS) est composée de quatre plans comportementaux : la "production d'énergie par les centrales", la "consommation d'énergie par les habitations", la "consommation d'énergie par les usines" et le "stockage d'énergie". Chacun de ces thèmes requiert une expertise particulière. Par ailleurs, un plan comportemen-

Figure



(a) Identification des plans comportementaux



(b) Fractionnement de l'agent en fonction de ces plans

FIGURE 5.2 – aMVC et plans comportementaux

tal est spécifique à un type d'agent alors qu'une dynamique ne l'est pas. Seconde différence : un plan comportemental peut avoir des implications sur plusieurs dynamiques. Par exemple, la dimension "sociale" du loup a des effets à la fois sur la dynamique de "communication" (un loup alpha est toujours obéi par ses congénères) et sur celle de "l'évolution de la population des loups".

Cette identification des plans comportementaux entraîne un deuxième découpage des agents. Chaque sous-constituant de l'agent (états et comportements) peut être ainsi divisé en fonction du plan auquel il se réfère. La composition de l'agent X dans la Figure 5.2a montre une certaine quantité de sous-ensembles (proportionnelle à la complexité du système) reliés les uns aux autres. Notre objectif ici est d'organiser ces sous-ensembles pour faciliter la conception de l'agent. Cette découpe transversale de l'agent suivant les plans comportementaux complète la partition obtenue par DOM en terme de dynamique. Dans l'exemple de la Figure 5.2b, la scission se traduit par deux plans comportementaux : le bleu et le rouge.

Exemple

Reprenant notre Simulation Orientée Agent de loups et de caribous, séparons chaque type d'agents suivant trois plans comportementaux : dans les domaines émotionnel, social et de la survie. Dans cet exemple, chacun des plans comportementaux participe aux dynamiques de "communication" et d'"évolution de la population" concernée. Celui de la survie implique la dynamique "spatiale" aussi. Dans le cas présent, nous obtenons six plans comportementaux qui seront organisés comme suit :

Dyn. / Plan C.	Spatiale	Communication	Evol. pop. Loup	Evol. pop. Caribou
Émotionnel du loup	✗	✓	✓	✗
Social du loup	✗	✓	✓	✗
Survie du loup	✓	✓	✓	✗
Émotionnel du caribou	✗	✓	✗	✓
Social du caribou	✗	✓	✗	✓
Survie du caribou	✓	✓	✗	✓

Légende :
✓ (ou texte correspondant) = participe
✗= ne participe pas

Fractionnement de l'agent en fonction des composantes aMVC

L'agent est ensuite décomposé, conformément à ce que nous avons vu en Section 4.4.2, en trois composantes (voir Figure 5.3). L'agent est donc caractérisé dans chaque plan comportemental de chaque dynamique par :

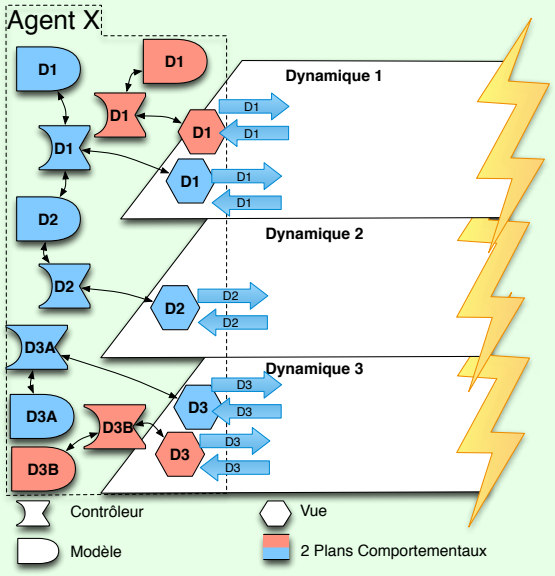
- **Le Modèle**, qui représente les caractéristiques internes de l'agent et les lois internes liées à la dynamique et au plan comportemental, sa connaissance. Ces lois permettent l'évolution propre de certains attributs (vieillissement d'un agent, augmentation progressive de la faim...). Elles permettent aussi de garder la cohérence et la consistance de ces attributs.
- **Le Contrôleur** qui contient les comportements de l'agent (processus décisionnel). Ceux-ci gèrent les interactions de l'agent avec l'environnement et, dans ce but, utilisent ses états : soit pour les consulter avant de prendre une décision, soit pour influencer leurs modifications afin de mémoriser l'apprentissage issu de l'expérience. Les comportements peuvent à la fois prendre des décisions pour agir sur l'environnement, mais également pour modifier l'état interne de l'agent.
- **La Vue**, formée des interacteurs de l'agent, permet l'interaction (influence et perception) avec l'environnement, dans la dynamique en question. Les Vues étant reliées aux dynamiques, leur nombre ne dépend pas directement des plans.

5.2.2 Recomposition de l'agent sous la forme aMVC

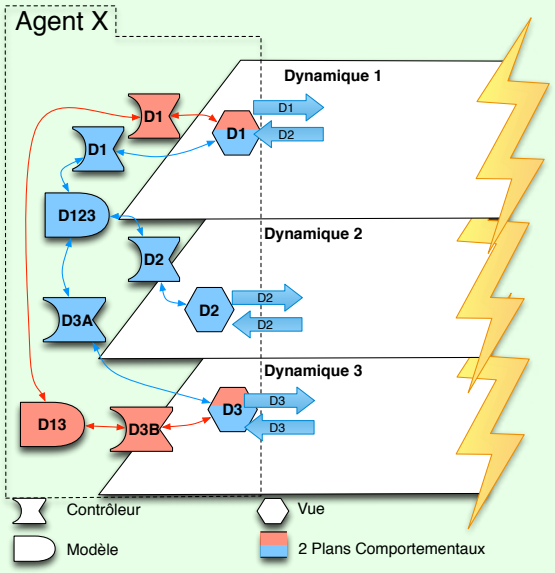
Nous avons établi trois fractionnements successifs pour décomposer notre agent trivial. Maintenant, recomposons ces éléments selon notre concept aMVC. Dans notre modélisation, chaque plan comportemental contient un Modèle, un Contrôleur et une Vue. En utilisant notre découpage précédent, notre agent est un élément composé de plusieurs composants aMVC, où chacun est lié à un plan comportemental. De cette façon, si nous prenons une couche aMVC pour chaque domaine, nous sommes en mesure de construire un agent aMVC comme sur la Figure 5.3. Nous y avons présenté deux possibilités pour la représentation de l'agent X :

- Dans le premier cas (Figure 5.3a), nous avons gardé la séparation obtenue en fin de processus. Cette séparation ne montre pas la connexité du graphe représenté par les agents. En effet, l'utilité forte des plans comportementaux est d'obtenir la connexité des constituants aMVC et donc de retrouver l'unité agent souvent invisible avec la seule décomposition sur les dynamiques.
- Dans le second cas (Figure 5.3b), nous avons proposé de fusionner les Modèles de même plan comportemental. Comme ils correspondent à des domaines d'expertise comportementale, en général un seul groupe d'experts s'occupe de cet ensemble. Ce regroupement évite la redondance de certaines informations du Modèle et simplifie sa mise en place. De plus, nous avons rassemblé les Vues dans la dynamique, en partant du principe qu'une même Vue peut être utilisée pour des plans comportementaux différents (comme la mâchoire du Loup pour manger, attaquer, mordiller...). Les règles de consistance permettent de garder la consistance des attributs partagés : lorsque le contrôleur $D1bleu$ modifie les attributs du modèle $D123$ (sur la partie 1), une répercussion sera faite sur le Modèle $D13$ (sur la partie 1).

Figure



(a) Agent aMVC avec la séparation obtenue en fin de processus



(b) Agent aMVC avec fusion des modèles par plan comportemental

FIGURE 5.3 – Fusion des Modèles de même plan comportemental pour un agent aMVC

En comparant la Figure 5.1a (qui est la représentation d'un agent en utilisant DOM uniquement) à la Figure 5.3 (représentant deux agents aMVC avec le complément de structuration proposé ici), nous obtenons une façon d'organiser les composants de l'agent facilitant la définition, l'utilisation et la réutilisation de celui-ci.

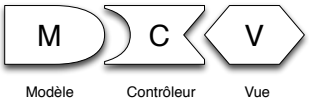
En reprenant notre formalisme établi dans le chapitre précédent, notre agent X aMVC sera noté :

$$
\begin{aligned}
Agent^X = \{ & \langle M^X_{D123}, C^X_{D1bleu} \rangle ; \langle C^X_{D1bleu}, V^X_{D1} \rangle ; \\
& \langle M^X_{D123}, C^X_{D2} \rangle ; \langle C^X_{D2}, V^X_{D2} \rangle ; \\
& \langle M^X_{D123}, C^X_{D3A} \rangle ; \langle C^X_{D3A}, V^X_{D3} \rangle ; \\
& \\
& \langle M^X_{D13}, C^X_{D1rouge} \rangle ; \langle C^X_{D1rouge}, V^X_{D1} \rangle ; \\
& \langle M^X_{D13}, C^X_{D3B} \rangle ; \langle C^X_{D3B}, V^X_{D3} \rangle ; \\
\} &
\end{aligned}
$$

Exemple

Dans notre exemple, fractionnons un agent Loup dans chaque plan comportemental, selon les trois composantes Modèle, Vue et Contrôleur. Ce fractionnement aboutit à la décomposition suivante suivante :

- Le plan comportemental émotionnel du loup (noté "**E**" en bleu) :
  - **Modèle :** (Niveau de…) Agressivité, Joie, Peur…
  - **Contrôleur :** Se divertir, Se reproduire…
  - **Vue :** Pattes, Mâchoire (pour la morsure amicale)…
- Le plan comportemental de survie du loup (noté "**Su**" en rouge) :
  - **Modèle :** (Niveau de…) Vie, Endurance, Faim, Soif…
  - **Contrôleur :** Attaquer, Appeler à l'aide, Patrouiller, Se nourrir…
  - **Vue :** Pattes, Mâchoire (pour la morsure de l'attaque)…
- Le plan comportemental social du loup (noté "**So**" en vert) :
  - **Modèle :** Appartenance à la horde (nom), Rang (Alpha, Oméga, Chef…), Compagne…
  - **Contrôleur :** Migrer, Chasser, Rechercher un partenaire…
  - **Vue :** Pattes, Mâchoire (pour communiquer)…



Dans les schémas, nous avons repris le formalisme graphique pour simplifier la représentation. Par exemple, M^X_{D123} sera représenté par sa forme et le signe $D123$, en ignorant le nom X de l'agent.

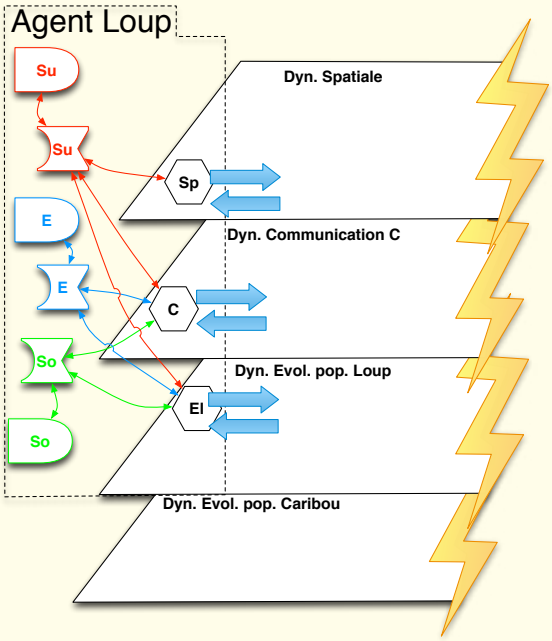
De manière générale, l'agent aMVC dans un cadre multidynamique intégre plusieurs instances des modules M, V et C. En posant que nous avons d dynamiques $\mathscr{D}_i$ et p plans comportementaux $\mathscr{P}_j$ par dynamique :

- À chaque $\mathscr{D}_i$ correspondent p plans comportementaux $\mathscr{P}_j$:
  - Pour chaque $\mathscr{P}_j$, nous obtenons deux couples $\langle M_{ij}, C_{ij}\rangle$ et $\langle C_{ij}, V_i\rangle$ (il est possible de n'en avoir qu'un dans le cas d'agents réactifs) :
    - un Contrôleur C_{ij} contenant les comportements de $\mathscr{C}$ en rapport avec ce $\mathscr{P}_j$ et $\mathscr{D}_i$.
    - un Modèle M_j des caractéristiques internes de l'agent en rapport avec ce plan comportemental.
    - une Vue V_i qui est l'ensemble des interacteurs de la dynamique $\mathscr{D}_i$.

Au maximum, nous obtenons un ensemble composé de $(d * p) * (d + p)$ couples, avec p Modèles, $d * p$ Contrôleurs, et d Vues : soit un total éventuel de $p + d * p + d$ modules.

Exemple

Dans notre simulation de loup et de caribou, un agent Loup aMVC est modélisé comme sur la Figure ci-dessous. Pour faciliter la compréhension, nous avons regroupé tous les Modèles d'un plan comportemental en un seul Modèle, tous les Contrôleurs en un seul. Par contre, nous avons gardé un seul ensemble d'interacteurs (Vue) par dynamique. En effet, que ce soit pour mordre en jouant ou en attaquant, le Loup utilisera le même effecteur : sa mâchoire.
Suivant le degré de finesse attendu, nous pouvons ne pas les regrouper et garder pour chaque dynamique ses sous-ensembles relatifs aux plans. Comme à chaque plan comportemental correspond un aMVC, notre agent Loup sera décomposé comme suit :



- Le plan comportemental émotionnel du loup ("**E**" en bleu).
- Le plan comportemental de survie du loup ("**Su**" en rouge).
- Le plan comportemental social du loup ("**So**" en vert).

5.3 Discussion sur l'agent aMVC appliqué à un cadre multidynamique

5.3.1 Discussion

Nous avons détaillé ici l'architecture agent aMVC pour le contexte DOM en montrant qu'elle fait intervenir un niveau supplémentaire de considération multicomportementale.

Cette modélisation commence par le bas (l'agent) et non par le haut (le système). Nous avons d'abord divisé l'environnement en dynamiques, ce qui a créé des sous-ensembles d'état et de comportements dans l'agent. Puis, nous avons découpé l'agent comme un "mille-feuilles" où chaque feuille est associée à un plan comportemental. Enfin, un dernier partitionnement a été fait en fonction des composantes dans chaque plan : le **Modèle** de l'agent, son **Contrôleur** et sa **Vue**.

Grâce à cette découpe, nous sommes en mesure de donner une couche comportementale à chaque expert, et, si nécessaire, de diviser encore plus le travail entre les différents experts grâce à la coupe aMVC, en donnant une partie de la couche (M, V ou C) à chacun. Ainsi, cette structuration facilite la création de l'agent dans un cadre collaboratif.

En outre, en utilisant la propriété du modèle aMVC, parmi d'autres avantages, nous sommes en mesure de réutiliser facilement et de personnaliser n'importe quelle partie de n'importe quelle couche comportementale. La séparation avancée dans aMVC favorise la réutilisation logicielle à plusieurs niveaux :

- **Modèle :** Un même état peut être la cible de plusieurs comportements différents
- **Vue :** Un même comportement peut avoir plusieurs interacteurs. Par exemple, le déplacement d'un animal imaginaire peut être décliné selon qu'il évolue dans des milieux différents : des ailes pour l'environnement aérien, des pattes pour l'environnement terrestre et des nageoires pour l'environnement aquatique. De plus, ces vues peuvent être réutilisées dans d'autres simulations ou pour d'autres agents du système.
- **Contrôleur :** Un comportement peut se manifester dans la même simulation chez des agents différents ou encore dans d'autres simulations du même domaine.

Nous sommes conscients de la complexité de l'architecture interne : il ne s'agit plus de manipuler un "simple" agent, mais un agent composé d'un très grand nombre de briques ($d * (2p+1)$), voir fin de Section 5.2.2. Cependant, nous pouvons aussi faire le parallèle avec la Programmation Orientée Objet, qui, elle aussi, entraîne la création d'un grand nombre de classes. L'approche MVC en **P**rogrammation **O**rientée **O**bjet (POO) génère plus de code que si nous nous en passons. Cela ne signifie pourtant pas que l'approche MVC est mauvaise : il s'agit de la conséquence d'une plus grande structuration du code. Autant pour une petite application, la programmation "en un seul bloc" contient moins de code et est beaucoup plus légère, mais ce n'est pas une solution à long terme. Le code selon l'approche MVC est conçu de manière à diminuer les coûts de maintenance, d'adaptation et de réutilisation. En POO, cette complexité est partiellement masquée lorsque nous utilisons des outils de haut niveau, des frameworks, qui permettent de faciliter sa mise en œuvre. Nous avons par cet intermédiaire une puissance d'expression importante, une flexibilité que n'a pas une programmation monolithique.

De la même manière, un framework adapté à notre modèle aMVC permettrait de contenir la complexité tout en gardant cette flexibilité et cette puissance d'expression, pour pouvoir

profiter pleinement des avantages du modèle aMVC. Un tel framework reposerait sur deux aspects :

- Une méthodologie adaptée pour la création de tels agents.
- Une plateforme capable de gérer de tels agents.

Le premier aspect fait l'objet du chapitre suivant. Le second aspect dépasse largement le cadre de travail de cette thèse centrée sur la modélisation et fera l'objet de travaux ultérieurs au sein de notre équipe de recherche. Cependant, nous proposons dans la suite de ce chapitre quelques caractéristiques d'une plateforme pouvant manipuler les agents aMVC.

5.3.2 Caractéristiques d'une architecture de plateforme supportant aMVC

Dans cette approche, nous avons proposé un triple fractionnement de l'agent. Une plateforme adaptée aux agents aMVC doit gérer, en plus du multidynamisme, ce type de structure. Et notamment :

- Plusieurs couches de Modèles (*CM*).
- Plusieurs couches de Contrôleurs (*CC*).
- Un ensemble de Vues reliées aux dynamiques.

Couches de modèles

Ce sont des ensembles d'états dynamiques liés à un domaine particulier (voir Figures 5.1b et 5.4). À chaque plan comportemental correspond un *CM*, sous-ensemble de données dynamiques (états des agents) qui évolue avec le temps en fonction des *CC* et des lois internes du plan comportemental (qui font la cohérence et la consistance des données).

Dans une simulation, rien n'est statique. L'effet d'une couche de contrôleurs CC_a sur celle des modèles CM_{a1} produit une nouvelle couche de données. Soit nous faisons évoluer la couche précédente, soit nous en créons de nouvelles CM_{a2}, et ainsi de suite des CM_{ai}, pour les réinjecter dans le système. Dans l'absolu, il est possible d'implémenter un système d'historique qui pourrait être utilisé pour les agents possédant une mémoire à long terme.

De plus, nous utilisons le Modèle actif (voir Section 4.3). Un des avantages est que d'autres sources peuvent changer le modèle. Par exemple, en dehors de ce modèle, dans le "logiciel", si quelqu'un voulait apporter un changement à l'ensemble des *CC* ou des *CM*, ce serait possible. Mais la raison la plus importante est que chaque couche de l'ensemble des *CM* a une vie propre ! En effet, étant donné qu'ils sont un sous-ensemble de données potentiellement dynamiques (états des agents) qui peuvent évoluer avec le temps en raison des *CC* et des lois internes, des changements se produisent par eux-mêmes sans interaction "extérieure".

Figure

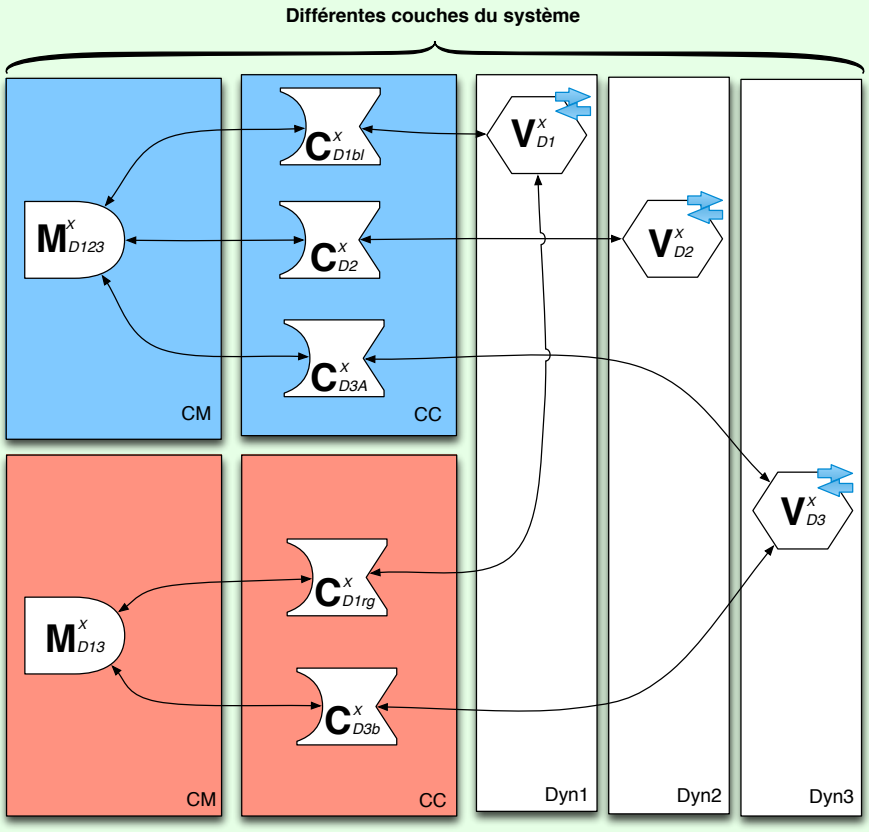


FIGURE 5.4 – Les différentes couches du système

Couches de contrôleurs

Un des avantages de l'utilisation des *CC* est que, dans un système complexe, les agents sont regroupés par familles (ou types). Chaque "type" se caractérise par un ensemble de comportements spécifiques : par exemple, le comportement d'un agent Loup Oméga sera le même que tous les autres agents Loup Oméga. En isolant ainsi les comportements des états et des interacteurs, nous pouvons factoriser les comportements et réduire la complexité du modèle.

Une couche de contrôleurs ne contient pas l'ensemble du comportement de l'agent, mais ceux liés à un plan comportemental du système complexe. Elle pourrait être définie par différentes méthodes ou formalismes connus, suivant les possibilités de la plateforme : logique, tableaux, machine de Turing, aUML... Ce ne sont que quelques exemples, mais nous pouvons utiliser un large éventail de méthodes de modélisation, selon la façon dont le modélisateur veut modéliser son système, étant donné qu'il s'agit de créer des briques élémentaires.

Vues

Les interacteurs représentent les canaux que possède un agent pour effectuer des influences et recevoir des perceptions dans une dynamique particulière. Ils doivent être gérés différemment. En effet, ils sont le lien qui permet à l'agent d'avoir une présence dans cette dynamique. Ils se positionnent directement dans le plan environnemental de la dynamique souhaitée.

Synthèse

En prenant l'exemple de la Figure 5.4, la décomposition se fera ainsi :

- 2 Couches de modèles : CM_{bleu} et CM_{rouge}.
- 2 Couches de contrôleurs : CC_{bleu} et CC_{rouge}.
- 3 Vues pour chaque agent : une Vue dans chaque dynamique.

Dans la Figure 5.4, nous avons choisi de simplifier la représentation en n'illustrant la décomposition que d'un seul agent. En réalité, chaque agent est décomposé selon les plans comportementaux correspondant à son type et est inclus dans la couche adéquate, comme le montre la Figure 5.5.

Figure

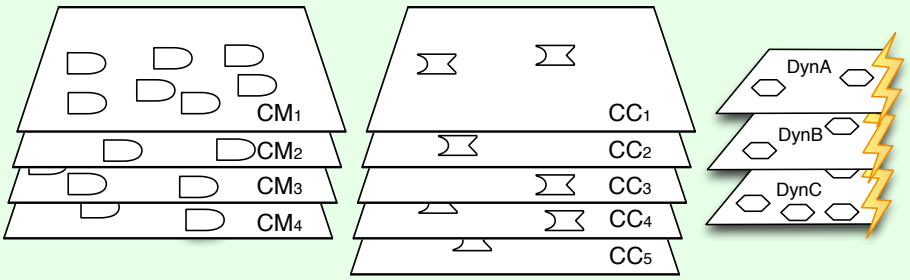


FIGURE 5.5 – Modélisation des couches dans le cas d'un système complexe

5.4 Conclusion

Dans le chapitre précédent, nous avons défini le concept initial de l'agent aMVC. Ici, nous avons appliqué ce modèle au cadre multidynamique. La décomposition qui en résulte nous a permis de compléter celle obtenue par la méthode DOM . Alors que cette dernière se focalisait sur les seules dynamiques de l'environnement, cette nouvelle décomposition est centrée sur les agents et leurs comportements.

Nous pouvons remarquer une continuité dans la réflexion sur le paradigme agent : les agents sont souvent utilisés lorsque nous cherchons à représenter un système complexe et que nous ne pouvons le faire de manière globale. Nous utilisons ainsi les agents pour décomposer ce système. Dans le concept aMVC que nous proposons, nous cherchons à représenter un agent difficile à appréhender de manière globale. Nous le décomposons par l'intermédiaire de briques élémentaires. La richesse induite par ce cadre implique la complexité constitutionnelle de l'architecture de l'agent.

Cependant, avec l'aide d'un framework adapté, cette complexité peut être maîtrisée, à l'instar des frameworks utilisant MVC en POO. À la fin de ce chapitre, nous avons proposé quelques caractéristiques d'une telle plateforme, en mettant en évidence les différentes couches et leurs relations. Dans le chapitre suivant, nous nous intéresserons à un volet complémentaire à cette proposition aMVC : une méthodologie pour mettre en place une SOA utilisant DOM et aMVC.

6 Cadre Méthodologique : La Modélisation MultiComportementale

Plan du chapitre

6.1 Introduction et contexte

Nous avons présenté dans le chapitre précédent une architecture ouvrant de nouvelles possibilités. Mais son potentiel ne peut s'exprimer pleinement que si son usage s'inscrit au sein d'une méthodologie cadrant les bonnes pratiques en vue d'une modélisation co-constructive efficace. Au vu du nombre rapidement important de sous-constituants générés par cette architecture pour un même agent, il nous semble important de proposer une approche méthodologique de conception appropriée à ce nouveau cadre de modélisation [Gangat *et al.*, 2013]. Ainsi le processus décrit dans ce chapitre a pour but de faciliter l'utilisation des concepts précédents de **a**gent **MVC** (aMVC) dans un cadre pluridisciplinaire et multidynamique.

Cette approche méthodologique permet d'avoir une représentation d'un système complexe, suivant le paradigme agent. Elle se fait par une décomposition par des éléments à la fois simples et fondamentaux pour décrire le système, sous forme de tableaux synoptiques. Elle découpe le système en cellules pour obtenir une vision thématique des agents et explorer les divers aspects de leurs comportements. Ensuite, nous modélisons l'agent comme étant un agrégat de ces cellules. Cette démarche aide à la modélisation du système complexe étudié.

Parallèlement, cette approche permet de codifier, de traduire et d'intégrer la connaissance provenant de disciplines différentes dans le modèle. Elle offre aussi une représentation du modèle, permettant sa cohérence et sa consistance malgré la pluridisciplinarité, avec différents niveaux d'abstraction possible. Elle aide à faire le lien entre les différents thématiciens, ce qui facilite la co-construction de modèles. Enfin, elle donne la possibilité à tous les acteurs de travailler en synergie, par l'intermédiaire des supports communs que forme cette décomposition élémentaire.

6.2 Tronc commun aux différentes méthodologies de conception de simulations

Dans la Section 2.2, nous avons donné la définition ainsi que des exemples de méthodologies de conception de simulations. La grande majorité des auteurs sont d'accord : la conception de simulations se fait par un processus particulier, mais il n'existe pas de consensus autour du détail de celui-ci.

L'expression "Tronc commun" insiste sur le fait qu'il s'agit d'un ensemble de phases communes à plusieurs méthodologies. Cette trame est basée sur des travaux précédents présentés au Chapitre 2. Elle nous permettra de positionner nos contributions présentées dans la Section 6.3 suivante. Ce tronc commun comporte les cinq phases de la Figure 6.1 :

1. **Une phase de modélisation initiale** : Les thématiciens, aidés par les modélisateurs, partent du système complexe réel et établissent un modèle de domaine. En général, il s'agit d'une modélisation grossière qui s'affine et gagne en pertinence au fur et à mesure.
2. **Une phase de formalisation** : Les modélisateurs, accompagnés des thématiciens, élaborent le modèle de conception. Ils adaptent les connaissances des experts au monde

agent, passant du langage non-formel à un langage formalisé "agent".

3. **Une phase d'opérationnalisation** : Les deux précédents groupes collaborent maintenant avec des modélisateurs-informaticiens[1], qui procèdent à une phase d'opérationnalisation. Elle consiste à exprimer les agents avec des propriétés en relation avec l'implémentation (sans toutefois implémenter le modèle). Il peut s'agir de propriétés techniques comme la distribution physique des agents ou de techniques d'ordonnancement temporel. En d'autres termes, les modélisateurs-informaticiens établissent avec les thématiciens et modélisateurs un "cahier des charges".
4. **Une phase d'implémentation** : Les programmeurs, à partir du "cahier des charges" précédent, implémentent la simulation sur la plateforme choisie, avec ses concepts spécifiques et réalisent les tests de validation.
5. **Une phase d'exécution** : Une fois le modèle informatique implémenté, les utilisateurs procèdent à une succession d'expériences sur la base de scénarios de simulation. Ces utilisateurs sont les thématiciens, les responsables du projet...

Dans cette trame, nous avons volontairement omis les cycles et retours ainsi que les étapes d'analyses. La principale raison est que, dans notre cadre de travail, nous nous intéressons particulièrement à la co-construction de modèle et à sa réutilisation. De plus, l'accompagnement d'experts en SOA est un point important à chaque phase. Ceux-ci aident à avancer de manière efficace et à employer un niveau de dialogue adapté à chacun et à ce que l'on veut produire pour une modélisation de SOA.

6.2.1 La collaboration entre les différents acteurs

Dans un projet de modélisation de comportement des tortues marines *Chelonia mydas* de la zone sud-ouest de l'Océan Indien [Gangat *et al.*, 2010, Dalleau *et al.*, 2012], nous avons abordé le sujet de la collaboration entre les différents acteurs participant à un projet de modélisation commun en partie, en structurant notre approche sur la dualité : modèle conceptuel / modèle informatique. En nous référant aux phases principales de conception de simulations identifiées par Ralambondrainy (voir Section 2.2.2), nous avons établi un prototype en travaillant sur deux documents en parallèle :

1. La description conceptuelle, qui énonce le modèle suivant ce que les thématiciens souhaitent exprimer.
2. La description informatique, qui spécifie le modèle tel qu'il est (ou sera) implémenté sur le prototype.

La distance entre ces deux documents est appelée distance implémentatoire. La réduction de cette distance impose des modifications sur les deux documents, donc des efforts de la

[1] Dans la Section 2.2.1 nous avons expliqué la différence entre modélisateurs-informaticiens et programmeurs. Le premier établit son modèle indépendamment de la manière que ce dernier sera implémenté. Le second s'appuie, lui, sur un langage de programmation ou une plateforme de simulation particulière.

Figure

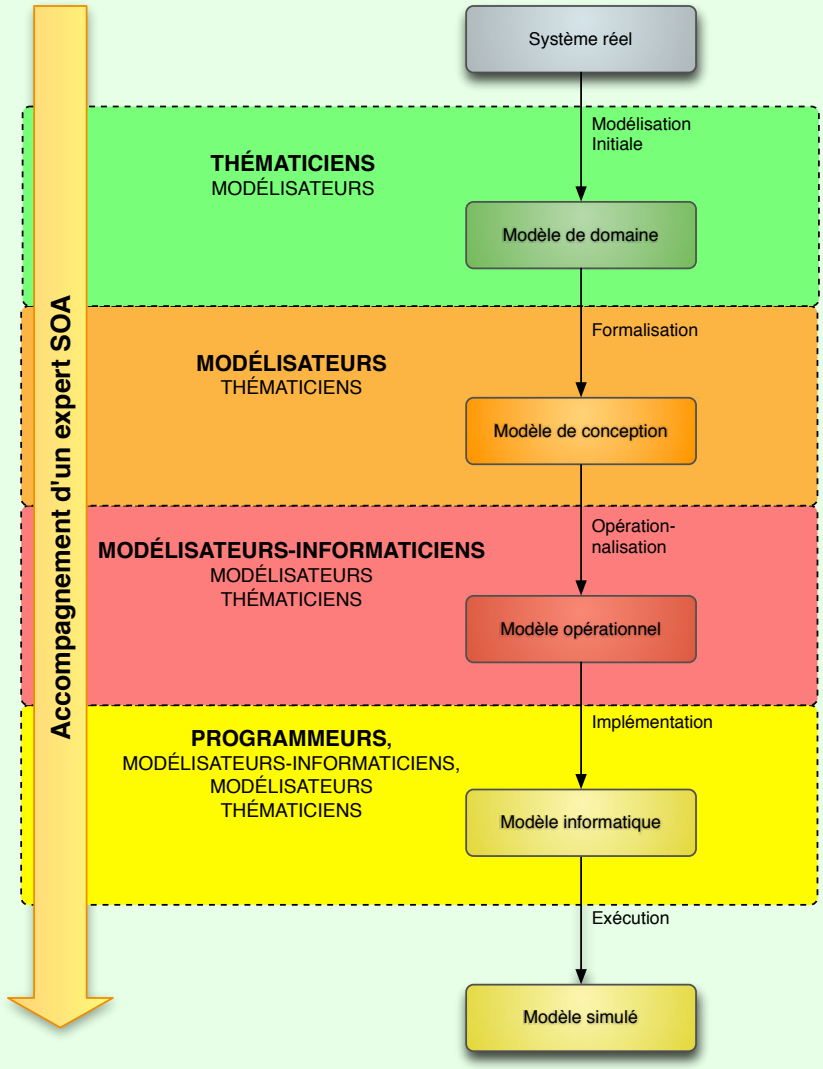


FIGURE 6.1 – Tronc commun aux méthodologies de conception de SOA

part de chacun des acteurs (pour ce projet, il s'agissait de thématiciens, de modélisateurs-informaticiens et de programmeurs). La description conceptuelle doit être réexprimée (voire simplifiée) pour être en adéquation avec les contraintes (et les limites) informatiques. La description informatique doit être améliorée pour se rapprocher au mieux des attentes des thématiciens.

Au final, plus la distance implémentatoire sera faible, plus le risque d'un manque de pertinence du prototype le sera aussi. Dès que la minimisation de ce risque sera jugée acceptable, l'effort du passage du prototype à l'outil final sera envisagé. Cette distance ne peut être réduite que lorsque chacun des intervenants travaille en synergie avec les autres. De plus, la présence d'un expert de la SOA demeure importante pour aider à la modélisation. Ce constat reste le même à chaque niveau, car la source de cette connaissance à laquelle nous devons nous référer suit une certaine hiérarchie :

- Les thématiciens, avec les modélisateurs, créent un modèle de domaine.
- Les modélisateurs, avec les thématiciens, mettent en place le modèle de conception.
- Les modélisateurs-informaticiens, travaillant surtout avec les modélisateurs et parfois les thématiciens, établissent un modèle opérationnel.
- Les programmeurs, aidés des modélisateurs-informaticiens, et parfois des modélisateurs et thématiciens, implémentent un modèle informatique.

Dans cette hiérarchie, pour chaque étape, le premier intervenant cité (en gras dans la Figure 6.1) joue le rôle prépondérant ; ceux qui suivent tiennent le rôle d'accompagnateurs. Par exemple, les modélisateurs-informaticiens établissent le modèle opérationnel, mais ils ont besoin de l'aide des modélisateurs et thématiciens aussi. Ils apportent leur concours tout au long de cette adaptation progressive du modèle : la mise en place d'un modèle de niveau n implique parfois un ajustement supplémentaire par rapport au modèle précédent $n-1$. Les acteurs du degré $n-1$ doivent donc ajuster leur propre modèle et vérifier la correspondance avec ceux du degré $n-2$. Et ainsi de suite...

Cette collaboration essentielle permet une meilleure modélisation. En particulier, la présence des experts dans le processus de modélisation-simulation est un élément-clef. Mais elle ne doit pas se limiter à la réalisation commune de quelques étapes. L'ensemble du processus de modélisation doit se faire en étroite collaboration. Suivant ce principe, la méthodologie que nous proposons permettra à tous les acteurs de travailler et de communiquer sur la base d'un cadre commun. Il est évident qu'il s'agit là d'une répartition théorique des rôles, mais souvent le même groupe de personnes représente les modélisateurs-informaticiens, les programmeurs et les experts en SOA. Cependant, il reste préférable de bien définir les rôles pour faciliter la création des différents modèles dans les contextes les plus généraux possible. Ce tronc commun nous permettra de mieux visualiser notre méthodologie qui facilitera l'accompagnement et la collaboration des acteurs du projet.

6.3 La Modélisation MultiComportementale

Dans un système complexe, les entités présentes interagissent par des actions en rapport avec leurs comportements. D'après la neuvième édition du dictionnaire de l'Académie française, le comportement est caractérisé par "*les activités des êtres vivants et leurs réactions physiologiques aux conditions de leur milieu*". Les comportements traduisent en psychologie

"*l'ensemble des réactions objectivement observables d'un sujet, d'un organisme qui répond à une stimulation.*". Dans ce chapitre, nous considérons la définition suivante :

Définition 16

Comportement : Le comportement d'un agent représente ses actions pour parvenir à l'objectif fixé.

L'approche méthodologique de **Modélisation MultiComportementale** exposée dans ce chapitre est une démarche systématique, qui permet de décomposer le thème d'étude en tâches simples. Elle est structurée en deux axes liés : comportements et actions. Les descriptions obtenues et représentées sous forme de tableaux se répartissent conformément à ces deux axes (voir Figure 6.2). Chacune de ces étapes, listées de A à F, sera décrite et expliquée dans la suite de cette Section.

Figure

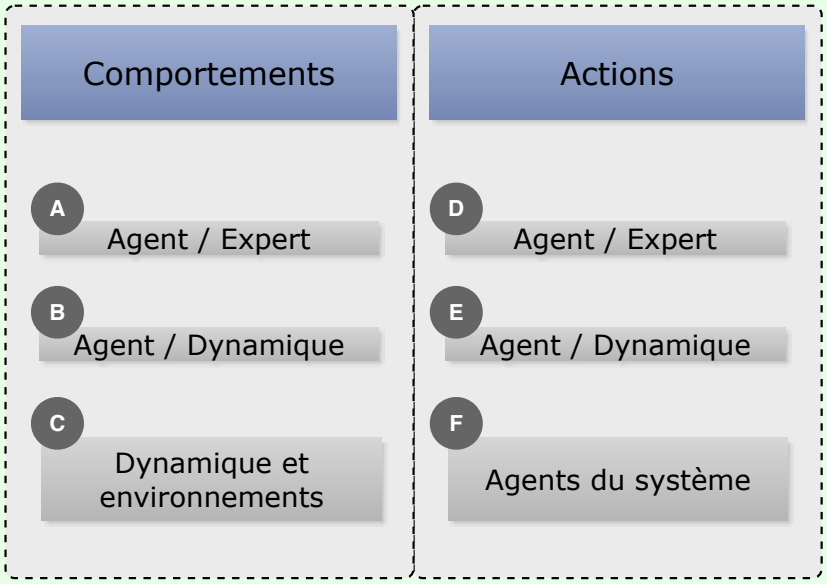


FIGURE 6.2 – Les deux axes de la méthodologie de la Modélisation MultiComportementale

Pour plus de clarté, tout au long de ce chapitre, de manière générale, le mot "agent" correspond à une **famille (ou un type) d'agents**, *c'est-à-dire* une spécification d'agents possédant les mêmes comportements et actions. Nous appelons **instance d'agent** une instance particulière de cette famille d'agents. Par exemple, nous avons le type agent "Loup" et les instances

Tableau

TABLE 6.1 – A : Les agents et les experts (Comportements)

Exp. / Ag.	Expert 1	Expert 2	Expert 3	(...)
Agent A	Comportements 1, 2, 3	✗	Comportement 1	...
Agent B	✗	Comportements 3, 4, 5	Comportement 4	...
Agent C	✗	✗	Comportements 6, 7, 8	...
Agent D	✗	✗	*Passif*	...
...	...	...	...	...

Ordre de priorité des comportements de l'agent A : 1,2,3
Ordre de priorité des comportements de l'agent B : 4,5,3
Ordre de priorité des comportements de l'agent C : 7,6,8
Légende :
✓ (ou texte correspondant) = participe
✗= ne participe pas

d'agent $loup_1$, $loup_2$, *etc.* Dans cette partie, nous décrirons les étapes comme étant une suite séquentielle, mais nous verrons au chapitre 7 que certaines tâches peuvent être menées en parallèle.

6.3.1 Description sous forme de comportements

La première série d'étapes consiste à décrire le système complexe au travers des divers comportements de chaque type d'agent.

Étape A : Identification des comportements du modèle de domaine

D'abord, nous demandons aux thématiciens (et modélisateurs) d'identifier les différentes familles d'agents. Ces dernières sont classées en deux grands types : les agents actifs (qui possèdent au moins un comportement) et les agents passifs (qui n'en possèdent aucun). De manière générale, un "objet" du monde réel qui n'a pas de but à atteindre au sein du système multi-agents est un agent passif.

Ensuite, pour chaque agent, chaque expert exprime les différents comportements qu'il peut décrire (voir le tableau 6.1). Un ordre de priorité par défaut des comportements est défini pour chaque agent. Cette priorité peut changer en fonction des besoins de la simulation, grâce

à un algorithme interne à l'agent. Par exemple[2], le comportement de plus haute importance pour un loup affamé est la chasse ; celle de l'animal rassasié est la reproduction. Dans ce cas, une partie de cet algorithme est centrée sur l'état "faim" de l'agent loup.

Ainsi, chaque expert apporte sa contribution à son niveau sur ces comportements :

- On peut rencontrer un comportement possédant une description commune pour deux agents différents : par exemple, le comportement de mouvement aléatoire peut être le même pour un termite et une fourmi.
- Deux descriptions différentes pour le même comportement d'un agent peuvent être définies par deux experts : par exemple, le comportement de mouvement aléatoire peut être défini par un expert en changeant uniquement de direction et par un second en modifiant à la fois vitesse et orientation.
- Un expert peut décrire intégralement le comportement comme n'en présenter qu'une partie (voir l'exemple plus loin).

Résumé de l'étape A

A : Identification des comportements (par les experts) :

- Choisir un agent. S'il s'agit d'un nouvel agent :
  - Description de l'agent.
  - Ajout à une liste des types d'agents.
  - Définition du statut actif de chaque agent : il est actif[3] lorsqu'il possède au moins un comportement, sinon il est passif.
- Décrire un comportement :
  - Description en langage naturel du comportement. Il est possible d'avoir :
    - Une description commune pour deux agents différents.
    - Deux descriptions différentes pour un même comportement A_n par des experts différents i et j : A_nEx_i et A_nEx_j.
    - La description partielle d'un comportement.
  - Ajout du comportement à la liste des comportements de l'agent.
  - Définition de la priorité du comportement.
- Définir l'algorithme de changement de priorité, si besoin est.

Résultat : Liste des agents, Listes des comportements dans le modèle de domaine et Algorithme de priorité.

[2] Il s'agit bien sûr d'un exemple totalement fictif.

[3] **Note :** Il est possible qu'un agent soit actif dans une dynamique, mais passif dans une autre.

Exemple

Reprenons l'exemple du chapitre précédent où nous souhaitons simuler plusieurs populations (loups et caribous), pour étudier leur évolution, avec l'ingérence des hommes.

Le premier tableau (ci-dessous) définit les comportements des agents. Nous avons trois experts différents : les deux premiers le sont dans leurs domaines bien précis (loup et caribou), le troisième possède une compétence approfondie dans le domaine de l'attaque des loups. Focalisons-nous sur l'agent Loup pour faciliter la compréhension. L'expert Loup peut exprimer de manière générale comment un loup chasse : recherche de proies, communication avec ses congénères, attaque. L'expert prédateur peut énoncer l'attaque avec une granularité plus fine : le déroulement du combat, le comportement du loup lorsque le caribou riposte... Bien que ce niveau de détail ne soit pas nécessaire pour le but fixé par cet exemple fictif, nous l'ajoutons à notre modélisation pour mettre en avant l'importance de bien définir comportements et actions.
Ici nous avons deux experts différents qui ont deux visions distinctes du même comportement.

Enfin, les experts définissent un algorithme de changement de priorité suivant l'état de satiété du loup qui est défini pour permettre le fonctionnement de l'algorithme.

Exp. / Agents	Expert Loup	Expert Caribou	Expert Prédateur	...
Agent Loup	Chasse (général) Migration ...	✗	Chasse (attaque)	
Agent Caribou	✗	Comportements X, Y ...	Combat	
Agent Homme	✗	✗	✗	...

Ordre de priorité des comportements de l'agent Loup : Chasse, Reproduction, Migration.
Algorithme de changement : "Tant que le niveau de faim du loup est supérieur à la moitié du niveau de satiété, il chassera sinon il va se reproduire."

Étape B : Déclinaison et formalisation des comportements du modèle conceptuel

Suite à cette première analyse, les modélisateurs doivent décliner chaque comportement avec le concept d'actant, en utilisant en particulier les notions d'acteur et d'objet. Cette sémantique permet ensuite de distinguer l'appartenance à une dynamique. En linguistique, le terme d'**actant** (Lucien Tesnière, 1965, dans [Tesnière, 1965]) désigne les constituants syntaxiques d'une phrase par opposition au circonstant (manière, repère temporel, lieu de l'action). Le verbe renvoie à un processus (que nous associons à un comportement). Chaque actant impliqué dans ce processus peut jouer le rôle sémantique suivant :

- **acteur** : celui qui agit ;
- agent : celui par qui le processus est accompli ;
- **objet/patient** : celui qui subit le processus ;
- bénéficiaire : celui qui reçoit les résultats du processus ;
- instrument : ce qui permet le processus ;

Par exemple, dans "Pierre donne un cadeau à Sophie [dans la rue]", nous avons :

- actant 1 : Pierre = **acteur** = sujet ;
- actant 2 : un cadeau = **objet** patient = objet direct ;
- actant 3 : Sophie = bénéficiaire = objet indirect.
- circonstant : dans la rue = complément circonstanciel de lieu, qui est subsidiaire et peut être ôté de la phrase.

Ici ce sont les concepts d'**acteur** et d'**objet** qui nous intéressent : par définition, l'acteur du comportement sera l'agent lui-même. L'objet du comportement peut être soit vide, soit l'agent lui-même (la même instance), soit un ou plusieurs autres agents (une autre instance ou un autre type d'agent, passif ou non). Par exemple, dans une simulation de termite, même si l'agent bois est passif, selon la description de l'expert, le comportement de manipulation de l'agent termite fait que l'agent bois est manipulé par le termite et dépend donc de ce comportement : **Agent termite** → *Manipulation.acteur* et **Agent bois** → *Manipulation.objet*.

En se posant la question : "Quelles sont les différentes dynamiques inhérentes à cette simulation, étant donné ce comportement ?", les modélisateurs obtiennent alors ces relations entre agents et dynamique, représentées le tableau 6.2 qui explicite dans quelles dynamiques chaque type d'agent exerce potentiellement une influence ou effectue une perception, au travers des comportements (plus précisément de leurs déclinaisons) et de sa présence.

D'une part, la sémantique des actants permet de garder une cohérence pour l'attribution d'une dynamique à un agent : par exemple, *Manipulation.acteur* et *Manipulation.objet* impliquent que l'agent termite et l'agent bois, respectivement, entrent en jeu dans la dynamique spatiale. Un agent peut donc participer à une dynamique sans avoir de comportement propre :

Tableau

TABLE 6.2 – B : Les agents et les dynamiques associées (Comportements)

Dyn. / Agents	Dynamique 1	Dynamique 2	Dynamique 3	...
Agent A	Comportements 1, 2	✗	Comportements 1, 3	...
Agent B	✗	Comportements 4, 5	Comportements 3	...
Agent C	Comportements 6, 7, 8	Comportements 9.objet	✗	...
...	...	...	...	...

En tant qu'**Agent A acteur Comportement 1**, ma fonction **active** est de **faire telle et telle chose**, afin d'**atteindre tel but**.
...

un objet inerte – un ballon de football – est un agent passif qui intervient dans une dynamique spatiale, sans pour autant avoir de comportement.

D'autre part, il est possible qu'un comportement fasse partie de plusieurs dynamiques différentes. Par exemple, le comportement de chasse (et la déclinaison chasse.acteur notamment) impose à la fois une intégration dans la dynamique spatiale, pour se mouvoir, et celle de la communication, pour appeler ses congénères. Ce comportement participe à ces deux dynamiques chez le loup. Ici l'expertise du thématicien ou d'un groupe de thématiciens du même domaine représente un plan comportemental, car le comportement proposé par un expert peut être scindé en plusieurs dynamiques.

La suite de cette étape est la formalisation de ces déclinaisons selon la forme : "*En tant qu'**<agent> acteur/objet** du **<comportement>**, ma fonction **active/passive** est de **<besoin>**,afin de **<but>**, au travers des dynamiques **<ensemble de dynamiques>***". Cette formalisation est inspirée des "*User Stories*" de Scrum[4]. Elle permet d'enlever les ambiguïtés possibles [Cohn, 2004]. En effet, les "users stories" ont été créées pour être compréhensibles par tous les acteurs d'un projet et incitent leurs auteurs à donner plus de détails. De plus, dans la méthodologie Scrum, elle permet le développement itératif du projet.

Résumé de l'étape B

B : Déclinaison et formalisation des comportements (par les modélisateurs) :

[4] Scrum est une méthode agile, issue des méthodes incrémentales qui permettent de maîtriser une production planifiée.

- Déclinaison des comportements selon acteur/objet si besoin est.
- Pour chaque déclinaison, création et/ou association à une ou plusieurs dynamiques.
- Formalisation de ces déclinaisons selon la forme : "*En tant qu'***<agent> acteur/objet** *du* **<comportement>**, *ma fonction* **active/passive** *est de* **<besoin>**,*afin de* **<but>**, *au travers des dynamiques* **<ensemble de dynamiques>**." et ajout à la description.

Résultat : Liste des comportements formalisés (et de leurs déclinaisons) du modèle conceptuel, Liste des dynamiques. Pour chaque dynamique nous avons aussi la liste des agents actifs (*c'est-à-dire* acteurs d'un comportement au moins dans la dynamique correspondante) et celle des agents passifs (*c'est-à-dire* uniquement objets d'un ou plusieurs comportements de la dynamique correspondante).

Exemple

Dans ce même exemple, le second tableau (ci-dessous) permet de définir la relation entre les agents et les dynamiques de la SOA. Celles-ci sont au nombre de trois :

- La dynamique de l'espace : les agents ont une position localisée et interagissent avec l'environnement et d'autres agents possédant un corps dans cet espace.
- La dynamique de communication : ici, les animaux communiquent.
- La dynamique de l'élevage, plus complexe : les hommes élèvent les caribous, les protègent...

Nous déclinons chaque comportement selon le concept acteur/objet lorsque c'est nécessaire. Par exemple, chasse.acteur correspond à une déclinaison pour l'agent Loup alors que chasse.objet correspond à celle pour le Caribou.

Dyn. / Agents	Dynamique Espace	Dynamique Communication	Dynamique élevage	...
Agent Loup	Chasse Migration	Chasse Migration	✗	
Agent Caribou	Se nourrir ...	...	...	
Agent Homme	...	...	...	

En tant que **Loup acteur** de **la chasse**, ma fonction **active** est de **chercher un caribou, le poursuivre et m'en nourrir**, afin de **satisfaire ma faim**...

Étape C : Bilan des dynamiques du système du modèle opérationnel

Suite à cette première série d'étapes, les équipes ont défini la liste des types d'agents de la SOA souhaitée. Pour chaque agent, la liste des comportements et de leurs déclinaisons accompagnées de leurs priorités a été définie. De même pour la liste des dynamiques où interviennent ces agents et leurs déclinaisons.

À l'issue de ces étapes, un premier bilan (voir tableau 6.3) définit les dynamiques du système, ainsi que :

- les environnements nécessaires à la SOA, relatifs à ces dynamiques,
- les agents qui interviennent dans ces dynamiques.

*Ici, nous utilisons la méthodologie **M**odélisation **O**rientée **D**ynamique (DOM) (voir Section 3.2) qui, en quelques mots, consiste à avoir recours à un environnement pour chaque dynamique. Même si nous ne l'employons pas directement ici, nous établissons un tableau du même type qui permet de définir l'(les) environnement(s), dans lequel (lesquels) les agents possèdent un corps et peuvent exercer des influences ou avoir des perceptions.*

Ce tableau permet de détecter les environnements utiles à concevoir, ainsi que les synergies envisageables entre les différentes dynamiques. Il est bien sûr possible d'intégrer plusieurs dynamiques au même environnement, comme de concevoir une dynamique agissant sur plusieurs environnements. Chacun d'entre eux, les modélisateurs-informaticiens le décrivent selon les besoins des experts, en se basant par exemple sur quelques propriétés proposées par Russel et Norvig [Russell et Norvig, 2003] :

- **Observable** (entièrement ou partiellement) *vs* **Inobservable** : Un environnement est observable (ou encore accessible) lorsque les interacteurs d'un agent donnent accès à l'état total de l'environnement. Lorsque celui-ci est partiellement observable, un champ d'accessibilité par rapport à la position de l'agent est déterminé dans cet environnement, en utilisant une métrique choisie.
- **Déterministe** *vs* **Non déterministe** *vs* **Stochastique** : Un environnement est déterministe si son prochain état est complètement déterminé par l'état courant et par les actions effectuées par les agents.
- **Épisodique** *vs* **Séquentiel** : Un environnement épisodique est tel que ses prochaines évolutions ne dépendent pas des états ni des actions précédents.
- **Dynamique** (semi-dynamique) *vs* **Statique** : Un environnement est dynamique pour un agent s'il évolue pendant que ce dernier décide quelle action accomplir.
- **Discret** *vs* **Continu** : Un environnement est discret lorsqu'il y a un nombre limité de perceptions et d'actions distinctes possibles.

Une fois ces environnements décrits, pour chacun d'entre eux, il est encore important d'y adjoindre deux informations :

Tableau

TABLE 6.3 – C : Les dynamiques et environnements du système

	Dynamique 1	Dynamique 2	Dynamique 3	...
Environnements	Env. α	Env. β	Env. γ	
Agents	A, C	B	A, B, C	

- La liste des lois de cet environnement : les différentes règles responsables de la dynamique du système comme l'évolution des attributs, les primitives de l'environnement, les règles de métrique...
- La liste des "attributs induits" par cet environnement. Le fait qu'un agent ait un corps dans un environnement implique certains attributs. Par exemple, le fait que l'agent Loup participe à la dynamique spatiale implique que ce dernier a un corps dans l'environnement spatial. Avoir un corps dans l'espace présuppose donc une position (x et y, voire z), une vitesse (nulle dans le cas d'objet immobile) et une orientation.

Exemple

Le premier bilan donne le tableau suivant. Dans notre exemple, nous avons décidé que l'agent Loup n'intervient pas directement dans la dynamique de l'élevage. Cette information, bien qu'évidente ici, est pratique dans le cas de SOA requérant de nombreux agents différents. Pour chaque environnement, les experts en établissent les caractéristiques (type, lois et attributs induits).

	Dynamique Espace	Dynamique Communication	Dynamique Élevage	...
Environnements	Environnement Spatial	Environnement de messagerie	Environnement Spatial	
Agents	Loup, Caribou, Homme	Loup, Caribou, Homme	Caribou, Homme	

Résumé de l'étape C

C : Bilan des dynamiques (par les modélisateurs-informaticiens) :

- Identifier les environnements à associer aux dynamiques :
  - Décrire le type d'environnement
  - Énoncer les lois de l'environnement (métrique...).
  - Créer des "attributs induits" par l'environnement pour un agent (vitesse, position sociale...).
- Identifier les agents à associer aux dynamiques : pour chaque agent, il faut établir la liste des déclinaisons de comportement possibles selon le modèle précédent.

Résultat : Liste des agents et des environnements associés aux dynamiques.

6.3.2 Description sous forme d'actions

Une fois les types d'agents, les comportements, les dynamiques et les environnements identifiés, intéressons-nous aux actions. Comme nous l'indique la définition 16, l'agent possède des comportements formés d'actions. Procédons donc à un affinement des comportements en actions.

Étape D : Identification des actions du modèle de domaine

Les thématiciens établissent le détail des comportements de chaque agent sous la forme d'un ensemble d'actions. Chaque expert décrit de manière détaillée les comportements relatifs à ce comportement particulier (qui peut parfois être commun à plusieurs agents). Chaque comportement se décompose en une ou plusieurs actions. Il est possible que deux experts aient des visions différentes de la même action. La mise en commun de ces connaissances donne les informations du tableau suivant 6.4, extension du tableau obtenu à l'étape A (tableau 6.1).

Cette identification permet :

- de connaître les actions liées à un comportement précis (d'un ou plusieurs agents),
- de reconnaître exactement les actions que chaque expert peut isoler.

Dans l'ordre proposé ici, l'étape D suit l'étape C. Cependant, il est possible qu'elle ait lieu juste après l'étape A ou encore après la description d'un comportement de l'étape A. En effet, la description de ces actions ne nécessite pas les étapes B et C, mais uniquement l'étape A pour le comportement concerné.

Tableau

TABLE 6.4 – D : Les agents et les experts (Actions)

Experts / Agents	Expert 1	Expert 2	Expert 3	…
Comportement 1 $\mathscr{C}_1$ (Agent A)	Ac_1Ex_1 Ac_2Ex_1	✗	Ac_1Ex_3	
Comportement 2 $\mathscr{C}_2$ (Agent A)	Ac_3Ex_1	✗	✗	
…				
Comportement 3 $\mathscr{C}_3$ (Agent A, B)	Ac_7Ex_2 Ac_8Ex_2	Ac_7Ex_2 Ac_8Ex_2	✗	
…				

Exemple

La suite de notre exemple aboutit au tableau ci-après. Les trois experts déclinent chaque comportement selon une ou plusieurs actions. Focalisons-nous sur un comportement particulier : celui de la chasse de l'agent Loup.
L'expert du loup scinde le comportement de chasse à décrire en quatre actions. L'expert des prédateurs, lui, décrit l'attaque en détail : les loups attaquent leur proie en la mordant par derrière, pour la fatiguer et ainsi mieux lui porter le coup de grâce.

Experts / Agents	Expert Loup	Expert Caribou	Expert Prédateur	…
Chasse (Agent Loup)	Ac_1 :Localisation de la proie Ac_2 :Poursuite Ac_3 :Attaque Ac_4 :Partage	✗	Ac_3 :Attaque	
Migration (Agent Loup)	Ac_6	✗	✗	
…				
Repos (Agent Loup, Agent Caribou)	Ac_9	Ac_{10} Ac_{11}	✗	
…				…

Tableau

TABLE 6.5 – E : Les agents et les dynamiques (Actions)

Dyn. / Comp.	Dynamique 1	Dynamique 2	Dynamique 3	…
Comportement 1 $\mathscr{C}_1$ (Agent A)	Ac_1D_1 Ac_2D_1	✗	Ac_1D_3	
Comportement 2 $\mathscr{C}_2$ (Agent A)	Ac_3D_1	✗	✗	
…				
Comportement 3 $\mathscr{C}_3$ (Agent A, B)	✗	Ac_7D_2 Ac_8D_2	✗	
…				

Résumé de l'étape D

D : Identification des actions (par les thématiciens) :

- Choisir un comportement et lister les actions.
- Décrire chaque action :
  - Description en langage naturel.
  - Ajout de l'action à la liste.

Résultat : Liste des actions du modèle de domaine de chaque comportement pour chacun des agents.

Étape E : Formalisation des actions du modèle conceptuel

Cette étape consiste en la formalisation des actions sous forme d'algorithme, voire de logique de premier ordre si nécessaire. Puis ces actions se répartissent selon les dynamiques. Autant il était possible qu'un comportement puisse faire partie de plusieurs dynamiques différentes, autant il est préférable que chaque action participe au moins de dynamiques possible, pour permettre ensuite la décomposition élémentaire. Ainsi nous obtenons les dépendances actions-dynamiques qui existent pour chaque comportement pour chacun des agents, comme sur le tableau 6.5.

Résumé de l'étape E

E : Formalisation des actions (par les modélisateurs) :

- Formaliser chaque action sous forme d'algorithme et/ou de logique de premier ordre (suivant la nécessité).
- Associer chaque action à la dynamique correspondante

Résultat : Liste des actions formalisées de chaque comportement pour chacun des agents.

Exemple

Encore une fois, on se focalisera sur l'agent Loup, plus particulièrement sur le comportement de chasse, décrit précédemment par deux experts différents. D'abord les modélisateurs formalisent les actions de ce comportement.

Ensuite, nous constatons que les actions liées à la chasse exercent des influences et perceptions sur la dynamique de l'espace. Les actions Ac_1 (Localisation de la proie) et Ac_4 (Partage de la proie) contribuent à la dynamique de communication : les loups, lors d'une chasse collective, communiquent entre eux pour localiser leur proie ainsi qu'au moment du partage.

Dyn. / Comp.	Dynamique Espace	Dynamique Communication	Dynamique Élevage	…
Chasse (Agent Loup)	Ac_1 : Localisation de la proie Ac_2 : Poursuite Ac_3 : Attaque Ac_4 : Partage	Ac_1 : Localisation de la proie Ac_4 : Partage	✗	
Migration (Agent Loup)	Ac_6	Ac_6	✗	
…				
Repos (Agent Loup, Agent Caribou)	Ac_9	Ac_9	✗	
…				

Étape F : Bilan des agents aMVC du système du modèle opérationnel

Le second bilan (voir tableau 6.6), obtenu à partir des connaissances précédentes, est centré sur les agents. Il permet de préparer les agents pour l'architecture aMVC (voir Chapitre 5). Elle sépare les trois composantes que nous avons identifiées sur un agent :

- **Les modèles M** : Son état, ou encore l'ensemble des attributs qui caractérise un agent. Nous utilisons comme notation :
  - $\mathscr{E}_a$ pour l'état complet de l'agent A.
  - $\mathscr{E}_{ai}$ pour l'ensemble des attributs qui le caractérise par rapport à l'action Ac_i.
  - $\mathscr{E}_{aj}$ pour l'ensemble des attributs qui le caractérise par rapport aux environne-

Tableau

TABLE 6.6 – F : Les agents de la SOA

Expert	Agent A			Agent B			Agent C			…
	M	V	C	M	V	C	M	V	C	
Expert 1	$\mathscr{E}_{a1}$	D_1 D_3	Ac_1	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{a2}$	D_1	Ac_2	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{a3}$	D_1	Ac_3	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{a7}$	D_2	Ac_7	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{a8}$	D_2	Ac_8	✗	✗	✗	✗	✗	✗	
	…	…	…							
Expert 2	$\mathscr{E}_{a7}$	D_2	Ac_7	$\mathscr{E}_{b7}$	D_2	Ac_7	✗	✗	✗	
	$\mathscr{E}_{a8}$	D_2	Ac_8	$\mathscr{E}_{b8}$	D_2	Ac_8	✗	✗	✗	
Expert 3	$\mathscr{E}_{a1}$	D_1 D_3	Ac_1	✗	✗	✗	✗	✗	✗	
…	…	…	…	…	…	…	…	…	…	…

Comportement 1 ($\mathscr{C}_1$)= Ac_1 et Ac_2
Comportement 2 ($\mathscr{C}_2$)= Ac_3
Comportement 3 ($\mathscr{C}_3$)= Ac_7 et Ac_8…

ments Env_j (*c'est-à-dire* les attributs induits).

$\mathscr{E}_a$ est donc l'union de tous les $\mathscr{E}_{ai}$ et $\mathscr{E}_{aj}$ relatifs à cet agent. Ces états sont en général en rapport avec les dynamiques et environnements considérés, mais pas toujours. Par exemple, la position spatiale d'un agent est en accord avec la dynamique spatiale, mais son identifiant n'est en corrélation avec aucune dynamique particulière.
A cette étape, les modélisateurs-informaticiens se basent sur l'avis des modélisateurs et des thématiciens pour décrire les variables du modèle : quels sont leurs types ? leurs valeurs minimales et maximales ? les intervalles de changement ? Quelles sont les valeurs par défaut ? à l'initialisation ? S'agit-il de paramètres de scénario (donc configurables dans une future interface) ou non ?…

- **Les vues V** : Les interacteurs $\mathscr{I}_k$ de l'agent (qui permettent l'influence et la perception) avec la dynamique associée. Dans le tableau, D_k dans la colonne V d'un agent signifie qu'il participe à la dynamique D_k et qu'il effectue donc des influences/perceptions grâce à des interacteurs dans cette dynamique.
  Par facilité d'écriture, nous assignerons à la colonne V les dynamiques directement. Cela sous-entend les interacteurs de l'agent liés à cette dynamique pour effectuer les actions.

- **Les contrôleurs C** : L'ensemble des actions Ac_l du comportement $\mathscr{C}$ considéré, qui gère les interactions de l'agent dans les dynamiques, avec les environnements. Ils sont formalisés sous forme de pseudo-code.
  Pour rappel, l'**algorithme** est la sémantique de l'expression de la résolution d'un problème. Elle est décrite de diverses manières : formules mathématiques pures, graphiques complexes, pseudo-codes... Le **pseudo-code** est une syntaxe possible de l'algorithme, pour le décrire de manière à faciliter le passage dans un langage programmation.

La question qui se pose pour un agent passif est la suivante : comme il n'a aucun comportement, quel est son contrôleur ? Par exemple, un agent Bois passif est déplacé par un agent termite. Ce problème est résolu par la création d'un Contrôleur "direct", présenté dans la Section 4.4.2.

Exemple

Le second bilan nous permet d'obtenir le tableau ci-après, avec, pour rappel, le comportement "Chasse" composé des actions Ac_1 (Localisation de la proie), Ac_2 (Poursuite), Ac_3 (Attaque) et Ac_4 (Partage). Utilisons les abréviations D_s pour la dynamique Espace, D_c pour celle de Communication et D_e pour celle d'Élevage.

Expert	Agent Loup			Agent Caribou			Agent Homme			…
	M	V	C	M	V	C	M	V	C	
Expert Loup	$\mathscr{E}_{l1}$	D_s D_c	Ac_1	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{l2}$	D_s	Ac_2	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{l3}$	D_s	Ac_3	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{l4}$	D_s D_c	Ac_4	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{l6}$	D_s D_c	Ac_6	✗	✗	✗	✗	✗	✗	
	$\mathscr{E}_{l9}$	D_s D_c	Ac_9	✗	✗	✗	✗	✗	✗	
	…	…	…	✗	✗	✗	✗	✗	✗	
Expert Caribou	✗	✗	✗	$\mathscr{E}_{c9}$	D_s D_c	Ac_9	✗	✗	✗	
				…	…	…				
Expert Prédateur	$\mathscr{E}_{l3}$	D_s	Ac_3	✗	✗	✗	✗	✗	✗	
…	…	…	…	…	…	…	…	…	…	…

Comportement Chasse = Ac_1, Ac_2, Ac_3 et Ac_4
Comportement Migration = Ac_6
Comportement Repos = Ac_9…

Résumé de l'étape F

F : Bilan des agents (par les modélisateurs-informaticiens) :

- Formaliser chaque comportement pour chacun des agents sous la forme aMVC :
  - M = État de l'agent
  - V = Interacteurs dans la dynamique concernée
  - C = Comportements (groupe d'actions) exprimés sous forme de pseudo-code

Résultat : Décomposition sous la forme aMVC des agents.

Grâce à ce bilan des agents, reprenons la formalisation énoncée dans le Chapitre 4 de plusieurs manières. Chaque agent est un ensemble non vide de couples $\langle M, C \rangle$ et $\langle C, V \rangle$ où :

- Pour chaque domaine d'expertise, nous créons un Modèle M_i, ensemble $\mathscr{E}_i$ des caractéristiques internes en relation avec ce domaine.
- Pour chaque comportement, nous mettons en place un Contrôleur C_j, ensemble $\mathscr{C}_j$ des actions qui compose ce comportement. Pour un même modèle, nous aurons donc plusieurs Contrôleurs.
- Pour chaque dynamique, nous avons une Vue V_k, ensemble $\mathscr{I}_k$ des interacteurs de l'agent dans cette dynamique.

Une autre possibilité serait d'avoir une plus grande finesse dans la gestion des comportements en créant un Contrôleur pour chaque action (au lieu d'ensemble d'actions) et un Modèle (état) pour chacun d'entre eux. Cela engendrerait effectivement une très grande complexité dans la gestion des relations et pour la consistance des informations de l'état de l'agent, mais n'oublions pas que ce niveau de complexité est aujourd'hui présent dans tous nos logiciels construits sur le modèle **M**odèle **V**ue **C**ontrôleur (MVC).

6.4 Avantages obtenus

Ces descriptions aident à représenter le système complexe considéré. Elle permet de construire progressivement le modèle final en intégrant petit à petit les informations en suivant les différentes étapes synthétisées sur la Figure 6.3, où nous illustrons à quel moment les concepts DOM et aMVC interviennent.

De par sa construction, notre méthodologie facilite la collaboration entre des thématiciens pour construire un modèle :

- En effet, chacun peut utiliser son vocabulaire pour exprimer sa connaissance sur les entités du système et ensuite la comparer entre collègues.
- Grâce à l'affinement progressif de cette méthodologie, il est simple de détecter les erreurs éventuelles, si chaque expert consulte et réajuste les informations issues des différents outils de cette méthodologie.

Cette méthodologie facilite aussi la collaboration entre les groupes d'experts, de modélisateurs, de modélisateurs-informaticiens... :

- En effet, chaque acteur possède une tâche bien déterminée en fonction de ses capacités, tout en ayant une vision globale du système.
- De plus, après l'implémentation de la simulation suivant l'architecture aMVC, il est aisé de rectifier et de réutiliser les comportements et actions des agents, au besoin.

Figure

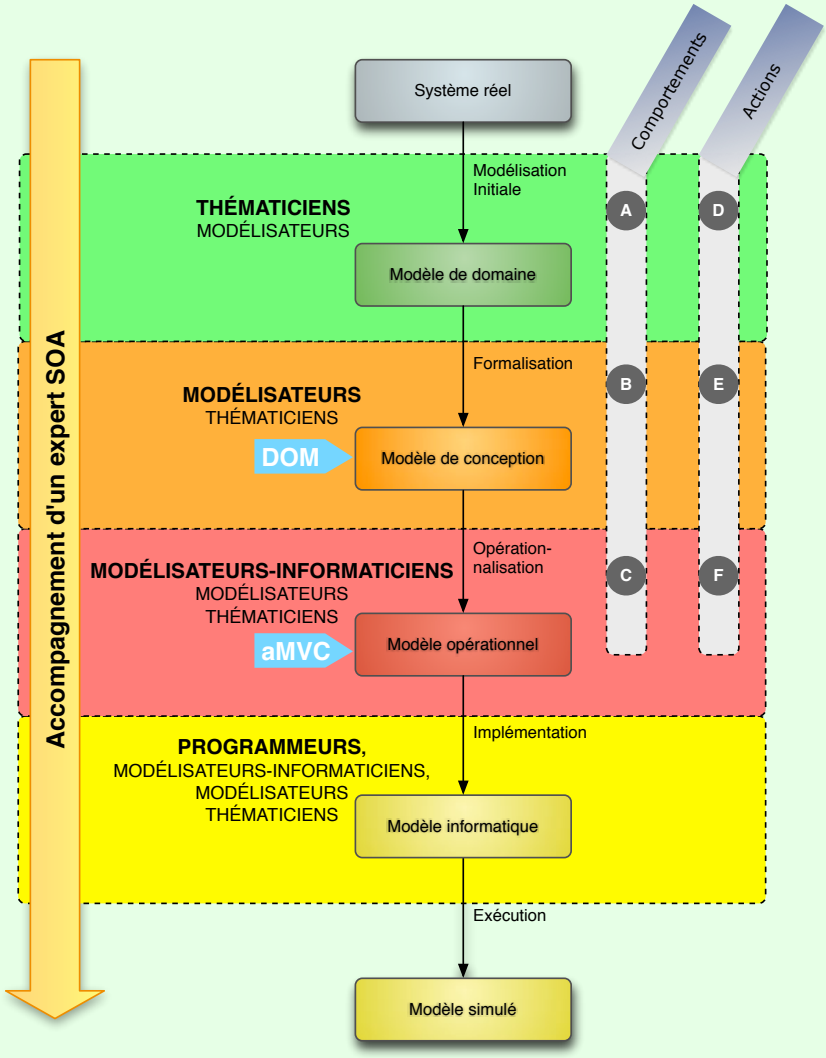


FIGURE 6.3 – Les étapes de la méthodologie de Modélisation MultiComportementale

La synergie qu'elle crée permet donc une collaboration plus efficace et facilite la création de comportements complexes.

6.5 Validation par approche comparative

L'approche "comparative" consiste dans la validation de la méthode par comparaison directe avec une méthode analogue déjà validée. Dans cette partie, nous situons notre travail

par rapport à quelques méthodologies connues en déroulant un exemple. Étant donné la simplicité de l'exemple, nous savons que cette comparaison n'est pas exhaustive, mais nous discuterons suivant l'exemple et les principes de ces méthodologies.

Le résumé de notre méthodologie peut être lu au travers de la Figure 6.3. Comme nous l'avons précisé, il ne s'agit pas d'une suite d'étapes. Certaines peuvent avoir lieu en parallèle, il est aussi possible de revenir sur une étape pour modifier certaines informations (ce qui forcément impliquera de revoir les étapes qui en découlent). Le déroulement du processus de la méthodologie se résume comme suit :

A : Identification des comportements (par les experts) :

- Choisir un agent ou s'il s'agit d'un nouvel agent :
  - Description de l'agent.
  - Ajout à une liste des types d'agents.
  - Définition du statut actif ou passif de chaque agent.
- Décrire un comportement :
  - Description en langage naturel du comportement.
  - Ajout du comportement à la liste des comportements de l'agent.
  - Définition de la priorité du comportement.
- Définir l'algorithme de changement de priorité, si besoin est.

Résultat : Liste des agents, Listes des comportements dans le modèle de domaine et Algorithme de priorité.

B : Déclinaison et formalisation des comportements (par les modélisateurs) :

- Déclinaison des comportements selon acteur/objet si nécessaire.
- Pour chaque déclinaison, création et/ou association à une ou plusieurs dynamiques.
- Formalisation de ces déclinaisons selon la forme : "*En tant qu'**<agent> acteur/objet** du **<comportement>**, ma fonction **active/passive** est de **<besoin>**,afin de **<but>**, au travers des dynamiques **<ensemble de dynamiques>**.*"

Résultat : Liste des comportements formalisés (et de leurs déclinaisons) du modèle conceptuel, Liste des dynamiques . Pour chaque dynamique nous avons aussi la liste des agents actifs et celle des agents passifs.

C : Bilan des dynamiques (par les modélisateurs-informaticiens) :

- Identifier les environnements à associer aux dynamiques :
  - Décrire le type d'environnement
  - Énoncer les lois de l'environnement.
  - Créer des "attributs induits" par l'environnement pour un agent.
- Identifier les agents à associer aux dynamiques : pour chaque agent, on établit la liste des déclinaisons des comportements possibles selon le modèle précédent.

Résultat : Liste des agents et des environnements associés aux dynamiques.

D : Identification des actions (par les thématiciens) :

- Choisir un comportement et lister les actions.
- Décrire chaque action :
  - Description en langage naturel.
  - Ajout du comportement à la liste.

Résultat : Liste des actions du modèle de domaine pour chaque comportement de chacun des agents.

E : Formalisation des actions (par les modélisateurs) :

- Formaliser chaque action sous forme d'algorithme et/ou de logique de premier ordre, suivant la nécessité.
- Associer chaque action à la dynamique correspondante

Résultat : Liste des actions formalisées de chaque comportement pour chacun des agents.

F : Bilan des agents (par les modélisateurs-informaticiens) :

- Formaliser chaque comportement pour chacun des agents sous la forme aMVC :
  - M = État de l'agent
  - V = Interacteurs dans la dynamique concernée
  - C = Comportements exprimés sous forme de pseudo-code

Résultat : Décomposition sous la forme aMVC des agents.

6.5.1 Exemple d'évaluation : les termites selon notre méthodologie de Modélisation MultiComportementale

Pour illustrer nos propos, voici un exemple bien connu : celui des termites de Mitchell Resnick [Resnick, 1997]. Il est inspiré par le comportement des termites empilant des copeaux de bois. Les termites suivent un ensemble de règles simples, aboutissant à la construction d'une pile unique de copeaux.

Étapes A, B et C

Ces étapes de la première phase se concentrent sur les comportements. Elle est résumé ainsi dans le tableau 6.7.

A : Dans cet exemple, nous distinguons deux familles d'agents : les agents de type "**Termite**" et ceux de type "copeau de **Bois**". L'agent **Bois** est un agent passif alors que l'agent **Termite** peut être décrit selon les deux comportements suivants :

- **Comportement d'Errance** : Chaque termite commence à errer au hasard.
- **Comportement de Manipulation** d'un copeau de bois : S'il rencontre un copeau, il le prend, il continue à errer jusqu'à rencontrer un autre copeau, il le dépose dans un espace vide à côté de l'autre.

Tableau

TABLE 6.7 – Étapes A, B et C

Agents \ Experts	Expert Termite
Agent Termite	Errance Manipulation
Agent Bois	*(Aucun comportement)*

(a) A : Les agents et les experts (Comportements)

Agents \ Dyn.	Dynamique Spatiale
Agent Termite	Errance Manipulation
Agent Bois	*Présence uniquement*

(b) B : Les agents et les dynamiques associées (Comportements)

	Dynamique Spatiale
Environnement	Env. Spatial 2D
Agents	Termite, Bois

(c) C : Les dynamiques et environnements du système

Les comportements de l'agent Termite, avec, par ordre de priorité par défaut (1 étant la priorité la plus haute), sont :

1. Comportement d'errance = 1
2. Comportement de Manipulation = 2

L'algorithme qui permet le changement de priorité est le suivant : Si la priorité d'errance d'un agent Termite est égale à 1 et qu'il rencontre un agent Bois, alors la priorité de l'errance passe à 2 et celle de la manipulation passe à 1. Si sa priorité de manipulation est égale à 1 et qu'il a déposé un agent Bois, alors la priorité de l'errance revient à 1 et celle de la manipulation passe à 2.

Cet algorithme se traduit par :

Algorithme 1 Changement de priorité des comportements de l'agent Termite

1: **Si** $priorite(Errance) = 1$ **Alors**
2: **Si** Rencontrer Bois **Alors** ▷ Si l'agent Termite rencontre un agent Bois
3: $priorite(Errance) = 2$ et $priorite(Manipulation) = 1$
4: **Fin Si**
5: **Fin Si**
6: **Si** $priorite(Manipulation) = 1$ **Alors**
7: **Si** Deposer Bois **Alors** ▷ Si l'agent Termite a fini de déplacer un agent Bois
8: $priorite(Errance) = 1$ et $priorite(Manipulation) = 2$
9: **Fin Si**
10: **Fin Si**

B : Le comportement d'errance n'a pas besoin d'être décliné, car l'agent Termite en est l'acteur et l'objet à la fois : il influe sur lui-même pour se déplacer. Par contre, le comportement de manipulation doit être décliné : *Manipulation.acteur* et *Manipulation.objet*. En effet, le comportement de manipulation de l'agent Termite fait que l'agent Bois est manipulé par le termite. L'acteur de la manipulation est donc l'agent Termite et l'objet de ce comportement, l'agent Bois.

La formalisation de ces comportements est la suivante :

- **Comportement d'Errance** : En tant que **Termite acteur** de **l'Errance**, ma fonction **active** est d' **errer aléatoirement**, afin de **trouver un agent Bois**, au travers de **la dynamique spatiale**.
- **Comportement de Manipulation.acteur** : En tant que **Termite acteur** de **la Manipulation**, ma fonction **active** est de **trouver un agent Bois, le déplacer en errant**, afin de **le déposer près d'un autre agent Bois**, au travers de **la dynamique spatiale**.
- **Comportement de Manipulation.objet** : En tant que **Bois objet** de **la Manipulation**, ma fonction **passive** est **d'être déplacé**, afin **d'être déposé près d'un autre agent Bois**, au travers de **la dynamique spatiale**.

Étant donné que le comportement de manipulation impose une présence dans la dynamique spatiale, nous pouvons en déduire que l'agent Termite a une présence dans cette dynamique. Le fait que l'agent Bois soit l'objet de cette manipulation, même s'il n'a pas de comportement propre, implique qu'il participe aussi à la dynamique spatiale. Le comportement d'errance de l'agent Termite se déroule aussi dans la dynamique spatiale.

C : L'environnement proposé pour cette dynamique spatiale est à deux dimensions (avec un repère centré par exemple), non accessible, déterministe, non épisodique, statique et discret. L'agent Termite n'a pas de connaissance de l'environnement dans sa totalité, mais il ne perçoit que ce qui se trouve à sa position. Tous les agents participant à cette dynamique ont obligatoirement une position x et y. De plus, ils ont une vitesse (nulle si l'agent ne se déplace

pas) ainsi qu'une direction (nulle si inutile).

Étapes D, E et F

La seconde phase de notre méthodologie sur cet exemple nous amène à nous intéresser aux actions et se résume ainsi dans le tableau 6.8.

D : L'expert des termites considère les deux comportements et explicite les actions :

- Comportement d'Errance : Il est composé uniquement de l'action d'**Errer** aléatoirement.
- Comportement de Manipulation : Lorsque l'agent Termite rencontre un copeau de bois, il va **Prendre le copeau**, puis **Errer avec le copeau** jusqu'à en rencontrer un autre. Ensuite, il devra **Déposer le copeau** dans un espace vide à côté de celui-ci. Enfin, il va **S'éloigner du tas** de copeaux et reprendre le comportement d'Errance.

E : Les modélisateurs formalisent ensuite les actions du modèle de domaine :

- Comportement d'errance : Comportement par défaut
  - Ac_{Errer} = **Errer** aléatoirement, *c'est-à-dire* se déplacer d'une certaine distance, à une certaine vitesse puis changer sa direction d'un certain angle.
- Comportement de Manipulation : Comportement lorsque l'instance de l'agent *Termite* considérée rencontre une instance *Bois1*
  - $Ac_{PrendreCp}$ = **Prendre le copeau** de bois : *Termite* se positionne sur *Bois1* et influe sur sa position (de sorte que $position_{Termite} = position_{Bois1}$). Il passe au comportement suivant.
  - $Ac_{ErrerCp}$ = **Errer avec le copeau** de bois jusqu'à en rencontrer un autre : *Termite* se déplace de la même manière qu'**Errer**, mais de telle sorte que $position_{Termite} = position_{Bois1}$ pour chaque pas de temps n. Il continue jusqu'à croiser une autre instance d'agent Bois *Bois2*. Il passe au comportement suivant.
  - $Ac_{DeposerCp}$ = **Déposer le copeau** de bois à côté d'un autre copeau : Lorsque *Termite* portant une instance d'agent *Bois1* rencontre *Bois2*, il recule puis dépose *Bois1* à côté de *Bois2* (suivant la métrique de l'environnement). Si *Termite* trouve sur son chemin *Bois2* au moment t, alors on aura :
    $position_{Termite}(t) = position_{Bois2}$ et
    $position_{Termite}(t-1) = position_{Termite}(t+1) = position_{Bois2}$.
  - $Ac_{Eloigner}$ = **S'éloigner du tas** de copeaux : *Termite* recule encore par rapport à la position précédente, change de direction d'un certain angle et recommence à **Errer**.

Dans notre cas, toutes les actions participent à une seule et même dynamique, la dynamique spatiale, exerçant ainsi des influences et perceptions dans l'environnement associé.

Tableau

TABLE 6.8 – Étapes D, E et F

Agents \ Experts	Expert Termite
Comportement Errance (Agent Termite)	Errer
Comportement Manipulation (Agent Termite)	Prendre le copeau Errer avec le copeau Déposer le copeau S'éloigner du tas
(Agent Bois)	*(Aucune action)*

(a) D : Les agents et les experts (Actions)

Agents \ Dyn.	Dynamique Spatiale
Comportement Errance (Agent Termite)	Errer
Comportement Manipulation (Agent Termite)	Prendre le copeau Errer avec le copeau Déposer le copeau S'éloigner du tas
(Agent Bois)	*(Présence uniquement)*

(b) E : Les agents et les dynamiques (Actions)

Expert	Agent Termite			Agent Bois		
	M	V	C	M	V	C
Expert Termite	$\mathscr{E}_{T-Errer}$	$D_{Spatiale}$	Ac_{Errer}	$\mathscr{E}_{Bois}$	$D_{Spatiale}$	$C_{directe}$
	$\mathscr{E}_{T-PrendreCp}$		$Ac_{PrendreCp}$			
	$\mathscr{E}_{T-ErrerCp}$		$Ac_{ErrerCp}$			
	$\mathscr{E}_{T-DeposerCp}$		$Ac_{DeposerCp}$			
	$\mathscr{E}_{T-Eloigner}$		$Ac_{Eloigner}$			

(c) F : Les agents du système

F : La dernière étape de notre méthodologie conduit les acteurs du projet à expliciter les ensembles des variables qui caractérisent l'agent par rapport au comportement. Le type agent "Bois", ayant un corps dans l'environnement spatial, possède donc les variables suivantes pour son état :

$$\begin{aligned} E_{Bois} = \{ & identifiant && = \text{"identifiant de cette instance de Bois"}, \\ & position && = \text{"position (x,y) dans l'espace 2D"}, \\ & vitesse && = \text{"vecteur de déplacement (i,j), nul"}, \\ & angle && = \text{"direction du Bois, nulle "}\} \end{aligned}$$

Son contrôleur sera de type "direct", *c'est-à-dire* qu'il effectuera les modifications de l'état de l'agent selon les informations des récepteurs.

Par contre, concernant le termite, nous dénombrons davantage de variables, chaque comportement ayant accès à une subdivision de cet ensemble. Nous notons :

- $\mathscr{E}_{Termite} \rightarrow$ l'état complet d'un agent de type "Termite".
- $\mathscr{E}_{T-EnvSpatial} \rightarrow$ le sous-ensemble en relation avec les "attributs induits" de l'environnement.
- $\mathscr{E}_{T-n} \rightarrow$ le sous-ensemble en relation avec le comportement n.

Nous avons donc :

$$\begin{aligned} E_{T-EnvSpatial} = \{ & identifiant && = \text{"identifiant de cette instance de Termite"}, \\ & position && = \text{"position (x,y) dans l'espace 2D"}, \\ & vitesse && = \text{"vecteur de déplacement (i,j)"}, \\ & angle && = \text{"direction du termite mesurée en radians"}\} \\ E_{T-Errer} = \{ & && \} \\ E_{T-PrendreCp} = \{ & id_{Bois1} && = \text{"identifiant de l'instance de Bois1 rencontrée/portée"}\} \\ E_{T-ErrerCp} = \{ & id_{Bois1} && = \text{"identifiant de l'instance de Bois1 rencontrée/portée"}\} \\ E_{T-DeposerCp} = \{ & id_{Bois1} && = \text{"identifiant de l'instance de Bois1 rencontrée/portée"}, \\ & id_{Bois2} && = \text{"identifiant de l'instance de Bois2 rencontrée"}\} \\ E_{T-Eloigner} = \{ & && \} \end{aligned}$$

Dans l'exemple présent, on peut considérer que l'état d'un agent de type Termite est le

suivant (id_{Bois1} et id_{Bois2} étant nuls dans certains cas) :

$$\begin{aligned} E_{Termite} = \{ & identifiant && = \text{"identifiant de cette instance de Termite"}, \\ & position && = \text{"position (x,y) dans l'espace 2D"}, \\ & vitesse && = \text{"vecteur de déplacement (i,j)"}, \\ & angle && = \text{"direction du termite mesurée en radians"}, \\ & id_{Bois1} && = \text{"identifiant de l'instance de Bois1 rencontrée/portée"}, \\ & id_{Bois2} && = \text{"identifiant de l'instance de Bois2 rencontrée"}\} \end{aligned}$$

L'agent Termite sera donc formalisé, en posant :

- $M = E_{Termite}$
- V_{espace} est l'ensemble des capteurs et effecteurs qui permettent au termite d'agir dans l'espace.
- $C_{Errance}$ contient les actions du comportement d'**Errance** (Ac_{Errer}).
- $C_{Manipulation}$ contient les actions du comportement de **Manipulation** ($Ac_{PrendreCp}$, $Ac_{ErrerCp}$, $Ac_{DeposerCp}$ et $Ac_{Eloigner}$).

$$Agent^{Termite} = \{\langle M, C_{Errance}\rangle; \langle M, C_{Manipulation}\rangle; \langle C_{Errance}, V_{Espace}\rangle; \langle C_{Manipulation}, V_{Espace}\rangle\}$$

À partir de ce point, nous obtenons les éléments essentiels pour permettre l'implémentation de la simulation sur plateforme ou dans un langage donné.

6.5.2 Positionnement par rapport aux approches de réutilisation des actions

Dans la Section 2.4.2, nous avons présenté IODA (**I**nteraction **O**riented **D**esign of **A**gent simulations), méthodologie complétée par une plateforme de simulation appelée JEDI, un environnement de développement intégré nommé JEDI-BUILDER et un explorateur de l'espace des simulations LEIA. Dans un premier temps nous allons nous positionner par rapport à IODA puis par rapport à une autre technique de réutilisation d'actions.

Pour positionner notre méthodologie par rapport à IODA, nous ignorons les étapes intermédiaires. Les principaux résultats sont : la "matrice d'interaction raffinée", la "matrice de mise à jour ordonnée", les "attributs des familles d'entités" et "de l'environnement" et enfin les différentes "primitives d'action et de perception" et celle "de l'environnement".

Les termites selon IODA

Le résultat de la méthodologie IODA appliqué à l'exemple des termites est directement issu du site du projet IODA[5] [Mathieu *et al.*, 2013]. Elle peut être résumée à travers la matrice d'interaction raffinée (voir tableau 6.9), la spécification de chacune des interactions présentes dans le tableau, des attributs des agents et de l'environnement.

[5] http://www.lifl.fr/SMAC/projects/ioda/ioda/models/termites_nest.php (Page consultée le 1er février 2013).

Tableau

TABLE 6.9 – Matrice d'interaction raffinée pour la simulation de termites dans IODA

Source \ Target	∅	Wood chip
Termite	(Wander, p=0) (Put down, p=1)	(Pick up, d=0, p=3) (Find empty place, d=0, p=2)
Wood chip		

Les interactions sont définies comme dans l'exemple suivant (pour plus de détails , se référer à [Mathieu *et al.*, 2013]) :

- **Wander's Interaction**
  - **Description** : Source agent wanders, by turning from a random number, and then moving forward for a particular distance.-
  - **Trigger** : true
  - **Preconditions** : true
  - **Actions** :
    double angle = Source.getDeviationAngle() x (2 x random() - 1) ;
    environment.turnAgent(Source, angle) ;
    environment.moveAgentForward(Source, Source.getSpeed()) ;

- **Wander's Primitives**
  - **getDeviationAngle()** : Defines in which interval the angle used to turn the agent is drawn. This interval is [-getDeviationAngle() ; getDeviationAngle()].
  - **getSpeed()** : Defines the distance of the move.

La description de l'agent est aussi donnée au travers de primitives. Par exemple, pour l'agent termite, nous avons :

```
getDeviationAngle():          return (pi / 4);
getSpeed():                   return (1);
carriesSomething():           return (this.inventory != null);
setCarriedElement(Agent t):   this.inventory = t;
getCarriedElement():          Agent tmp = this.inventory;
                              this.inventory = null;
                              return tmp;
```

Autres approches

GAMA (**G**eneric **A**daptive **M**obile **A**gent architecture) est une plateforme de SOA fournissant un environnement complet pour le développement de modélisations et simulations d'agents situés. Elle est basée sur Éclipse et a été développée par l'équipe de recherche MSI (voir Section 2.3.2).

Parmi les points forts de cette plateforme, nous pouvons noter les différents niveaux d'abstraction qu'elle propose pour les agents. De plus, elle permet à des scientifiques non-informaticiens de mettre en place des modèles de conception et d'interagir avec les agents lors des simulations. Elle utilise un langage de modélisation GAML et propose une bibliothèque importante de primitives pour les actions des agents.

Par l'intermédiaire de cette bibliothèque, GAMA permet la réutilisation d'action. Une action est une capacité spéciale d'un agent d'une certaine espèce. Elle est représentée par un bloc d'instructions, avec entrées et sorties, qui peut être réutilisé à volonté. Par exemple :

```
action action_addition {
  arg arg1 type: int;
  arg arg2 type: int;
  return arg1 + arg2;
}
```

Synthèse de la comparaison

Comme dans IODA, tout comportement d'un agent est concrétisé par une interaction, il est intéressant de comparer ses résultats à nos descriptions. En effet, dans le bilan des agents (voir tableau 6.6) nous retrouvons les interactions des matrices, les attributs des familles d'entités et les primitives d'action et de perception. Notre bilan des dynamiques fournit les attributs de l'environnement et ses primitives. Bien que notre méthodologie ne soit pas fondée sur l'interaction, mais le comportement, elle produit les mêmes résultats, ordonnés différemment.

Comme nous l'avons vu dans le Chapitre 2, IODA permet la réutilisation des interactions. Mais l'un des avantages de notre méthodologie, par rapport à IODA, réside dans la notion de comportement, avec sa priorité et sa définition par une ou plusieurs actions : elle est capable de gérer des ensembles complexes de comportements. Par exemple, il n'est pas possible de définir dans IODA la chasse comme composée de plusieurs actions : rechercher la proie, partager avec les autres membres de la meute sa position, poursuivre la proie, attaquer de manière groupée et partager la nourriture. Dans IODA, ces comportements seront des interactions au même niveau que dormir, boire, *etc.*

Il semble qu'il soit possible de définir une suite d'interactions par l'intermédiaire d'un "Modèle de sélection d'interactions" et d'utiliser la participation simultanée d'un agent à plusieurs interactions. IODA offre bien la possibilité de modéliser des "interactions complexes" [Kubera, 2010], mais sa définition de ce concept est une interaction simple impliquant plus de deux agents (en tant que cibles ou sources), ce qui ne permet pas de résoudre notre problème.

Les approches bibliothèques permettent aussi, jusqu'à un certain niveau, la réutilisation des actions pour l'agent. Il est intéressant de constater que dans la représentation de l'action

de GAMA, les variables sont bien des arguments, comme nous le proposons dans notre modèle aMVC. Notre approche possède l'avantage d'expliciter le lien entre les comportements et les attributs de l'état de l'agent, ou encore entre les comportements et les interacteurs.

6.5.3 Positionnement par rapport aux approches de réutilisation des comportements

Les termites selon AGR

La méthodologie **A**gent, le **G**roupe et le **R**ôle (AGR, voir Section 2.4.2) a été appliquée à l'exemple des termites dans un rapport sur TurtleKit [Michel, 2002], plateforme qui permet de construire des simulations pour Madkit à partir de Logo. Dans NetLogo [Wilensky, 1999], on dénombre deux types d'entités :

- le "turtle" : il s'agit d'un agent (mobile).
- le "patch" : il représente une partie de l'environnement, par un carré de taille fixe et immobile. L'environnement spatial des agents est discrétisé en un ensemble de patches pouvant exécuter des opérations.

Ici, les termites sont les "turtles" et les copeaux de bois ne sont pas des agents, mais des "patches". Nous avons donc :

- des agents "Entité-Termite". Chaque agent possède ici un seul rôle et fait partie d'un seul groupe.
- un "Rôle-Termite" composé de plusieurs comportements : searchForChip, findNewPile, findEmptyPatch et getAway.
- un groupe "Simulation de termites" qui regroupe tous les agents "Entité-Termite".

Les points qui nous intéressent sont les comportements de la Figure 6.4 implémentés comme suit :

```
public String searchForChip() {
    wiggle();
    if (getPatchColor() == Color.yellow) {
        setPatchColor(Color.black);
        randomHeading();
        fd(20);
        return("findNewPile");
    } else return ("searchForChip");
}
public String findNewPile() {
    if(getPatchColor()==Color.yellow) return("findEmptyPatch");
    else {
        wiggle();
```

Figure

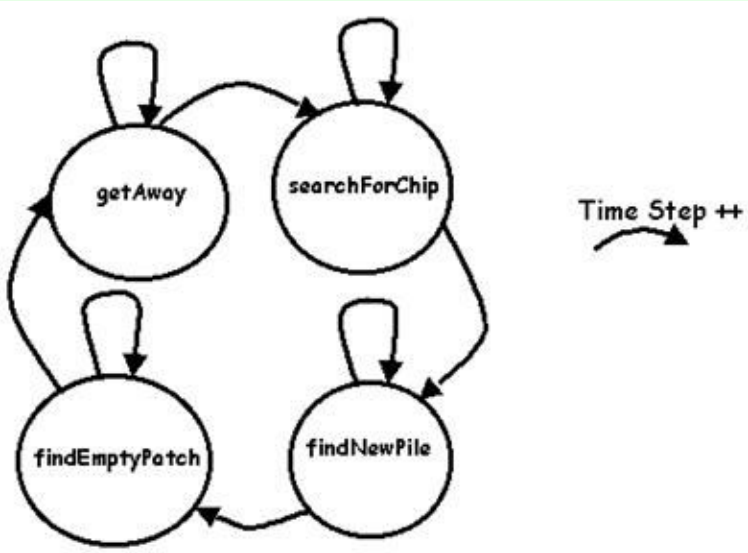


FIGURE 6.4 – Comportements du termite dans le modèle AGR, selon [Michel, 2002]

```
        return("findNewPile");
    }
}
public String findEmptyPatch() {
    wiggle();
    if(getPatchColor()== Color.black) {
        setPatchColor(Color.yellow);
        return("getAway");
    } else return("findEmptyPatch");
}
public String getAway() {
    if(getPatchColor()== Color.black) return("searchForChip");
    else {
        randomHeading();
        fd(20);
        return("getAway");
    }
}
public void wiggle() {
    fd(1);
```

```
    turnRight(Math.random()*45);
    turnLeft(Math.random()*45);
}
```

Synthèse de la comparaison

L'exemple des termites ne montre pas la puissance de la méthodologie AGR. Cependant, cette notion de rôle est intéressante. Notre méthodologie n'est pas orientée sur ce concept, cependant nous pouvons remarquer la similarité avec notre méthodologie par rapport à cette notion où un agent a un comportement sur plusieurs dynamiques différentes. Ainsi, si le découpage suivant les dynamiques peut représenter les groupes, alors nous pouvons dire que les comportements peuvent représenter les rôles. L'algorithme de priorité peut être optimisé dans ce but. Par exemple, si nous définissons un ensemble de comportements A puis un autre ensemble B, nous pouvons assigner ces ensembles de manière à répondre au besoin de la notion de rôle : selon certaines conditions l'agent fera les actions de A et selon d'autres les actions de B.

MadKit proposait [Gutknecht *et al.*, 2001] aussi une bibliothèque en ligne qui permet aux utilisateurs de télécharger des agents ou des formalismes SEdit. Cependant cette réutilisation impose l'agent en entier et non les comportements.

6.5.4 Positionnement par rapport aux approches de co-construction

Notre approche méthodologique permet, grâce à ses différents supports et son approche, une collaboration étroite entre les différents intervenants, pour co-construire le modèle, bien que ce ne soit pas nettement visible sur l'exemple des termites. Nous avons vu dans la Section 2.3.2 que peu de méthodologies s'approchent de celle que nous proposons.

Cependant, il est important de souligner que les approches proposées sont complémentaires. Ainsi, les outils de PAMS ou la modularité au niveau des modèles de JDEV sont des pistes pour nos travaux futurs.

6.6 Conclusion

L'approche méthodologique de Modélisation MultiComportementale est une approche pour la co-construction d'un modèle réutilisable lorsque nous exploitons la proposition d'architecture précédente aMVC. Elle consiste en un ensemble de descriptions élémentaires du système complexe considéré pour aboutir à la mise en place de DOM couplée aMVC. Cette approche méthodologique se déroule en six phases qui permettent de découper précisément, avec des étapes identifiées, la modélisation d'un système complexe. Chaque étape est définie par un ensemble de descriptions élémentaires, pouvant être représentées sous forme de tableaux.

D'une part, cette approche méthodologique a été élaborée avec l'intention de simplifier la co-construction du modèle, grâce à la synergie créée, permettant de conduire un projet pluridisciplinaire de SOA avec un grand nombre d'acteurs. Les concepteurs pourront travailler

de manière autonome sur leurs dynamiques et la plateforme en fera l'intégration en assurant la cohésion et la robustesse du système.

D'autre part, elle apporte une aide à la réutilisation des modèles : basée sur l'approche DOM et notre nouveau pattern aMVC, cette approche offre une manière simple de créer les briques élémentaires du système de manière indépendante, de les associer et de les combiner pour former des agents, selon des dynamiques. Elle permet ainsi de comparer la logique de différentes possibilités pour une même dynamique et ainsi d'offrir la possibilité d'étudier un grand nombre d'alternatives de modélisation d'un même système complexe, de les analyser ensuite à une échelle très fine.

Depuis de début de notre travail de recherche, nous avons utilisé le terme "*pluridisciplinaire*". Ce terme marque une approche plutôt parallèle, lorsque des experts de domaines distincts travaillent ensemble, chacun gardant la spécificité de ses concepts et méthodes. Le but est atteint ensuite par l'addition des contributions de chacun. C'est ce terme que nous avons choisi, pour marquer cette possibilité de travailler chacun selon son domaine. Cependant, au terme de ce chapitre, nous pouvons utiliser aussi "*interdisciplinarité*" pour caractériser notre méthodologie, qui permet au final un échange de connaissances et de méthodes entre les experts de domaines différents, par l'enrichissement progressif du modèle.

7 Cadre Expérimental : Prototype d'atelier de Modélisation MultiComportementale

Plan du chapitre

7.1 Introduction

Dans les précédents chapitres, nous avons proposé une architecture Agent aMVC, appliquée dans un cadre multidynamique DOM, avec l'appui d'une approche méthodologique **M**odélisation **M**ulti**C**omportementale (MMC) centrée sur la décomposition et la réutilisation des comportements.

Nous allons décrire ici un prototype d'atelier permettant d'accompagner la mise en œuvre de cette méthodologie par une équipe projet. Cet atelier permet d'illustrer l'automatisation du processus et les possibilités d'inférences sur ses éléments. Il s'agit d'un cadre expérimental de la méthodologie. L'objectif est d'accompagner les acteurs du projet pour la création des modèles en suivant les étapes A à F, pour aboutir à un modèle opérationnel. Une fois cette modélisation terminée, ce résultat pourra être exploité pour produire les modèles informatiques propres à chaque plateforme de simulation agent (cette dernière étape sort du cadre de notre approche méthodologique de modélisation). Cet atelier est donc un outil d'accompagnement de process qui permet aux groupes de travail de mener une modélisation productive en exploitant au mieux les connaissances de chacun des membres.

En sortie de ce prototype, le cahier des charges des programmeurs présente : la structure du code de l'agent avec la formalisation aMVC, l'ensemble des dynamiques et environnements ainsi que les informations associées. Cette connaissance est obtenue sous une forme descriptive indépendante de toute plateforme ou implémentation.

7.2 Atelier de modélisation

7.2.1 Choix technologiques

Pour implémenter notre prototype, nous avons choisi d'utiliser le framework jConverse[1]. Développé dans un précédent travail par Mahmoud Gangat et moi même, ce framework utilise les concepts combinés du **D**omain-**D**riven **D**esign (DDD) et du **M**odel-**D**riven **S**oftware **D**evelopment (MDSD).

Pour l'enregistrement et le chargement des objets, jConverse emploie la technologie de persistance Hibernate avec l'interface JPA. Hibernate est un ORM (**O**bject **R**elational **M**apping) qui fait le mapping entre le "monde objet" des objets manipulés et le "monde relationnel" des bases de données. JPA est une spécification qui offre une certaine indépendance par rapport aux technologies de persistance.

Pour l'affichage graphique, jConverse a recours à des annotations sur les objets du domaine afin de générer à la volée les éléments qui permettront d'interagir avec ces objets, au travers de la notion de CRUD (**C**reate, **R**ead, **U**pdate and **D**elete). Celle-ci permet de réaliser les opérations de listing, ajout, modification et suppression sur un objet de la base.

L'architecture du framework, multicouche (voir Figure 7.1), comporte :

- La Couche Interface Web qui dialogue avec l'utilisateur.

[1] Disponible sur https://code.google.com/p/jconverse/ (Page consultée le 1er février 2013).

- La Couche Objets Métiers concernant la logique spécifique de l'application.
- La Couche DAO (Data Access Object) qui fournit les données préenregistrées et utilise les actions de persistance CRUD.
- La Couche Interface JPA.
- La Couche Hibernate.
- La couche JDBC, API fournie avec Java permettant de se connecter à des bases de données.
- La base de données.

Le principe du framework consiste à utiliser des objets métiers complétés par des annotations afin de construire le socle de l'application. L'objectif est de faciliter le prototypage et permettre des livraisons rapides en suivant les principes Agile [Zannier *et al.*, 2004]. Une fois que tous les objets métiers ont été mis en place, l'utilisateur final utilise l'application à travers une interface Web. Ce framework facilite la génération des interfaces en offrant une bonne flexibilité et une certaine robustesse.

Figure

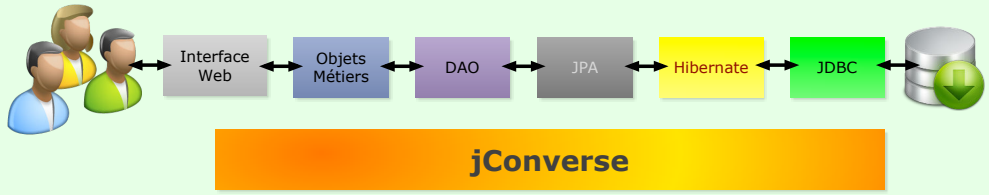


FIGURE 7.1 – Architecture de jConverse

Nous avons choisi d'utiliser ce framework car il utilise le mécanisme d'introspection (appelé aussi réflexivité) qui permet d'automatiser la découverte dynamique des informations relatives à une classe ou à un objet. Ce mécanisme va servir à guider les saisies des utilisateurs afin de renseigner pleinement l'ensemble des éléments qui vont successivement émerger au cours du travail de modélisation. Il va aussi, plus particulièrement, nous permettre de faciliter l'interfaçage dans les couples $\langle M, C \rangle$ et $\langle C, V \rangle$ en fournissant un support à l'identification des points de jonction entre les objets. D'autre part, l'utilisation des notions sous-jacentes à DDD et MDSD, qui permettent notamment la génération de codes *via* des templates ou des interprétations, sorties du contexte du prototype, pourrait compléter notre approche et fournir une assistance supplémentaire pour passer du modèle opérationnel au modèle informatique spécifique à une plateforme ou un langage donné.

Figure

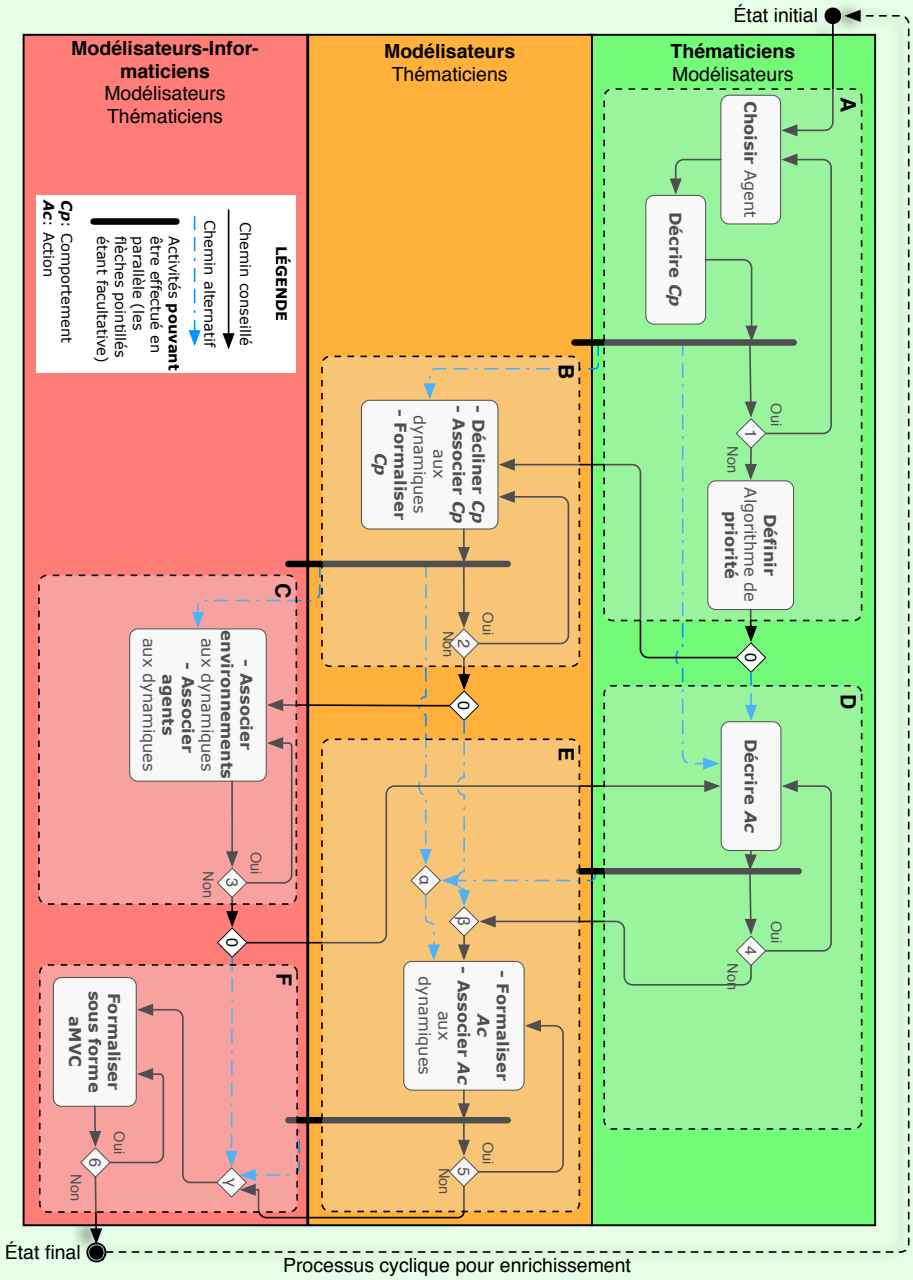


FIGURE 7.2 – Diagramme UML des activités, représentant les étapes de la méthodologie

7.2.2 Conception

Le prototype que nous avons conçu reprend les six étapes de l'approche méthodologique du Chapitre 6. Nous les avons schématisées dans le diagramme UML des activités de la Figure 7.2. Ce diagramme peut être vu comme le plan de la conception du prototype. Dans la suite de ce chapitre, le terme "utilisateur" désigne la personne utilisant le prototype et correspond selon le cas aux thématiciens, modélisateurs ou modélisateurs-informaticiens.

La légende des nœuds de décision des Figures 7.2 et 7.3 est la suivante :

0 : Ces nœuds marquent un changement de couche de modélisation possible. Ils peuvent être aussi des raccourcis pour changer d'étapes ou passer la main à d'autres utilisateurs.

- Après l'étape A, passer à B ou D.
- Après l'étape B, s'orienter vers C ou E (si l'étape D est finie).
- Après l'étape C, se diriger vers D ou F (si E est terminée).

1 : Reste-t-il des comportements à décrire ?

2 : Y a-t-il encore des comportements à formaliser ?

3 : Reste-t-il des dynamiques à traiter ? S'il n'y en pas, on commence l'affinement du modèle.

4 : Est-ce qu'il y a encore des actions à décrire ? Sinon, on quitte le modèle de domaine pour le modèle conceptuel. Ce nœud marque la sortie du modèle de domaine.

5 : Faut-il formaliser encore des actions ? S'il n'y en a pas, on quitte le modèle conceptuel pour le modèle opérationnel. Ce nœud indique la sortie du modèle conceptuel.

6 : Doit-on traiter d'autres actions ? Sinon, on quitte le modèle opérationnel. Ce nœud établit la sortie du modèle opérationnel et la fin de la méthodologie .

α **:** La condition est que le comportement traité en B l'a été aussi en D.

β **:** La condition est que tous les comportements traités en D l'ont été aussi en B.

γ **:** Il faut que tous les comportements aient été traités en C.

Pour faciliter la compréhension du diagramme de la Figure 7.2, intéressons-nous uniquement à l'étape A. Le résumé de cette étape dans le chapitre précédent était :

- **Choisir un agent** ou s'il s'agit d'un nouvel agent :
  - Description de l'agent.
  - Ajout à une liste des types d'agents.
  - Définition du statut actif ou passif de chaque agent.
- **Décrire un comportement** :
  - Description en langage naturel du comportement.

  - Ajout du comportement à la liste des comportements de l'agent.
  - Définition de la priorité du comportement.
- **Définir l'algorithme de changement de priorité** (si besoin est).

Figure

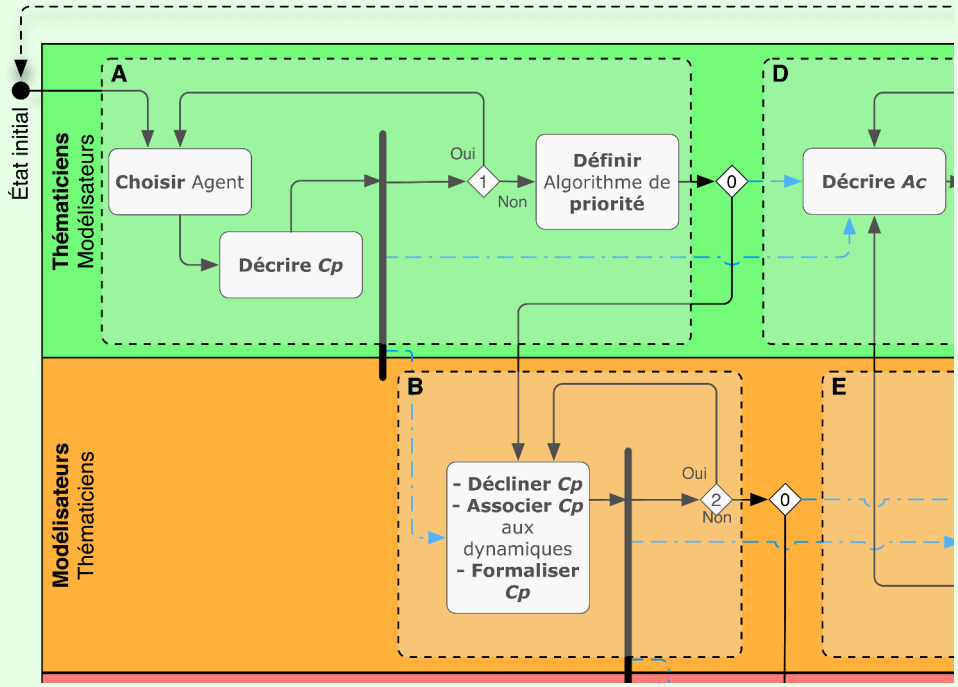


FIGURE 7.3 – Zoom sur l'étape A du diagramme UML des activités de la méthodologie

La Figure 7.3 constitue un agrandissement de cette étape, dont voici les trois grandes activités : le choix de l'agent (ou sa création le cas échéant), la description du comportement et la définition de l'algorithme de priorité. Après la description du premier comportement, trois options s'offrent :

- L'utilisateur continue à décrire d'autres comportements en retournant à l'étape A.
- Il peut décrire les actions relatives à ce comportement en passant à l'étape D (ou un autre utilisateur peut le faire à sa place).
- Il passe la main au second groupe d'acteurs qui se mettent à formaliser ce comportement à l'étape B.

Ces trois options peuvent se réaliser en parallèle (possibilité représentée sur le diagramme par la barre épaisse) : par exemple, pendant que le thématicien continue à décrire les autres

comportements, un modélisateur peut toujours formaliser le premier. Cependant, pour le déroulement d'un projet, il est préférable de procéder dans l'ordre des étapes et de suivre le "chemin conseillé" (flèches pleines) au lieu des "chemins alternatifs" (flèches en pointillé bleu). En effet, une fois les itérations de l'étape A terminées, passer par les étapes B puis C offre une plus grande efficacité pour l'étape D (qui permet de procéder à l'extension du modèle obtenu en A). Suivre le "chemin conseillé" revient à suivre un parcours traversant deux fois les différentes couches de modèles pour un affinage plus naturel.

Le reste des étapes de la Figure 7.2 est semblable à la première. Sur ce diagramme, l'état initial peut être le début du projet : on commence avec un projet vierge. Il est possible aussi d'utiliser un modèle conçu précédemment pour l'enrichir ou l'affiner. L'état final est un modèle stable prêt à être implémenté en un modèle informatique. Cette stabilité est obtenue grâce à une première phase de profilage due aux itérations des étapes A, B et C, puis par une deuxième phase : le processus d'affinement des étapes D, E et F. Le modèle obtenu en fin de parcours peut être réinjecté dans le processus pour amorcer un nouveau cycle de raffinement.

Il est possible de réitérer individuellement chacune des étapes le nombre de fois nécessaire. Si une modification importante est apportée par la suite sur le résultat d'une des étapes, il faut répercuter les changements dans les étapes suivantes. Par exemple, la modification de la description d'une action en D implique des changements relatifs à cette action sur E et F. Cette répercussion est soulignée dans le diagramme de classe de la Figure 7.4 par l'intermédiaire des relations de compositions.

Dans le prototype, nous représentons les agents, les comportements, les actions, les aMVC et les dynamiques comme des **objets**. Pour un agent par exemple, une hiérarchie s'établit comme dans la Figure 7.4. Un objet `Agent` est une composition de plusieurs `aMVC`. Remarquons dans le modèle de domaine que nous avons des objets `DomainBehavior`, chacun contenant une liste de `DomainAction`. Le modèle conceptuel offre une hiérarchie semblable pour les objets associés. Cependant, les objets du modèle conceptuel sont reliés à ceux du modèle de domaine. Par exemple, le `DomainBehavior` est lié à un `ConceptualBehavior` ou plusieurs, si plusieurs modélisateurs ont modélisé différemment le même comportement.

Nous pouvons au passage remarquer que ce diagramme de classe montre l'incidence du parcours des modèles en deux phases. Notamment la remontée réalisée lors de la transition de l'étape C à D, pour retourner du modèle opérationnel au modèle de domaine, est importante : l'identification des actions (sur laquelle repose la modélisation des instances `DomainAction` de l'étape D) nécessite d'avoir des instances d'`OperationalBehavior` (obtenues à l'étape C).

Figure

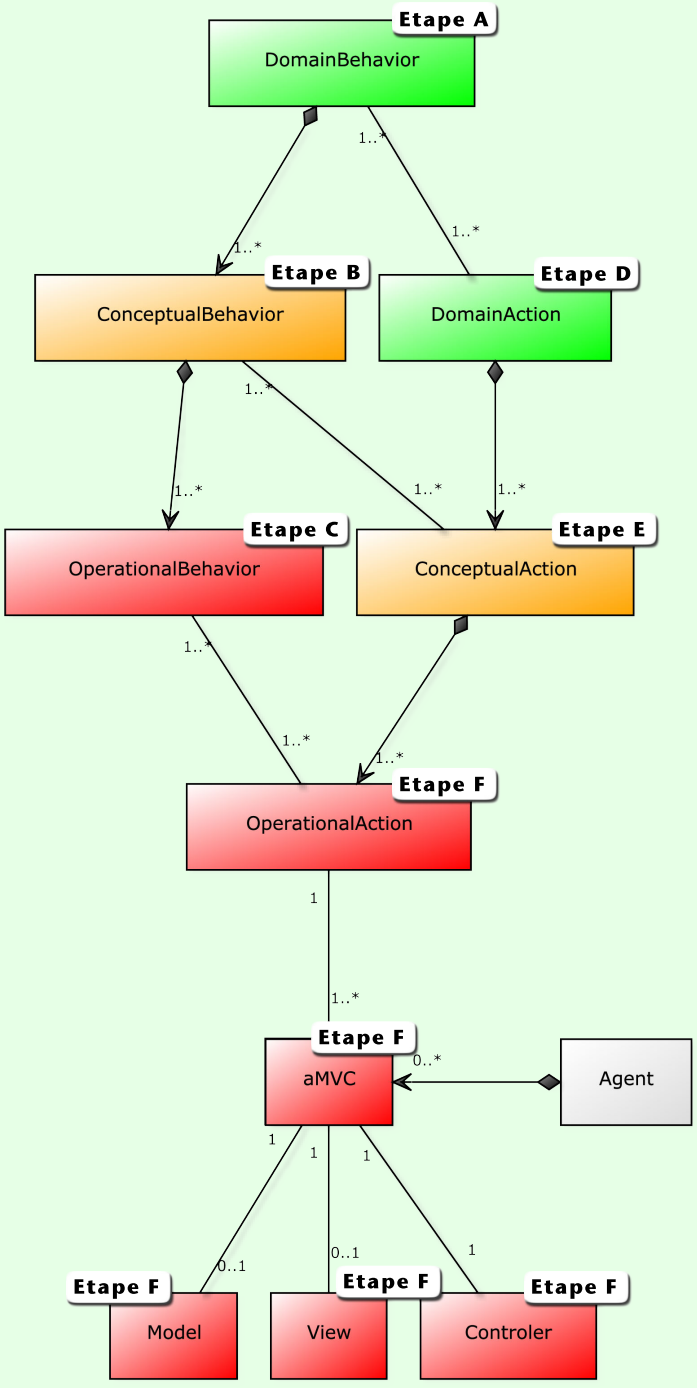


FIGURE 7.4 – Diagramme de classe simplifié d'un agent dans l'atelier

Les classes de ce diagramme se ventilent dans chacune des étapes de la façon suivante :

- `DomainBehavior` : Étape A, modèle de domaine.
- `ConceptualBehavior` : Étape B, modèle conceptuel.
- `OperationalBehavior` : Étape C, modèle opérationnel.
- `DomainAction` : : Étape D, modèle de domaine
- `ConceptualAction` : Étape E, modèle conceptuel.
- `OperationalAction`, `aMVC`, `Model`, `View` et `Controler` : Étape F, modèle opérationnel.

À chacune de ces étapes, le prototype assiste l'utilisateur dans la définition des objets correspondant à leur classe respective. Il en est de même pour les hiérarchies de classe relatives aux autres entités à modéliser (les dynamiques et les environnements).

7.2.3 Implémentation

L'implémentation de l'atelier par le framework jConverse se fait simplement par l'intermédiaire des annotations. Celles-ci permettent une certaine richesse en terme de manipulation. Parmi les annotations, nous pouvons citer en particulier deux annotations héritées de Hibernate :

- `@Entity` : Cette annotation précise que l'objet est une entité qui sera stockée dans la base de données. Dans le cas d'emploi de cache, il est possible d'utiliser `@javax.persistence.Cacheable(true)`.
- `@ManyToMany`, `@OneToOne`, `@OneToMany` et `@ManyToOne` : Ces annotations désignent les relations entre les objets, respectivement plusieurs-à-plusieurs, un-à-un, un-à-plusieurs et plusieurs-à-un. L'option `mappedBy` crée une relation bidirectionnelle. L'option `cascade` permet de cascader les opérations effectuées sur l'entité.

Les annotations spécifiques à jConverse permettent de générer les interfaces de saisies, avec mise en évidence du champ d'action associé à la méthode suivant ces annotations. Parmi les balises les plus intéressantes, citons :

- `@Order` qui précise l'ordre d'affichage de l'interface de la méthode.
- `@PresentationChild` qui facilite le formatage des interfaces de saisies de la méthode (nombre de colonnes, alignements, polices...).
- `@Choice` définit le type d'élément d'interface graphique qui affichera les objets (sous forme de listes déroulantes, de ComboBox ou de résumés) et les différentes options d'édition (avec ou sans boutons).
- `@Zoom` associe une option "Voir" aux objets de la liste, ce qui aide l'utilisateur à obtenir des informations détaillées sur l'objet concerné.

Pour créer un objet `DomainBehavior` représentant un comportement de domaine, nous le préfacerons par :

```
    @Entity
    @javax.persistence.Cacheable(true)
    @Parameter
    public class DomainBehavior extends GenericObject {
      private Set<DomainAction> domainActions;
      private Set<ConceptualBehavior> conceptualBehaviors;
//[Suite du code]
    }
```

`GenericObject` est la classe abstraite contenant les informations communes à toutes les autres classes. `DomainBehavior` contient deux collections : `DomainAction` et `ConceptualBehavior`. Les méthodes concernant ces collections sont introduites de la manière suivante :

```
//[Code précédent]
      @Order(1600)
      @ManyToMany(cascade = CascadeType.REFRESH, fetch = FetchType.LAZY)
      @PresentationChild(PresentationChild.Attributes.FORCE_NAME_TO_LEFT)
      @Choice(value = Choice.Mode.IN_LINE_EDITION,
              display = Choice.Display.RESUME)
      @Zoom
      public Set<DomainAction> getDomainActions() { //[..]
      }
      public void setDomainActions(Set<DomainAction> domainActions) {
       //[..]
      }
      public DomainBehavior addDomainActions(DomainAction... domainActions) {
       //[..]
      }
//[Suite du code]
```

Chaque classe a été renseignée en suivant cette structure. Dans l'implémentation, certaines cardinalités seront assouplies pour permettre à l'utilisateur d'avoir un diagramme incomplet pendant la création, mais seront gérées par des règles de cohérence. Une fois les objets métiers implémentés, nous avons commencé une modélisation probatoire du projet **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation (EDMMAS) dans lequel nous avons mis en place certains des agents du projet originel. L'interface de l'atelier se présente comme sur la Figure 7.5, dans le cas d'un `DomainBehavior` :

1. **Ligne du haut :**

    - Ordre de l'étape (A à F)
    - Nom de l'atelier et fil d'Ariane (pour se situer au sein du processus)
    - Type de modèle en cours (modèle de domaine, conceptuel ou opérationnel)

Figure

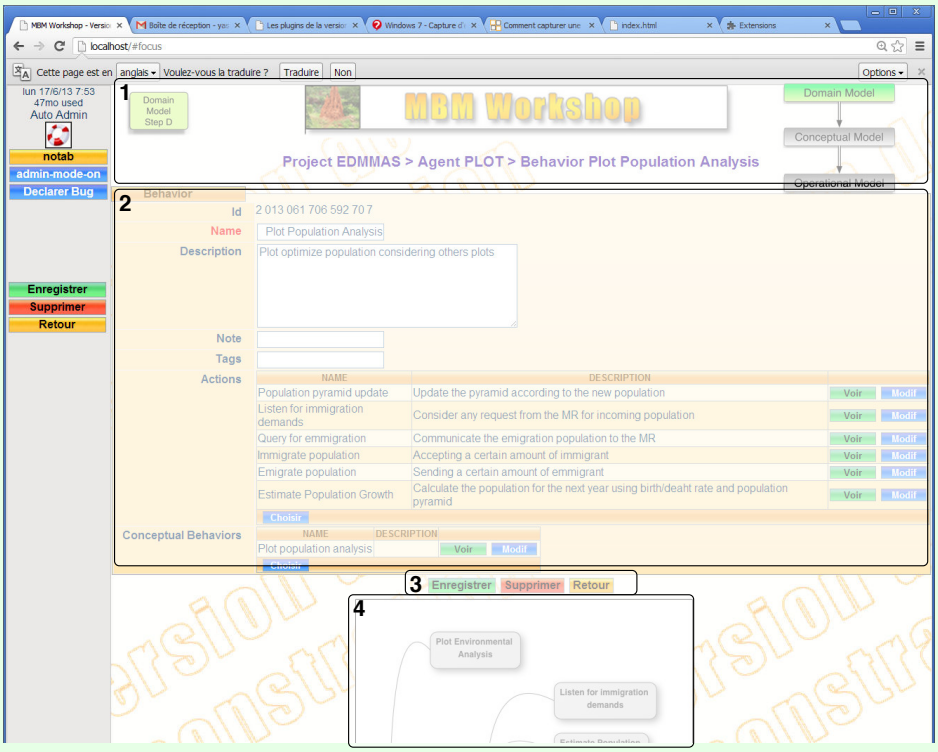


FIGURE 7.5 – Interface de l'atelier (étape D : liste des actions d'un comportement de l'agent PLOT)

2. **Cadre d'édition :**

   - Champs communs à toutes les entités :

     – **Id :** Identificateur unique de l'objet en cours.
     – **Nom :** Nom de l'objet.
     – **Description :** Description de l'objet.
     – **Note :** Notation de la pertinence de l'objet. On fait une moyenne de plusieurs notations des différents utilisateurs pour choisir le meilleur objet en fonction de leurs avis.
     – **Tag :** Étiquettes facilitant une recherche rapide de l'objet. Ici, il s'agit de simples mots sur lesquelles on peut faire une recherche. Mais il possible de faire évoluer ce champ, pour une meilleure organisation, par l'intermédiaire de taxonomie ou de thesaurus.

  - **Créé (non affiché) :** Date de création de l'objet et identité de son auteur.
  - **Modifié (non affiché) :** Date de dernières modifications et leur auteur.
  - **Statut (non affiché) :** Valide/Brouillon/Abandonné/Erreur...
  - **Similarité à (non affiché) :** L'objet est-il similaire à un autre ? Par exemple, il peut exister deux propositions différentes pour un même comportement. Le choix pourra se faire par la note.

- Champs spécifiques à certains objets :

  - **Zones de texte :** Elles permettent d'ajouter les détails nécessaires pour l'objet en question (fonction, but, formalisation...).
  - **Listes déroulantes :** Elles facilitent certains choix (Actif/passif...).
  - **Listes d'objets modifiables :** Elles offrent une manière d'ajouter d'autres objets en rapport avec l'objet en question. Par exemple, ici le comportement "Plot Population Analysis" possède une liste d'objets de type "Action" et une liste d'objets de type "Conceptual Behaviors". Celle-ci s'avère utile lorsque deux modélisateurs proposent des formalisations distinctes du même comportement. Les opérations possibles sont :

    * **Voir :** pour consulter les informations de l'objet correspondant.
    * **Modifier :** pour modifier l'objet en question.
    * **Choisir :** pour choisir un objet dans une liste d'objets du type voulu ou en créer une nouvelle. Par défaut, le prototype utilise la même instance d'objet. Cependant, il est possible de cloner cet objet pour permettre l'édition de ses attributs, ce qui remplira le champ "Similarité à" automatiquement.

3. **Bouton par défaut :** Enregistrer, Supprimer et Retour
4. **Visualisation de l'agent en cours :** pour visualiser l'agent concerné par les modifications en cours.

Nous avons développé des outils spécifiques (pour la génération de fichier `XML`, de *templates* GEAMAS-NG...). La conception des objets métiers et le développement des utilitaires ont été fait sur Eclipse. L'atelier (voir Figure 7.6) a été installé sur une machine virtuelle faisant office de serveur, possédant une installation fonctionnelle de `Java EE`, 1 Go de mémoire RAM et 0,5 Go de disque dur pour le framework, dont quelques Mo pour la base de données. Le poste client n'a besoin que d'un navigateur Web.

Figure

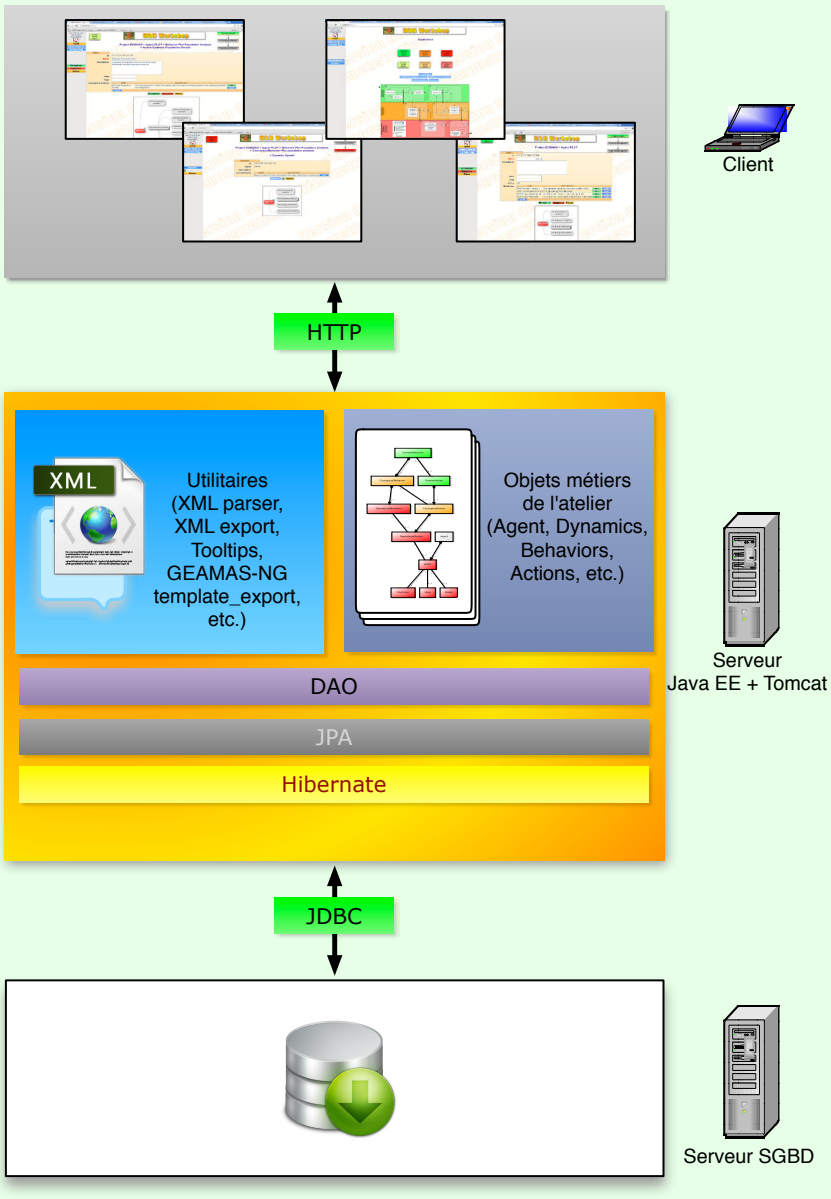


FIGURE 7.6 – Architecture informatique de l'atelier

7.2.4 Synthèse

Cet atelier de modélisation est un prototype accompagnant les acteurs du projet pour la création des modèles suivant les étapes A à F, pour obtenir un modèle opérationnel. La sortie qu'offre cet atelier est une structure XML représentant le système que l'on peut résumer suivant la DTD[2] :

```
<!ELEMENT System  ( Agent+ , Dynamic+ , DomainBehavior+ , DomainAction+ )>

<!ELEMENT Dynamic  ( InduceAttributes , Environnement )>
<!ELEMENT Environnement  ( #PCDATA )>
<!ELEMENT InduceAttributes  ( #PCDATA )>

<!ELEMENT Agent  ( aMVC* )>
<!ELEMENT aMVC  ( Model , View , Controler , OperationnalActionRelated )>
<!ELEMENT OperationnalActionRelated  EMPTY>
<!ELEMENT Controler  ( #PCDATA )>
<!ELEMENT View  ( #PCDATA )>
<!ELEMENT Model  ( #PCDATA )>

<!ELEMENT DomainBehavior  ( ConceptualBehavior+ , OperationnalBehavior+ )>
<!ELEMENT OperationnalBehavior  ( #PCDATA )>
<!ELEMENT ConceptualBehavior  ( #PCDATA )>

<!ELEMENT DomainAction  ( ConceptualAction+ , OperationnalAction+ )>
<!ELEMENT OperationnalAction  ( #PCDATA )>
<!ELEMENT ConceptualAction  ( #PCDATA )>
```

Ce qui nous donne un fichier XML du type :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document created by: Yassine Gangat, MBM Workshop -->

<System ProjectName=" " >
<!-- Under this line are listed informations usefull to programmers  -->

 <Agent Name=" " Description=" " Id=" " Note=" " Tag=" " Created=" " >
      <aMVC Name=" " Description=" "  Id=" " Note=" " Tag=" " Created=" " >
         <Model>Variables </Model>
         <View> Text </View>
         <Controler> Algorithm </Controler>
         <OperationnalActionRelated Name=" " Id=" " BelongToOperationalBehavior=" "/>
      </aMVC>
 </Agent>

 <Dynamic Name=" " Description=" "  Id=" " Note="" Tag=" " Created=" " >
```

[2] Nous avons volontairement masqué la liste des attributs par simplification (ATTLIST). Il s'agit des champs nom, description, id...pour chaque balise

```
      <InduceAttributes>  </InduceAttributes>
      <Environment Name=" " > Attributes of environment </Environment>
 </Dynamic>

<!-- Under this line are listed general informations -->

 <DomainBehavior Name=... >
      <ConceptualBehavior Name=... > As actor of behavior... </ConceptualBehavior>
      <OperationnalBehavior Name=... > Formalism </OperationnalBehavior>
 </DomainBehavior>

 <DomainAction Name=... >
      <ConceptualAction Name=... > Formalism </ConceptualAction>
      <OperationnalAction Name=... > aMVC </OperationnalAction>
 </DomainAction>

</System>
```

Cette décomposition nous donne une structure aMVC par action. Comme nous l'avons précisé dans les chapitres précédents, il est nécessaire de regrouper tous les actions du comportement sous un même aMVC. De plus, il est préférable de fusionner les Modèles de même plan comportemental et les Vues de même dynamique, pour éviter la redondance de certaines informations du Modèle ou des Vues.

Pour obtenir ensuite le modèle informatique, l'aspect humain est indispensable pour finaliser le processus : ce sont les programmeurs qui interviendront pour écrire le code final spécifique à la plateforme ou au langage cible. Ils interprèteront la sortie proposée afin de l'adapter en fonction des caractéristiques implémentatoires du contexte cible.

Notre approche a été de concevoir cet atelier pour un contexte générique et non dans celui, déjà spécifique, de notre plateforme de simulation GEAMAS-NG, d'où cette sortie `XML` neutre. Cependant, selon la plateforme souhaitée, on peut disposer d'une bibliothèque capable de produire des *templates* de code pré formatés à partir des informations dans la base de données de l'atelier. Par exemple, pour GEAMAS-NG, nous avons un agrégateur de code qui permet de créer des *templates* pour les agents, avec en commentaires les informations nécessaires aux programmeurs pour implémenter les agents plus efficacement.

7.3 Test dans le cadre du projet EDMMAS

EDMMAS est un projet arrivé à terme. Pour tester notre prototype d'atelier, nous avons tenté de recréer la modélisation des agents de ce projet. Pour éviter toute interférence par rapport à nos connaissances précédentes, nous avons demandé à une tierce personne, ayant des connaissances de base dans les agents et la modélisation, de créer certains agents (notamment l'agent `Parcelle` appelé ici `Plot`).

Il ne s'agit donc pas d'effectuer un test "grandeur nature" mais d'illustrer comment se déroulent les étapes et comment se cristallise l'information au travers de l'approche MMC.

Figure

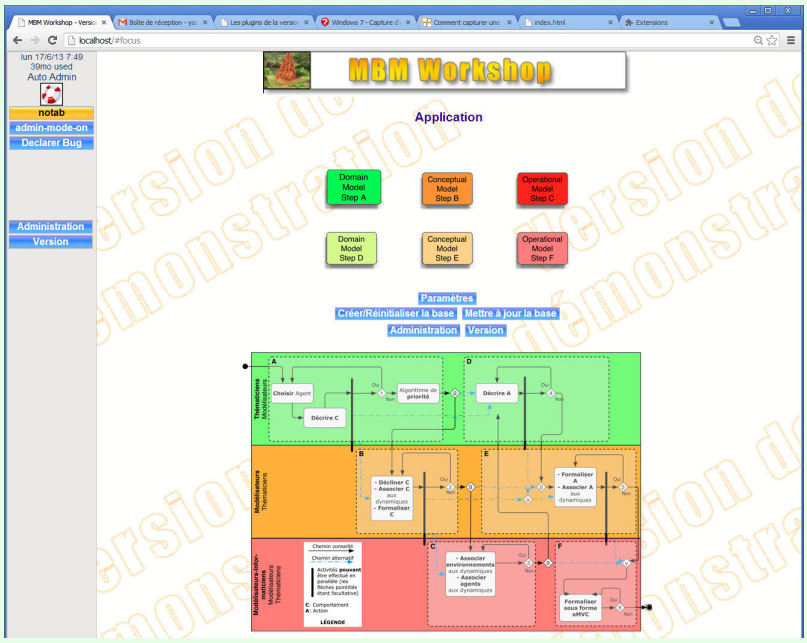


FIGURE 7.7 – Démarrage de l'atelier

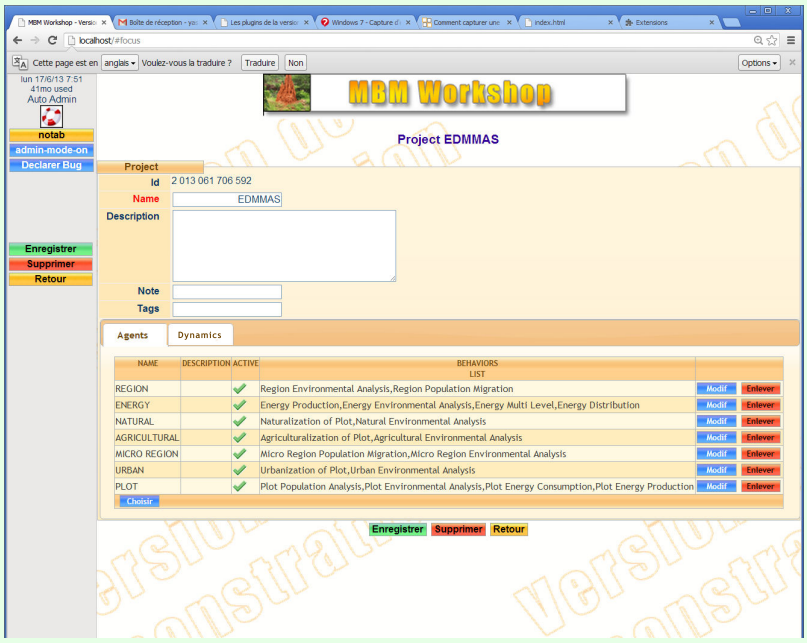


FIGURE 7.8 – Informations sur le projet

Lorsque l'interface se lance sur le projet déjà existant (voir Figure 7.7), un rappel de la méthodologie est présenté, avec un accès direct aux différentes étapes. Dans notre cadre de test, l'utilisateur a accès à toutes les étapes. Il est cependant possible de restreindre l'accès à certaines sections par l'intermédiaire de *logins*. Nous pouvons aussi consulter les informations sur le projet en cours (voir Figure 7.8). Cet affichage offre un résumé du projet : les agents du système et les dynamiques, auxquelles sont rattachés les environnements.

Figure

FIGURE 7.9 – Étape A : Description des comportements de l'agent PLOT

Les différentes étapes construisent au fur et à mesure les éléments du système. Ainsi, l'étape A (voir Figure 7.9) offre la possibilité de visualiser les comportements déjà dans l'agent, d'en choisir un parmi ceux déjà créés ou d'en décrire un nouveau. Il en va de même pour les autres étapes.

Figure

FIGURE 7.10 – Étape C : Association des environnements aux dynamiques

Par exemple, l'étape C permet la création de la dynamique spatiale réutilisée ensuite dans les autres parties (notamment l'étape E de la Figure 7.11). L'étape F (voir Figure 7.12) nous montre la formalisation de l'action sous la forme aMVC.

Figure

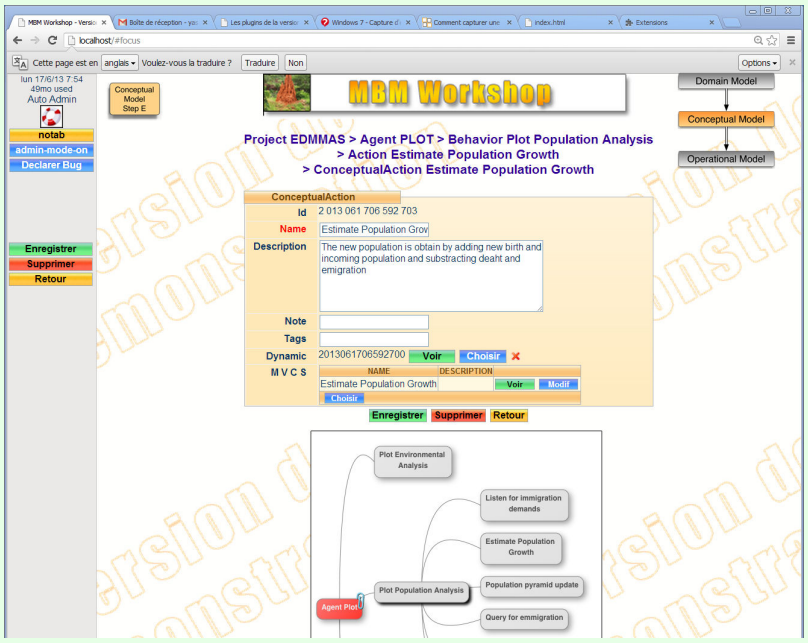


FIGURE 7.11 – Étape E : Formalisation d'une action d'un comportement de l'agent PLOT

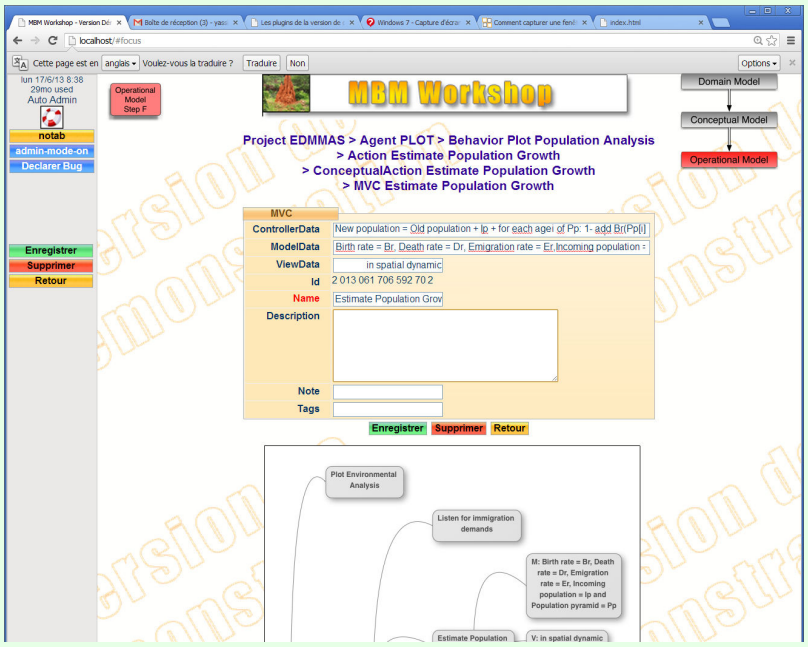


FIGURE 7.12 – Étape F : Mise sous forme aMVC d'une action d'un comportement de l'agent PLOT

L'agrégation de ces différentes formalisations débouche sur tous les comportements de l'agent `Plot`. La sortie `XML` obtenue sur ce test est la suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Document created by: Yassine Gangat, MBM Workshop -->

<System ProjectName="EDMMAS" >
<!-- Under this line are listed informations usefull to programmers  -->

  <Agent Name="Plot" Description="Agent that represent a piece of land"
         Id="2 013 061 706 592 700" Note="" Tag="" Created="" >

     <aMVC  Name="Estimate Population Growth"
            Description="Estimate Population Growth"
            Id="2 013 061 706 592 702" Note="" Tag="" Created="" >
          <Model>
              Birth rate = Br, Death rate = Dr,
              Emigration rate = Er, Incoming population = Ip
              and Population pyramid = Pp
          </Model>
          <View> Spatial dynamic </View>
          <Controler>
              New population = Old population + Ip
                           + for each age bracket i of Pp:
                             1- add Br(Pp[i])
                             2- substract Dr(Pp[i])
                             3- substract Er(Pp[i])
          </Controler>
          <OperationnalActionRelated Name="Estimate Population Growth"
                                     Description="The new population is obtained
                                     by adding new birth and incoming population
                                     and substracting death and emigration"
                                     BelongToOperationalBehavior="Plot population analysis"
           />
     </aMVC>
<!-- [autres aMVCs] -->
  </Agent>
<!-- [autres Agents] -->
  <Dynamic Name="Spatial Dynamic" Description=""  Id="2 013 061 706 592 500"
           Note="" Tag="" Created="" >
     <InduceAttributes>Geographical Position </InduceAttributes>
     <Environnement Name="Reunion Island" >
         Partialy observable, Discrete, Determinist, Sequential
     </Environnement>
  </Dynamic>
<!-- [Reste du code] -->
</System>
```

7.4 Conclusion

L'approche méthodologique MMC proposée dans le chapitre précédent a été implémentée sous forme d'atelier dans ce chapitre. Cet atelier permet d'accompagner les acteurs du projet dans la création des modèles, en gérant la mutualisation de leurs connaissances suivant les schémas cadrés par note approche méthodologique MMC.

Le résultat produit en sortie de cet atelier est un fichier `XML` qui expose la structure du code de l'agent sous la formalisation aMVC, l'ensemble des dynamiques et environnements, et les informations associées. Ce document sera utilisé par les programmeurs pour implémenter ensuite le modèle opérationnel.

La pierre manquante pour aboutir à un outil "*ready to run*" serait une bibliothèque spécifique à chaque plateforme capable d'interpréter de manière automatique le fichier XML produit par l'atelier en un modèle informatique, en fournissant le code complet de tous les constituants... mais l'élaboration d'une telle bibliothèque serait sans doute à elle seule un travail de thèse à part entière.

8 Conclusion générale

> Une personne qui n'a jamais commis d'erreurs n'a jamais tenté d'innover.
>
> — *Albert Einstein*

8.1 Synthèse des travaux

La co-construction et la réutilisation de modèles font l'objet de plusieurs travaux dans le domaine de la simulation. Cependant, dans le domaine plus spécifique de la **S**imulation **O**rientée **A**gent (SOA), nous constatons un manque de méthodologie sur ces deux points malgré un besoin fort de la part des thématiciens.

Les recherches spécifiques à la problématique de co-construction d'une SOA sont rares, car les modélisateurs sont le plus souvent en collaboration étroite pour la création des modèles. Dans des SOA de petite envergure, l'informaticien devient lui-même le modélisateur et l'interlocuteur entre les différents experts. La question reste en suspens pour les projets de grande envergure, où souvent ce sont les modélisateurs qui ont pour "devoir" de réunir, comprendre et modéliser les "visions" des thématiciens.

Concernant la réutilisation de modèle, il existe un spectre de réutilisation, allant de l'utilisation de petites portions de code, en passant par les grands composants, pour arriver au modèle complet. Plusieurs obstacles s'opposent à une réutilisation optimale. Nous pouvons citer notamment le manque de motivation des développeurs de modèles à adopter des procédures permettant la réutilisation de leurs créations. En effet, ces procédures induisent une augmentation du coût de développement du modèle (sans en percevoir les avantages, car ces créations seraient acquises par d'autres). D'autre part, une des principales préoccupations est le niveau de confiance d'un code obtenu dans un autre contexte. Enfin demeurent le temps et le coût de la familiarisation avec le code de quelqu'un d'autre, ce qui peut excéder le temps et les coûts-avantages de la réutilisation.

Pour atteindre ces deux objectifs, nous avons proposé une architecture agent suivant une approche multicomportementale basée sur le pattern **a**gent **MVC** (aMVC).

Les deux axes que nous avons décidé d'explorer dans ce mémoire ont été les suivants :

1. **La co-construction de modèles**, ou comment faciliter la conception de modèles dans un projet pluridisciplinaire ? Comment améliorer la collaboration entre thématiciens pour la construction d'une SOA ?
2. **La réutilisation de modèles**, ou comment faciliter la conception de modèles pour leur réemploi ? Comment faire pour que les prototypes, souvent très riches en connaissances, survivent à la fin des projets pour lesquels ils ont été initialement conçus ?

Pour bien cerner ces questionnements, nous nous sommes confronté à un terrain d'expérimentation réel : le modèle **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation (EDMMAS), basé sur **M**odélisation **O**rientée **D**ynamique (DOM) qui est une première approche aidant à la réutilisation de modèle. Après avoir montré les limites de cette dernière sur la gestion de la conception des agents, nous avons proposé le Design Pattern aMVC ainsi que son application dans un cadre multidynamique. Enfin, nous avons proposé une approche méthodologique de **M**odélisation **M**ulti**C**omportementale (MMC) permettant de guider le processus de co-construction établie sur la base de la nouvelle architecture agent découlant de ce pattern.

8.1.1 Contributions

Dans un premier temps, nous présentons la Modélisation Orientée Dynamique (DOM), méthodologie introduite par notre équipe précédemment, à travers une expérimentation sur le passage à l'échelle de ce concept. Si cette approche est efficace au niveau conceptuel et particulièrement pour la gestion des environnements, nos travaux ont permis de révéler un décalage conséquent entre le modèle opérationnel et le modèle conceptuel. Ce décalage causera ensuite un travail important lors de l'implémentation des agents, notamment pour la description de leurs comportements impliquant plusieurs dynamiques. Plus les agents sont corrélés à des dynamiques différentes, plus ils deviennent délicats à appréhender sur le plan implémentatoire en induisant une plus grande complexité dans les modèles opérationnel et informatique.

Ce décalage est une source potentielle d'erreur au moment de l'opérationnalisation puis de l'implémentation des modèles. Pour limiter cet écart et rendre les modèles plus cohérents, nous proposons un modèle d'architecture interne d'agent aMVC, s'inscrivant en complément de DOM.

Cette vision de l'architecture étant novatrice, il nous semble important d'accompagner cette proposition d'une approche méthodologique de conception appropriée à ce nouveau cadre, la MMC. Cette méthodologie nous permet de simplifier l'utilisation de aMVC, car celle-ci peut apparaître comme délicate au regard du nombre important de sous-constituants qui peuvent être induits au niveau des agents.

Trois principales contributions des chapitres centraux ressortent de ce travail de thèse et s'organisent selon la Figure 8.1.

1. **La proposition d'un modèle simulant la gestion de l'énergie à La Réunion : EDMMAS,**

Figure

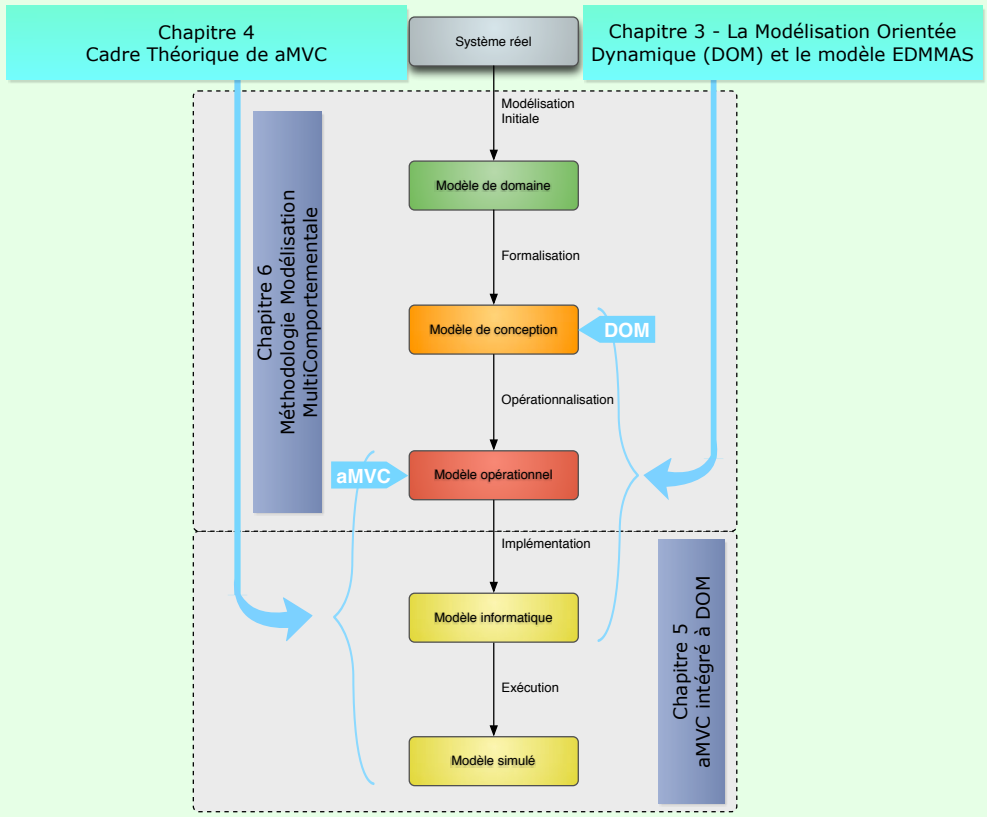


FIGURE 8.1 – Organisation des contributions de la thèse

et sa mise en place sous la forme d'un prototype éprouvant le passage à l'échelle de DOM (voir Chapitre 3). Il s'agit d'un cas concret d'une démarche pour la réutilisation de modèle, conduisant à la mise en évidence de la problématique sur laquelle portent les autres contributions issues du travail de cette thèse.

DOM est une méthodologie, à laquelle nous avons contribué lors de sa construction au sein de notre équipe de recherche. Elle décompose un système complexe en sous-systèmes sur la base d'un ensemble de dynamiques pré-identifiées, dans le but de réduire la complexité. L'environnement est ensuite utilisé comme milieu de couplage des dynamiques permettant au modèle global de former un tout. Cette approche atténue les difficultés de la réutilisation dans la modélisation d'un système complexe, notamment par l'introduction du concept de dynamiques dans le modèle conceptuel. Par ailleurs, cette méthodologie facilite la gestion du multienvironnement.

EDMMAS est un modèle qui simule la gestion d'énergie, conjointement avec l'évolution

de la population et l'urbanisation du territoire. Le prototype issu de ce modèle utilise DOM dans un cadre multidynamique. Grâce à celui-ci, nous avons montré que DOM a permis d'enrichir de manière incrémentale une SOA, par la mise en place d'une nouvelle dynamique sans déstabiliser ni remettre en cause l'organisation et la conception des autres déjà existantes. Cette dynamique, modélisant les transactions électriques[1], a été introduite de manière complètement modulaire et disjointe, gardant la cohérence globale du système, tout en étant en relation avec deux autres dynamiques issues de précédents projets : l'évolution de la population et l'urbanisation du territoire.

2. **La proposition d'un modèle d'architecture opérationnelle d'implémentation des agents, basé sur les designs patterns : le modèle aMVC** (voir Chapitres 4 et 5). Ce modèle a été introduit pour pallier le décalage entre le modèle opérationnel et le modèle conceptuel produit dans les situations de réutilisation de modèle et mis en évidence au travers de notre première contribution. Son application donne une architecture informatique de description d'une SOA en cohérence avec une modélisation multidynamique. Il organise les composants informatiques qui vont représenter le modèle conceptuel, pour faciliter l'implémentation et gérer ce décalage. Ce modèle est inspiré des Design Patterns du monde objet, en particulier du **M**odèle **V**ue **C**ontrôleur (MVC), qui, malgré sa simplicité apparente, a facilité notamment la réutilisation des modules des applications multivues, multifenêtres... Nous avons transposé MVC aux besoins du monde agent tel que nous l'abordons.

   Grâce à ce pattern, nous avons pu réaliser un processus de décomposition structurelle de l'agent, dissociant les entités qui le composent afin de mettre en relation la couche conceptuelle multicomportementale avec la couche informatique sous-jacente capable de gérer ce modèle de façon opérationnelle. Dans notre proposition sur le plan informatique, chaque agent répond donc à une structuration simple aMVC formalisant des interfaces opératoires adaptées à la gestion fine de mécanismes de partage de comportements agissant dans des environnements distincts.

   Cette structuration est fondée sur les trois composantes que nous avons identifiées dans l'agent : ses états, ses comportements et ses interacteurs (effecteurs et récepteurs). Après un premier fractionnement selon les dynamiques puis selon les plans comportementaux, concept introduit pour représenter les domaines d'expertise comportementale, nous faisons une scission suivant ces trois composantes, représentant respectivement le Modèle, le Contrôleur et la Vue de aMVC. Ces différentes couches aMVC, par agrégation, forment les briques élémentaires de notre agent. Du point de vue du modélisateur, aMVC rend ce mécanisme transparent, car le processus de recomposition de l'agent est automatisé au niveau conceptuel.

   Nous avons validé ce modèle en l'appliquant sur un cadre multidynamique, en l'intégrant à DOM.

3. **La proposition d'une approche méthodologique adaptée à cette nouvelle architec-**

[1] Il s'agit de la production, consommation et gestion du courant électrique.

ture : MMC (voir Chapitres 6 et 7). Cette méthodologie se déroule en six phases (listées de A à F) découpant précisément, avec des étapes identifiées, la modélisation d'un système complexe. Chaque étape est définie par un ensemble de descriptions élémentaires. Elle peut être représentée sous forme de tableaux, incluant les notions d'actions, de comportements, de dynamiques... Il s'agit donc de représenter les différents éléments acteurs du système complexe de la manière la plus simple et systémique pour construire la simulation à partir de ces briques élémentaires. Cette méthodologie, mise en œuvre au travers de la réalisation d'un prototype probatoire, montre ainsi l'automatisation du processus et les possibilités d'inférences sur ses éléments.

Elle permet de conduire un projet pluridisciplinaire de SOA avec un grand nombre d'acteurs, facilitant la co-construction des modèles grâce à la synergie créée. Les concepteurs pourront travailler de manière autonome sur leur dynamique et la plateforme en fera l'intégration en assurant la cohésion et la robustesse du système. Basé sur l'approche DOM et notre nouveau pattern aMVC, cette méthodologie offre donc la capacité de créer les briques élémentaires du système de manière indépendante, de les associer et de les combiner pour former des agents, selon des dynamiques. Elle permet ainsi de comparer la logique de différentes possibilités pour une même dynamique et d'ouvrir la possibilité d'étudier un grand nombre d'alternatives de modélisation d'un même système complexe, et de les analyser ensuite à une échelle très fine.

Ces trois contributions forment un ensemble d'outils et de méthodologies qui facilitent la co-construction pluridisciplinaire et la réutilisation de modèles.

8.1.2 Publications

Nous avons publié plusieurs articles durant la rédaction de cette thèse. La majorité d'entre eux est directement liée à notre problématique et leur contenu est parfois repris, de façon diffuse, dans les différents chapitres de cet ouvrage :

- 1 article publié dans une revue internationale :
  - [Gangat *et al.*, 2012b] Y. Gangat, D. Payet and R. Courdier. **Methodology for a New Agent Architecture Based on the MVC Pattern**, *Lecture Notes in Computer Science - Artificial Intelligence : Methodology, Systems, Applications*, Vol.755 : 230 — 239.
- 3 articles publiés puis présentés dans des conférences internationales :
  - [Gangat *et al.*, 2012a]Y. Gangat, D. Payet and R. Courdier. **Another step toward reusability in agent-based simulation : Multi-behaviors & MVC**, *24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, Athens, Greece, nov.7-9, 2012.
  - [David *et al.*, 2012] D. David, Y. Gangat, D. Payet and R. Courdier. **Reification of emergent urban areas in a land-use simulation model in Reunion Island**, *ECAI*

workshop on Intelligent Agents in Urban Simulations and Smart Cities (IAUSSC'12), Montpellier, France, aug. 27-31, 2012.

- [Gangat *et al.*, 2010] Y. Gangat, M. Dalleau, D. David, N. Sebastien and D. Payet, **Turtles are the turtles**, *The European Simulation and Modelling Conference (ESM'10)*, Hasselt University, Belgique, oct. 25-27, 2010.

- 4 articles publiés et présentés dans des conférences nationales :

  - [Gangat *et al.*, 2013] Y. Gangat, D. David, D. Payet and R. Courdier. **Approche méthodologique pour la modélisation multicomportementale dans la SOA**, *Workshop de Modélisation Agents pour les Systèmes Complexes - Plate-Forme IA (MASyCo-PFIA)*, Lille, France, juil.2,2013.
  - [Gangat *et al.*, 2009a] Y. Gangat, D. David, D. Payet and R. Courdier. **Modéliser et simuler l'aménagement énergétique d'un territoire par les systèmes multi-agents**, *Colloque sur la Convergence des Réseaux, de l'Informatique et du Multimédia pour les E-services (CRIME'09)*, Saint-Denis de La Réunion, France, nov. 12-13, 2009.
  - [Gangat *et al.*, 2009b] Y. Gangat, R. Courdier and D. Payet, **Démonstration : Aménagement énergétique d'un territoire – Une approche par simulation multi-agents**, *Journées Francophones Systèmes Multi-Agents (JFSMA'09)*, Lyon, France, oct. 19-21, ed. Cepadues, 2009.
  - [David *et al.*, 2009] D. David, D. Payet, R. Courdier and Y. Gangat, **XELOC : un support générique pour la configuration et l'initialisation de systèmes multi-agents**, *Journées Francophones Systèmes Multi-Agents (JFSMA'09)*, Lyon, France, oct. 19-21, ed. Cepadues, 2009.

- 2 posters de vulgarisation réalisés et présentés aux Doctoriales de l'université de La Réunion

8.2 Perspectives

Notre approche repose sur une démarche ascendante en partant des concepts en amont de l'architecture – "les agents" – et non des concepts de haut niveau du système. Grâce au découpage résultant, nous sommes à même de donner un terrain d'action à chaque expert, et si nécessaire de diviser encore plus le travail entre les différents thématiciens via la structuration aMVC, en confiant une brique élémentaire (M, V ou C) à chacun. Cette modélisation nous aide dans la création de l'agent. En outre, exploiter les propriétés du modèle MVC permet de réutiliser et de personnaliser facilement n'importe quelle partie de n'importe quelle couche d'un agent.

Bien que le modèle aMVC ait été proposé pour pallier les limites du concept de DOM, il n'est pas lié à ce dernier, dans le sens où il peut être employé par celui-ci. Pour faciliter son

utilisation, nous avons proposé l'approche méthodologique MMC.

Si nous reprenons l'état de l'art présenté par les auteurs de [Pokahr et Braubach, 2009] (voir la première ligne de la Figure 8.2), nous pouvons placer MMC comme une approche méthodologique appliquée au niveau d'autres comme Gaia ou Prometheus. L'architecture choisie pour cette méthodologie est donc aMVC et DOM. La suite de ce travail s'inscrira dans la mise en place d'un langage adapté et d'un environnement de développement qui intégrera la plateforme de simulation.

Figure

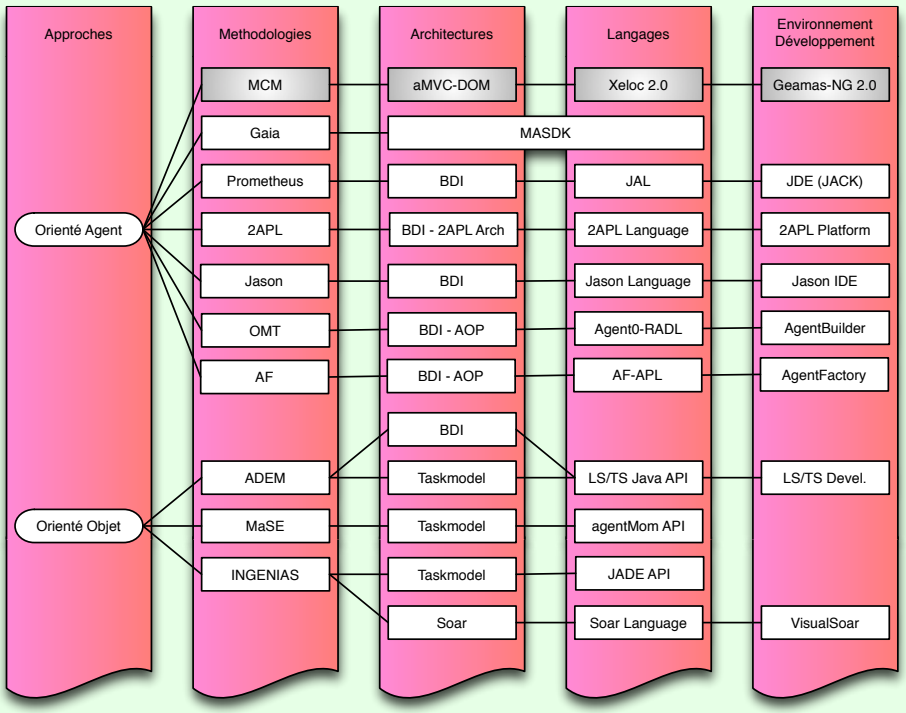


FIGURE 8.2 – Positionnement de nos propositions, basé sur une Figure de [Pokahr et Braubach, 2009]

Notre approche nous permet d'entrevoir aussi de nouveaux champs d'investigation, en particulier au niveau global de l'organisation des couches et sur le potentiel d'évolution dynamique de ses interconnexions. Peut-on généraliser tous les bénéfices du modèle MVC à aMVC ? Est-ce qu'un environnement ne serait pas juste un aMVC avec des propriétés particulières ? Serait-il possible d'appliquer la méthode Scrum en totalité grâce à cette méthodologie ? Ne faudrait-il pas réévaluer plus profondément ce que nous considérons comme gravé dans du marbre pour repenser plus largement l'agent ?

Les travaux issus de cette thèse ont apporté des contributions qui ont permis de répondre à la problématique initiale, mais elles entraînent aussi de nouveaux questionnements : au niveau développement ainsi qu'au niveau recherche, les deux étant, bien évidemment, intimement liés.

8.2.1 Perspectives générales

La première catégorie de perspectives concerne **le développement des plateformes de modélisation et de simulation**, notamment la nôtre : **GEAMAS N**ew-**G**eneration (GEAMAS-NG).

La mise en place efficace de cette approche méthodologique pour une utilisation générique, étant basée sur des nouveaux concepts et architectures, nécessite de repenser les plateformes de Simulation Orientée Agent, avec une architecture adaptée à aMVC implémentant MMC, une bibliothèque de modèles... Nous avons présenté quelques caractéristiques dans la Section 5.3.2, qui peuvent servir de base à un développement plus avancé.

L'idée est ici d'incorporer dans un même environnement du type de **AMP**[2] les deux aspects : la méthodologie de modélisation et la plateforme. Ce qui faciliterait une sorte d'Ingénierie dirigée par les modèles (Model Driven Engineering[3]) pour la modélisation et la simulation de systèmes complexes par la SOA [Garro *et al.*, 2013].

D'une part, les solutions aux problèmes liés à la coédition de comportements doivent être améliorées. Au lieu du "*last edit wins*" ou des systèmes de verrouillage usuels, il serait intéressant de mettre en place des algorithmes mieux adaptés (basés par exemple sur l'expertise de l'utilisateur), d'utiliser des techniques inspirées des outils collaboratifs en ligne tels que Google Docs, Etherpad (voir CoODD en Section 2.3.2)... ou encore des infrastructures génériques de collaboration (GCI pour **G**eneric **C**ollaboration **I**nfrastructure) comme le proposent les auteurs de [Heinrich *et al.*, 2012].

D'autre part, nous aimerions incorporer aussi des outils conventionnels de collaboration distribuée synchrone et asynchrone, ainsi que la possibilité d'offrir une manière d'interagir entre les acteurs après la conception de la simulation. Par exemple, nous pourrions permettre les changements de Vue ou de Contrôleurs en temps réel, au travers d'une interface durant l'exécution du modèle, ce qui nous ferait entrer dans un cadre de simulation participative.

Une autre perspective est de faciliter encore plus la modélisation en utilisant des concepts tels que *Google App Inventor*[4] [Tyler, 2011]. Cette application utilise un système de briques (voir Figure 8.3) qui s'emboîtent pour faciliter la création d'une application. L'idée serait d'utiliser des briques simples pour créer un aMVC qui servira ensuite de brique complexe. Ces ensembles de briques complexes permettraient de créer des agents, qui par la suite

[2] Agent Modeling Platform (AMP) est un framework basé sur Éclipse, permettant la représentation, l'édition, la génération, l'exécution et la visualisation de modèles orientés agents. Voir : http://eclipse.org/amp/

[3] Le MDE est une approche du développement qui préconise la focalisation des efforts sur le modèle applicatif plutôt que sur le code.

[4] *App Inventor* pour Android (http://appinventor.mit.edu/) est une application développée par Google. Elle est actuellement entretenue par le *Massachusetts Institute of Technology* (MIT). Elle facilite le développement des applications Android et le rend accessible à ceux qui ne sont pas très familiers avec les langages de programmation. Elle est basée sur une interface graphique semblable à Scratch et à celle de StarLogo.

Figure

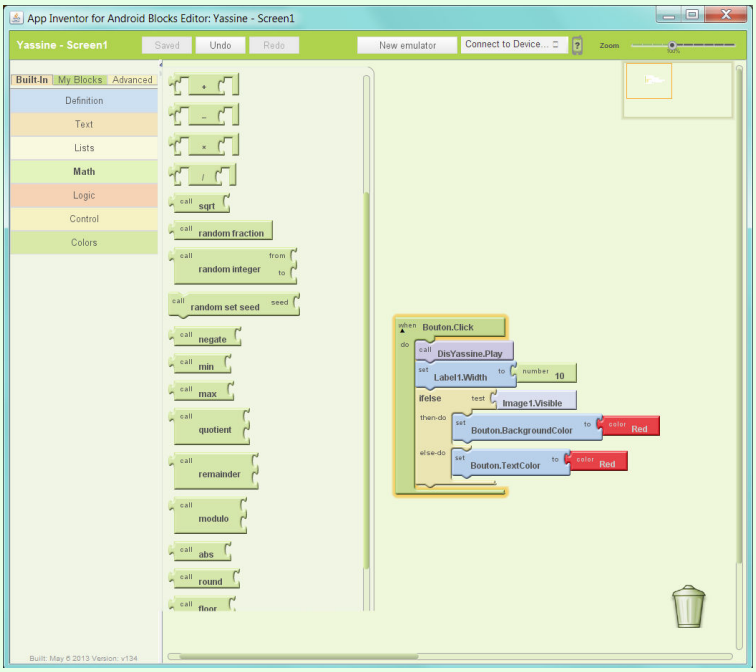


FIGURE 8.3 – Exemple de briques dans Google App Inventor

seraient eux-mêmes intégrés comme des briques du modèle. Nous aurions des bibliothèques de comportements, d'agents et de modèles.

Ces ensembles d'outils formeraient l'environnement de la nouvelle version de GEAMAS-NG.

La seconde catégorie de perspectives concerne **la mise en place d'ontologies et de langages adaptés** : une interface servant de passerelle entre cette approche méthodologique MMC et les simulateurs, qui ferait office de machine virtuelle agent (à l'instar d'une JVM) et qui fonctionnerait sur la plupart des simulateurs connus, avec des standards de type Knowledge Query Manipulation Language (KQML) ou Agent Communication Language (ACL) de FIPA (Foundation for Intelligent Physical Agents).

De plus, nous pouvons créer une ontologie de modélisation agent qui permettrait d'échanger facilement des modèles et de fournir une bonne démarche de conceptualisation.

Ces deux outils seraient une extension de e**X**tensible **E**diting **L**anguage **O**f **C**onfiguration (XELOC) en perspective d'une version `2.0`.

8.2.2 Perspectives personnelles

Dans la continuation naturelle de nos travaux, les pistes que j'ai énoncées ci-dessus devraient servir d'appui aux chercheurs de notre communauté pour utiliser les contributions de cette thèse.

Mais en ce qui me concerne, je compte m'investir dans deux voies pour lesquelles j'ai un intérêt tout particulier, dans le contexte où j'ai vécu dans mon laboratoire et au travers de mes échanges avec les chercheurs :

1. Continuer dans la lignée de nos recherches dans le domaine des systèmes complexes.
2. Partir sur une nouvelle voie et expérimenter le modèle aMVC dans le cadre de la robotique

Perspectives dans le cadre de la simulation des systèmes complexes

Il reste encore un certain nombre de limites liées à la co-construction de modèles collaboratifs. Je prévois des pistes de recherches intéressantes qui peuvent être menées dans cette optique : créer des modèles à plusieurs, les valider, les varier...

La première perspective de recherche se concentre sur la collaboration entre les participants et la production de nouvelles connaissances, transcendant MMC, en développant une démarche de co-construction dynamique des **S**ystème **M**ulti-**A**gents (SMA) sur les réseaux, où chacun des thématiciens pourrait modéliser à distance la connaissance et le comportement dynamique des entités auxquelles il s'intéresse au sein d'une Simulation Orientée Agent de systèmes complexes.

Ces connaissances seraient formalisées sous forme de modèles, dynamiques, agents ou comportements, disponibles avec plusieurs niveaux d'abstraction. Ces modèles de références s'améliorent au fur et à mesure des réutilisations ; ils peuvent coexister sous forme de "fork" lors des modifications successives, tout en gardant une certaine cohérence avec le modèle de référence.

Perspectives dans le domaine de la robotique multi-agent

Au sein de l'équipe, un axe nouveau est en train de naître. J'imagine bien, par rapport à cet axe, pouvoir expérimenter aMVC dans l'architecture des agents qui seraient intégrés dans le domaine de la robotique.

La seconde perspective de recherche que je désire explorer est la généralisation de l'approche aMVC pour la robotique multi-agents. Cette généralisation nous permettra, par exemple, de décrire des comportements réutilisables sur plusieurs types de robots sans que ces comportements soient totalement tributaires des capteurs et effecteurs du robot. L'utilisation de la flexibilité apportée par aMVC dans cette description pourrait être un plus.

J'espère avoir l'opportunité de pouvoir mettre en œuvre ces perspectives. Ces années de recherche ont été une expérience riche en apprentissages et découvertes. Les échanges internationaux et relations pluridisciplinaires m'ont permis de consolider mes connaissances

sur les systèmes complexes ainsi que sur l'informatique en général. Les activités au sein de mon laboratoire m'ont offert l'opportunité de découvrir de nouveaux axes technologiques, notamment la robotique avec Nao...

« — Les machines font de leur mieux. »
« — Ouais, mais c'est tout c' qu'elles peuvent faire. Quand y a pépin, l'homme, lui, doit faire plus que son mieux, sans ça il est foutu. »

— *Isaac Asimov*

Bibliographie

[Arc, 1992] (1992). A metamodel for the runtime architecture of an interactive system : the UIMS tool developers workshop. *SIGCHI Bull.*, 24(1):32–37. *Citation en page* ***79***

[Abras, 2009] ABRAS, S. (2009). *Système domotique Multi-Agents pour la gestion de l'énergie dans l'habitat.* Thèse de doctorat, Institut Polytechnique de Grenoble. *3 citations en pages* ***47, 56, et 58***

[Alexander *et al.*, 1977] ALEXANDER, C., ISHIKAWA, S. et SILVERSTEIN, M. (1977). *A Pattern Language : Towns, Buildings, Construction.* Oxford University Press, center for édition. *Citation en page* ***66***

[Amblard *et al.*, 2001] AMBLARD, F., FERRAND, N. et HILL, D. R. C. (2001). How a Conceptual Framework Can Help to Design Models Following Decreasing Abstraction. *SCS-European Simulation Symposium*, pages 843–847, Marseille, France. *3 citations en pages* ***23, 38, et 72***

[Amiguet, 2000] AMIGUET, M. (2000). *MOCA : Un modèle componentiel dynamique pour les systèmes organisationnels.* Thèse de doctorat, Université de Neuchatel. *Citation en page* ***37***

[Anderson *et al.*, 2003] ANDERSON, M., MCDANIEL, E. et CHENNEY, S. (2003). Constrained animation of flocks. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 286–297, Aire-la-Ville, Switzerland. Eurographics Association. *Citation en page* ***2***

[Aridor et Lange, 1998] ARIDOR, Y. et LANGE, D. B. (1998). Agent design patterns : elements of agent application design. *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 108–115, New York, USA. ACM. *Citation en page* ***73***

[Báez-Barranco *et al.*, 2007] BÁEZ-BARRANCO, J.-A., STRATULAT, T. et FERBER, J. (2007). A Unified Model for Physical and Social Environments. WEYNS, D., PARUNAK, H. et MICHEL, F., éditeurs : *Environments for Multi-Agent Systems*, volume 4389 de *Lecture Notes in Computer Science*, pages 41–50. Springer Berlin Heidelberg. *Citation en page* ***37***

[Balci, 1994] BALCI, O. (1994). Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Annals of Operations Research*, 53(1):121–173. *Citation en page* ***30***

[Balci *et al.*, 2011] BALCI, O., ARTHUR, J. D. et ORMSBY, W. F. (2011). Achieving reusability and composability with a simulation conceptual model. *Journal of Simulation*, 5(3):157–165. *2 citations en pages **9 et 25***

[Balci et Ormsby, 2007] BALCI, O. et ORMSBY, W. F. (2007). Conceptual modelling for designing large-scale simulations. *Journal of Simulation*, 1(3):175–186. *Citation en page **25***

[Beck et Cunningham, 1987] BECK, K. et CUNNINGHAM, W. (1987). Using Pattern Languages for Object-Oriented Programs. Technical Report CR-87-43, Apple Computer, Inc. and Tektronix, Inc. *Citation en page **67***

[Berends et van Tooren, 2007] BERENDS, J. P. T. J. et van TOOREN, M. J. L. (2007). Design of a Multi-Agent Task Environment Framework to Support Multidisciplinary Design and Optimisation. *Citation en page **9***

[Bodker, 1991] BODKER, S. (1991). *Through the Interface : A Human Activity Approach to User Interface Design.* L. Erlbaum Associates Inc., Hillsdale, NJ, USA. *Citation en page **84***

[Boman *et al.*, 1998] BOMAN, M., DAVIDSSON, P., SKARMEAS, N., CLARK, K. et GUSTAVSSON, R. (1998). Energy Saving and Added Customer Value in Intelligent Buildings. *Building*, 1(13):505–516. *Citation en page **47***

[Boronea *et al.*, 2009] BORONEA, , LEON, F., ZAHARIA, M. H. et ATANASIU, G. (2009). Design Patterns for Multi-agent simulations. 4(4):15–26. *Citation en page **73***

[Borshchev et Filippov, 2004] BORSHCHEV, A. et FILIPPOV, A. (2004). From System Dynamics and discrete event to practical Agent Based Modeling : reason, techniques, tools. *Conference of the System Dynamics Society*, Oxford. *Citation en page **18***

[Braun et Kroo, 1997] BRAUN, R. D. et KROO, I. M. (1997). Development and application of the collaborative optimization architecture in a multidisciplinary design environment. *Multidisciplinary Design Optimization : State of the Art*, pages 98–116. *Citation en page **9***

[Breunese *et al.*, 1998] BREUNESE, A., TOP, J. L., BROENINK, J. F. et AKKERMANS, J. M. (1998). Libraries of Reusable Models : Theory and Application. *Simulation*, 71(1):7–22. *Citation en page **19***

[Bucciarelli, 1994] BUCCIARELLI, L. L. (1994). *Designing engineers.* MIT Press, Cambridge, MA, USA. *Citation en page **16***

[Bucciarelli, 2003] BUCCIARELLI, L. L. (2003). Designing and learning : a disjunction in contexts. *Design Studies*, 24(3):295–311. *Citation en page **16***

[Burbeck, 1987] BURBECK, S. (1987). *Applications Programming in Smalltalk-80 : How to Use Model-View-Controller (MVC).* Softsmarts, Inc. *3 citations en pages **74, 81, et 90***

[Campos et Hill, 1998] CAMPOS, A. M. C. et HILL, D. R. C. (1998). An agent-based framework for visual-interactive ecosystem simulations. *Transactions Of The Society For Computer Simulation*, 15(4):139–152. *Citation en page **72***

[Chan *et al.*, 2010] CHAN, W. W. K. V., SON, Y.-J. Y. et MACAL, C. M. (2010). Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. *Winter Simulation Conference WSC Proceedings*, pages 135–150. *Citation en page **31***

[Cohn, 2004] COHN, M. (2004). *User stories applied : for agile software development.* Addison-Wesley, Boston. *Citation en page **128***

[Cossentino *et al.*, 2003] COSSENTINO, M., BURRAFATO, P., LOMBARDO, S. et SABATUCCI, L. (2003). Introducing pattern reuse in the design of multi-agent systems. *Proceedings of the NODe 2002 agent-related conference on Agent technologies, infrastructures, tools, and applications for E-services*, NODe'02, pages 107–120, Berlin, Heidelberg. Springer-Verlag. *Citation en page **73***

[Courdier *et al.*, 2002] COURDIER, R., GUERRIN, F., ANDRIAMASINORO, F. et PAILLAT, J.-m. (2002). Agent-based simulation of complex systems : Application to collective management of animal wastes. *Journal of Artificial Societies and Social Simulation*, 5(3):23. *3 citations en pages **1, 2, et 3***

[Coutaz *et al.*, 1995] COUTAZ, J., NIGAY, L. et SALBER, D. (1995). Agent-based architecture modelling for interactive systems. *International Conference on Systems Man and Cybernetics*, 6:191–209. *Citation en page **79***

[Coutaz *et al.*, 1996] COUTAZ, J., NIGAY, L. et SALBER, D. (1996). The AMODEUS Project. Rapport technique. *Citation en page **79***

[Couturier *et al.*, 2012] COUTURIER, V., HUGET, M.-P. et TELISSON, D. (2012). Patterns for Agent-Based Information Systems : A Case Study in Transport. KALLONIATIS, C., éditeur : *Innovative Information Systems Modelling Techniques*, pages 49–72. InTech. *Citation en page **73***

[Cruz Torres *et al.*, 2011] CRUZ TORRES, M. H., VAN BEERS, T. et HOLVOET, T. (2011). (No) more design patterns for multi-agent systems. *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, VMIL'11*, SPLASH '11 Workshops, pages 213–220, New York, USA. ACM. *Citation en page **69***

[Dahmann et Morse, 1998] DAHMANN, J. S. et MORSE, K. L. (1998). High Level Architecture for simulation : an update. *Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings. 2nd International Workshop on*, pages 32–40. *Citation en page **19***

[Dalleau *et al.*, 2012] DALLEAU, M., CICCIONE, S., MORTIMER, J. A., GARNIER, J., BENHAMOU, S. et BOURJEA, J. (2012). Nesting phenology of marine turtles : insights from a re-

gional comparative analysis on green turtle (Chelonia mydas). *PloS one*, 7(10):e46920. *Citation en page **120***

[David, 2010] DAVID, D. (2010). *Prospective Territoriale par Simulation Orientee Agent*. Thèse de doctorat, Université de La Réunion. *6 citations en pages **3, 4, 15, 52, 54, et 60***

[David *et al.*, 2012] DAVID, D., GANGAT, Y., PAYET, D. et COURDIER, R. (2012). Reification of emergent urban areas in a land-use simulation model in Reunion Island. *ECAI workshop on Intelligent Agents in Urban Simulations and Smart Cities (IAUSSC2012)*, volume 2012, pages 28–32, Montpellier. *2 citations en pages **4 et 181***

[David *et al.*, 2007] DAVID, D., PAYET, D., BOTTA, A., LAJOIE, G., MANGLOU, S. et COURDIER, R. (2007). Un couplage de dynamiques comportementales : Le modèle DS pour l'aménagement du territoire. *Journées Francophones Systèmes Multi-Agents (JFSMA)*, pages 129–138. *2 citations en pages **3 et 51***

[David *et al.*, 2009] DAVID, D., PAYET, D., COURDIER, R. et GANGAT, Y. (2009). XELOC : un support générique pour la configuration et l ' initialisation de systèmes multi-agents. *Journées Francophones Systèmes Multi-Agents (JFSMA)*, pages 251–256, Lyon, France. Cepadues. *4 citations en pages **4, 48, 60, et 182***

[Davidsson, 2001] DAVIDSSON, P. (2001). Multi Agent Based Simulation : Beyond Social Simulation. *Proceedings of the Second International Workshop on Multi-Agent-Based Simulation-Revised and Additional Papers*, MABS '00, pages 97–107, London, UK. Springer-Verlag. *Citation en page **31***

[Davidsson et Johansson, 2005] DAVIDSSON, P. et JOHANSSON, S. J. (2005). On the metaphysics of agents. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1299–1300, New York, USA. ACM. *2 citations en pages **96 et 98***

[Devedžić, 1999] DEVEDŽIĆ, V. (1999). A survey of modern knowledge modeling techniques. *Expert Systems with Applications*, 17(4):275–294. *Citation en page **19***

[Drogoul *et al.*, 2003] DROGOUL, A., VANBERGUE, D. et MEURISSE, T. (2003). Simulation Orientée Agent : où sont les agents. *Actes des Journées de Rochebrune, Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels.* *2 citations en pages **9 et 13***

[Drogoul *et al.*, 2002] DROGOUL, A., VANBERGUE, D., MEURISSE, T., UNIVERSITÉ, L., PLACE, P. et CEDEX, J. P. (2002). Multi-Agent Based Simulation : Where are the Agents ?, Multi-Agent-Based Simulation. *II, Sichman J.S., Bousquet F., and Davidsson P. (Eds.), Proceedings of MABS 2002, Third International Worshop*, pages 89–104. Springer-Verlag. *2 citations en pages **9 et 13***

[Duncan, 2002] DUNCAN, J. (2002). Ring masters. *Cinefex*, 89:64–131. *Citation en page **2***

[Eldabi *et al.*, 2003] ELDABI, T., LEE, M. W. et PAUL, R. J. (2003). A Framework for Business Process Simulation : The Grab and Glue Approach. *SCS-European Simulation Symposium*. *Citation en page **28***

[Epstein, 2008] EPSTEIN, J. M. (2008). Why Model ? *Journal of Artificial Societies and Social Simulation*, 11(4):12. *Citation en page **26***

[Ferber, 1995] FERBER, J. (1995). *Les Systèmes Multi Agents : vers une intelligence collective*. *2 citations en pages **96 et 98***

[Ferber et Gutknecht, 1998] FERBER, J. et GUTKNECHT, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. *Multi Agent Systems*, pages 128–135. *Citation en page **36***

[Ferber *et al.*, 2003] FERBER, J., GUTKNECHT, O. et MICHEL, F. (2003). From agents to organizations : an organizational view of multi-agent systems. *Agent-Oriented Software Engineering*, pages 214–230. *Citation en page **36***

[Ferber et Stratulat, 2009] FERBER, J. et STRATULAT, T. (2009). Towards an integral approach of organizations in multi-agent systems : the MASQ approach. *Multi-agent Systems : Semantics and Dynamics of Organizational Models*, (March):1–23. *Citation en page **37***

[Filippi, 2003] FILIPPI, J.-B. (2003). *Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes*. Thèse de doctorat, Université de Corse. *Citation en page **21***

[Filippi et Bisgambiglia, 2004] FILIPPI, J.-B. et BISGAMBIGLIA, P. (2004). JDEVS : an implementation of a {DEVS} based formal framework for environmental modelling. *Environmental Modelling & Software*, 19(3):261–274. *Citation en page **21***

[Gaillard *et al.*, 2010] GAILLARD, F., KUBERA, Y., MATHIEU, P. et PICAULT, S. (2010). Une méthode pour naviguer dans l'espace des simulations. *Revue d'Intelligence Artificielle*, 24(5):575–599. *2 citations en pages **33 et 34***

[Galán *et al.*, 2009] GALÁN, J. M., IZQUIERDO, L. R., IZQUIERDO, S. S., SANTOS, J. I., DEL OLMO, R., LÓPEZ-PAREDES, A. et EDMONDS, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1):1. *3 citations en pages **9, 11, et 14***

[Gamma *et al.*, 1995] GAMMA, E., HELM, R., JOHNSON, R. E. et VLISSIDES, J. (1995). *Design Patterns*. Addison Wesley. *2 citations en pages **67 et 81***

[Gangat, 2009] GANGAT, Y. (2009). Simulation orientée agent pour l'aide à l'aménagement énergétique de l'île. Technical report, Université de La Réunion. *Citation en page **62***

[Gangat *et al.*, 2009a] GANGAT, Y., COURDIER, R., DAVID, D. et PAYET, D. (2009a). Modélisation et Simulation de l'aménagement énergitique d'un territoire. *Colloque sur la Convergence*

des Réseaux de l'Informatique et du Multimédia pour les Eservices CRIME'09, pages 17–22, Saint Denis, La Réunion. *3 citations en pages **3, 45, et 182***

[Gangat *et al.*, 2009b] GANGAT, Y., COURDIER, R. et PAYET, D. (2009b). Démonstration : Aménagement énergétique d'un territoire - une approche par simulation multi-agents. *Journées Francophones Systèmes Multi-Agents (JFSMA)*, pages 237–240, Lyon, France. Cepadues. *3 citations en pages **3, 45, et 182***

[Gangat *et al.*, 2010] GANGAT, Y., DALLEAU, M., DAVID, D., SEBASTIEN, N. et PAYET, D. (2010). Turtles are the turtles. *European Simulation and Modelling Conference (ESM)*, pages 439–442, Hasselt, Belgium. Eurosis. *3 citations en pages **3, 120, et 182***

[Gangat *et al.*, 2012a] GANGAT, Y., PAYET, D. et COURDIER, R. (2012a). Another step toward reusability in agent-based simulation : Multi-behaviors & aMVC. *24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, pages 1112–1119, Athens, Greece. IEEE Computer Society. *2 citations en pages **84 et 181***

[Gangat *et al.*, 2012b] GANGAT, Y., PAYET, D. et COURDIER, R. (2012b). Methodology for a New Agent Architecture Based on the MVC Pattern. *Lecture Notes in Computer Science - Artificial Intelligence : Methodology, Systems, and Applications*, 755:230–239. *2 citations en pages **102 et 181***

[Gangat *et al.*, 2013] GANGAT, Y., PAYET, D. et COURDIER, R. (2013). Approche méthodologique pour la modélisation multicomportementale dans la SOA. *Workshop de Modélisation Agents pour les Systèmes Complexes - Plate-Forme IA (MASyCo-PFIA)*, Lille, France. A Paraître. *2 citations en pages **119 et 182***

[Garro *et al.*, 2013] GARRO, A., PARISI, F. et RUSSO, W. (2013). A Process Based on the Model-Driven Architecture to Enable the Definition of Platform-Independent Simulation Models. PINA, N., KACPRZYK, J. et FILIPE, J., éditeurs : *Simulation and Modeling Methodologies, Technologies and Applications SE - 8*, volume 197 de *Advances in Intelligent Systems and Computing*, pages 113–129. Springer-Verlag. *Citation en page **184***

[Gass, 1984] GASS, S. I. (1984). Documenting a computer-based model. *Interfaces*, pages 84–93. *Citation en page **28***

[Gaudou *et al.*, 2009] GAUDOU, B., HO, T. V. et MARILLEAU, N. (2009). Introduce Collaboration in Methodologies of Modeling and Simulation of Complex Systems. *Intelligent Networking and Collaborative Systems, 2009. INCOS '09. International Conference on*, pages 1–8. *Citation en page **19***

[Gaudou *et al.*, 2011] GAUDOU, B., MARILLEAU, N. et HO, T. (2011). Toward a Methodology of Collaborative Modeling and Simulation of Complex Systems. CABALLÉ, S., XHAFA, F. et ABRAHAM, A., éditeurs : *Intelligent Networking, Collaborative Systems and Applications*, volume 329 de *Studies in Computational Intelligence*, pages 27–53. Springer-Verlag. *Citation en page **19***

[Gül et Maher, 2006] GÜL, L. F. et MAHER, M. L. (2006). Studying design collaboration in DesignWorld : an augmented 3D virtual world. *Proceedings of the 3rd International Conference on Computer Graphics, Imaging and Visualisation. IEEE.* *Citation en page **16***

[Gutknecht *et al.*, 2001] GUTKNECHT, O., FERBER, J. et MICHEL, F. (2001). Integrating tools and infrastructures for generic multi-agent systems. *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 441–448, New York, USA. ACM. *3 citations en pages **36, 37, et 153***

[Gutknecht *et al.*, 2000] GUTKNECHT, O., MICHEL, F. et FERBER, J. (2000). MadKit : une architecture de plate-forme multi-agent générique. Rapport technique 33, Montpellier. *Citation en page **36***

[Hayden *et al.*, 1999] HAYDEN, S., CARRICK, C. et YANG, Q. (1999). Architectural design patterns for multiagent Coordination. *Proceedings of the International Conference on Agent Systems.* *Citation en page **73***

[Heinrich *et al.*, 2012] HEINRICH, M., LEHMANN, F., SPRINGER, T. et GAEDKE, M. (2012). Exploiting single-user web applications for shared editing : a generic transformation approach. *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 1057–1066, New York, NY, USA. ACM. *Citation en page **184***

[Hussey et Carrington, 1997] HUSSEY, A. et CARRINGTON, D. (1997). Comparing the MVC and PAC architectures : a formal perspective. *Software Engineering IEEE Proceedings*, volume 144, pages 224–236. *Citation en page **80***

[Jennings *et al.*, 1998] JENNINGS, N. R., SYCARA, K. et WOOLDRIDGE, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 306:275–306. *4 citations en pages **1, 96, 97, et 98***

[Juziuk *et al.*, 2012] JUZIUK, J., WEYNS, D. et HOLVOET, T. (2012). Design Patterns for Multi-Agent Systems : A Systematic Literature Review. pages 1–21. *2 citations en pages **70 et 71***

[Kamphuis *et al.*, 2004] KAMPHUIS, R., CARLSSON, P., WARMER, C., KESTER, J. et KOK, K. (2004). Distributed intelligence for supply/demand matching to improve embedding of distributed renewable energy sources. *CRISP : Distributed Intelligence in Critical Infrastructures for Sustainable Power*, pages 115–123, Grenoble. *Citation en page **47***

[Klügl et Karlsson, 2009] KLÜGL, F. et KARLSSON, L. (2009). Towards Pattern-Oriented Design of Agent-Based Simulation Models. BRAUBACH, L., van der HOEK, W., PETTA, P. et POKAHR, A., éditeurs : *Multiagent System Technologies*, volume 5774 de *Lecture Notes in Computer Science*, pages 41–53. Springer Berlin / Heidelberg. *2 citations en pages **69 et 71***

[Krasner et Pope, 1988] KRASNER, G. E. et POPE, S. T. (1988). A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal Of Object Oriented Programming*, 1(3):26–49. *Citation en page **74***

[Kubera, 2010] KUBERA, Y. (2010). *Simulations orientées-interactions des systèmes complexes*. Thèse de doctorat, Université Lille 1. *3 citations en pages **33, 36, et 150***

[Lajoie, 2007] LAJOIE, G. (2007). Mémoire d'habilitation à diriger des recherches : "Recherches en modélisation urbaine". *Citation en page **50***

[Lajoie et Hagen-Zanker, 2007] LAJOIE, G. et HAGEN-ZANKER, A. (2007). La simulation de l'étalement urbain à La Réunion : apport de l'automate cellulaire Metronamica® pour la prospective territoriale. *Cybergeo*. *Citation en page **50***

[Law *et al.*, 2007] LAW, A. M., KELTON, W. D. et OTHERS (2007). *Simulation modeling and analysis*, volume 3. McGraw-Hill, 4th édition. *Citation en page **31***

[Lee et Chen, 2005] LEE, M. W. et CHEN, S. Y. (2005). A software reuse approach for developing Grab-and-Glue models. *WSEAS International Conference on SIMULATION, MODELING AND OPTIMIZATION*, volume 2005, pages 166–171. *Citation en page **28***

[Luck *et al.*, 2003] LUCK, M., MCBURNEY, P. et PREIST, C. (2003). *Agent Technology : Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink. *Citation en page **98***

[Macal, 2005] MACAL, C. M. (2005). Model Verification and Validation. *Workshop on "Threat anticipation : Social science methods and models"*, pages 52–58. *Citation en page **30***

[Maes, 1991] MAES, P. (1991). The agent network architecture (ANA). *ACM SIGART Bulletin*, 2(4):115–120. *Citation en page **66***

[Maher *et al.*, 2008] MAHER, M. L., ROSENMAN, M. et MERRICK, K. (2008). Agents for multidisciplinary design in virtual worlds. *Artificial Intelligence For Engineering Design Analysis And Manufacturing*, 21(3):267–277. *Citation en page **19***

[Maher *et al.*, 2006] MAHER, M. L., ROSENMAN, M., MERRICK, K., MACINDOE, O. et MARCHANT, D. (2006). DesignWorld : an augmented 3D virtual world for multidisciplinary, collaborative design. *Citation en page **19***

[Mahmoud et Maamar, 2006] MAHMOUD, Q. H. et MAAMAR, Z. (2006). Applying the MVC Design Pattern to Multi-Agent Systems. *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, pages 2420–2423. *Citation en page **72***

[Mansour, 2007] MANSOUR, S. (2007). *Un modèle de gestion distribuée de groupes ouverts et dynamiques d'agents mobiles*. Thèse de doctorat, Université de Pau et des Pays de l'Adour. *Citation en page **37***

[Marcenac et Giroux, 1998] MARCENAC, P. et GIROUX, S. (1998). GEAMAS : A Generic Architecture for Agent-Oriented Simulations of Complex Processes. *Applied Intelligence*, 8(3):247–267. *Citation en page **3***

[Mathieu *et al.*, 2013] MATHIEU, P., PICAULT, S. et KUBERA, Y. (2013). IODA Project - Termites nest simulation : model. *2 citations en pages **148 et 149***

[Mathieu *et al.*, 2001] MATHIEU, P., ROUTIER, J.-C. et URRO, P. (2001). Un modèle de simulation agent basé sur les interactions. *Actes des Journees Francophones sur les Modeles Formels de l'Interaction (MFI)*, pages 407–417, Toulouse. *Citation en page **33***

[Michel, 2002] MICHEL, F. (2002). Introduction to TurtleKit : A platform for building Logo Based MAS with MadKit. Rapport technique. *2 citations en pages **151 et 152***

[Michel, 2004] MICHEL, F. (2004). *Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents*. Thèse de doctorat, Université Monpellier II. *Citation en page **36***

[Michel, 2007] MICHEL, F. (2007). The IRM4S model : the influence/reaction principle for multiagent based simulation. *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, volume 5 de *AAMAS '07*, pages 133 :1—-133 :3, New York, USA. ACM. *Citation en page **86***

[Mikulecky, 1999] MIKULECKY, D. (1999). Définition of complexity. Rapport technique, Medical College of Virginia Commonwealth University. *Citation en page **1***

[Minsky, 1965] MINSKY, M. L. (1965). Matter, Mind and Models. *International Federation of Information Processing Congress*, volume I, pages 45–49. Spartan Books. *Citation en page **9***

[Newell, 2001a] NEWELL (2001a). Reply to the Respondents to " A Theory of Interdisciplinary Studies ". *Issues in Integrative Studies*, 148(19):137–148. *Citation en page **17***

[Newell, 2001b] NEWELL, W. (2001b). A theory of interdisciplinary studies. *Issues in Integrative Studies*, 25(19):1–25. *Citation en page **17***

[Nguyen *et al.*, 2009] NGUYEN, T. K., GAUDOU, B., HO, T. V. et MARILLEAU, N. (2009). Application of PAMS Collaboration Platform to Simulation-Based Researches in Soil Science : The Case of the MIcro-ORganism Project. *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pages 1–8. *2 citations en pages **19 et 21***

[Nguyen *et al.*, 2008] NGUYEN, T. K., MARILLEAU, N. et HO, T. V. (2008). PAMS – A New Collaborative Framework for Agent-Based Simulation of Complex Systems. BUI, T., HO, T. et HA, Q., éditeurs : *Intelligent Agents and Multi-Agent Systems*, volume 5357 de *Lecture Notes in Computer Science*, pages 287–294. Springer-Verlag. *2 citations en pages **19 et 72***

[Nguyen *et al.*, 2011] NGUYEN, T. K., MARILLEAU, N., HO, T. V., FALLAH, A. et EL FALLAH, A. (2011). New protocol supporting collaborative simulation. *Proceedings of the Second Symposium on Information and Communication Technology*, pages 137–145. *Citation en page **19***

[Nikolic *et al.*, 2007] NIKOLIC, I., BEERS, P. J. et DIJKEMA, G. P. J. (2007). Facilitating Interdisciplinary Modelling of Complex Problems. *Annual Hawaii International Conference on System Sciences*, page 197b. *2 citations en pages **18 et 23***

[North et Macal, 2011] NORTH, M. J. et MACAL, C. M. (2011). Product Design Patterns for Agent-Based Modeling. *2011 Winter Simulation Conference*, numéro 1996, pages 3087–3098, Phoenix, USA. *Citation en page **81***

[Nutaro et Hammonds, 2004] NUTARO, J. et HAMMONDS, P. (2004). Combining the model/-view/control design pattern with the DEVS formalism to achieve rigor and reusability in distributed simulation. *The Journal of Defense Modeling and Simulation : Applications, Methodology, Technology*, 1(1):19–28. *Citation en page **72***

[Nwana, 1996] NWANA, H. (1996). Software agents : An overview. *Knowledge Engineering Review*, pages 1–49. *Citation en page **96***

[Ocio et Brugos, 2009] OCIO, S. et BRUGOS, J. (2009). Multi-agent Systems and Sandbox Games. *Artificial Intelligence and the Simulation of Behaviors (AISB) : AI and Games Symposium*, Edinburgh, Scotland. SSAISB. *Citation en page **2***

[Oreskes *et al.*, 1994] ORESKES, N., SHRADER-FRECEHTTE, K. et BELITZ, K. (1994). Verification, validation, and confirmation of numerical models in the earth sciences. *Science*, 263(5147): 641–646. *Citation en page **32***

[Ozawa *et al.*, 2000] OZAWA, M., CUTKOSKY, M. R. et HOWLEY, B. J. (2000). Model Sharing among Agents in a Concurrent Product Development Team. *Proceedings of the IFIP TC5 WG5.2 Third Workshop on Knowledge Intensive CAD*, pages 143–166, Deventer, The Netherlands, The Netherlands. Kluwer, B.V. *Citation en page **19***

[Paul, 2002] PAUL, R. J. (2002). Internet management issues. chapitre The Intern, pages 209–219. IGI Publishing, Hershey, PA, USA. *2 citations en pages **27 et 28***

[Paul et Taylor, 2002] PAUL, R. J. et TAYLOR, S. J. E. (2002). What use is model reuse : is there a crook at the end of the rainbow ? *Winter Simulation Conference WSC Proceedings*, 1:648–652. *Citation en page **28***

[Payet *et al.*, 2006] PAYET, D., COURDIER, R., SEBASTIEN, N. et RALAMBONDRAINY, T. (2006). Environment as support for simplification, reuse and integration of processes in spatial MAS. *2006 IEEE International Conference on Information Reuse Integration*, pages 127–131. *3 citations en pages **3, 44, et 46***

[Phillips, 1999] PHILLIPS, W. (1999). Architectures for synchronous groupware. Rapport technique. *Citation en page **79***

[Pidd, 2002] PIDD, M. (2002). Reusing simulation components : simulation software and model reuse : a polemic. *Winter Simulation Conference WSC Proceedings*, WSC '02, pages 772–775. Winter Simulation Conference. *Citation en page **24***

[Pidd, 2009] PIDD, M. (2009). *Tools for thinking : modelling in management science*, volume 1987. John Wiley and Sons Ltd, Chichester, U.K. *Citation en page **29***

[Pokahr et Braubach, 2009] POKAHR, A. et BRAUBACH, L. (2009). A Survey of Agent-oriented Development Tools. EL FALLAH SEGHROUCHNI, A., DIX, J., DASTANI, M. et BORDINI, R. H., éditeurs : *Multi-Agent Programming :*, pages 289–329. Springer US. *Citation en page **183***

[Potel, 1996] POTEL, M. (1996). MVP : Model-View-Presenter the taligent programming model for c++ and java. *Taligent Inc*, (C). *Citation en page **76***

[Ralambondrainy, 2009] RALAMBONDRAINY, T. (2009). *Observation de simulations multi-agents à grande échelle*. Thèse de doctorat, Université de la Réunion. *5 citations en pages **4, 9, 12, 31, et 32***

[Ralambondrainy *et al.*, 2006] RALAMBONDRAINY, T., COURDIER, R. et PAYET, D. (2006). An ontology for observation of multiagent based simulation. *Proceedings of the 2006 IEEE International Conference on Web Intelligence and Intelligent Agent Technology*, pages 351–364. IEEE Computer Society. *Citation en page **4***

[Ramat, 2006] RAMAT, E. (2006). Introduction à la modélisation et à la simulation à événements discrets. *Modélisation et simulation multiagents : applications aux Sciences de l'Homme et de la Société*, pages 49–74. *Citation en page **9***

[Reenskaug, 2003] REENSKAUG, T. M. H. (2003). The Model-View-Controller (MVC) Its Past and Present. *Oslo Draft*, (Mvc):1–16. *Citation en page **74***

[Reese, 1987] REESE, R. (1987). Software reuse and simulation. *Winter Simulation Conference WSC Proceedings*. *Citation en page **40***

[Resnick, 1997] RESNICK, M. (1997). *Turtles, termites, and traffic jams : explorations in massively parallel microworlds*. Complex adaptive systems. MIT Press. *Citation en page **142***

[Robinson, 1997] ROBINSON, S. (1997). Simulation model verification and validation : increasing the users' confidence. *Winter Simulation Conference WSC Proceedings*, WSC '97, pages 53–59, Washington, DC, USA. IEEE Computer Society. *2 citations en pages **28 et 29***

[Robinson, 2004] ROBINSON, S. (2004). Simulation : the practice of model development and use. *Citation en page **26***

[Robinson, 2006] ROBINSON, S. (2006). Conceptual modeling for simulation : issues and research requirements. *Winter Simulation Conference WSC Proceedings*, numéro 1994 de WSC '06, pages 792–800. Winter Simulation Conference. *Citation en page **9***

[Robinson *et al.*, 2004] ROBINSON, S., NANCE, R., PAUL, R. J. et PIDD, M. (2004). Simulation model reuse : definitions, benefits and obstacles. *Simulation Modelling Practice and Theory*, 12(7-8):479–494. *4 citations en pages **9, 19, 26, et 40***

[Rosenman et Gero, 1998] ROSENMAN, M. et GERO, J. (1998). CAD modelling in multidisciplinary design domains. SMITH, I., éditeur : *Artificial Intelligence in Structural Engineering*, volume 1454 de *Lecture Notes in Computer Science*, pages 335–347. Springer-Verlag. *Citation en page **16***

[Rosenman et Gero, 1996] ROSENMAN, M. A. et GERO, J. S. (1996). Modelling multiple views of design objects in a collaborative cad environment. *Computer-Aided Design*, 28(3):193–205. *Citation en page **19***

[Rosenman *et al.*, 2007] ROSENMAN, M. A., SMITH, G., MAHER, M. L., DING, L. et MARCHANT, D. (2007). Multidisciplinary collaborative design in virtual environments. *Automation in Construction*, 16(1):37–44. *2 citations en pages **17 et 19***

[Russell et Norvig, 2003] RUSSELL, S. et NORVIG, P. (2003). *Artificial Intelligence : A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd, 2009 édition. *2 citations en pages **98 et 130***

[Sargent, 1986] SARGENT, R. G. (1986). Joining existing simulation programs. *Winter Simulation Conference WSC Proceedings*, numéro 1 de WSC '86, pages 512–516, New York, NY, USA. ACM. *2 citations en pages **9 et 28***

[Sargent, 1996] SARGENT, R. G. (1996). Verifying and validating simulation models. *Winter Simulation Conference WSC Proceedings*, WSC '96, pages 55–64, Washington, DC, USA. IEEE Computer Society. *Citation en page **28***

[Sargent, 1999] SARGENT, R. G. (1999). Validation and verification of simulation models. *Winter Simulation Conference WSC Proceedings*, WSC '04, pages 17–28. *2 citations en pages **9 et 31***

[Sauvage, 2003] SAUVAGE, S. (2003). *Conception de systèmes multi-agents : un thésaurus de motifs orientés agent*. Thèse de doctorat, Université de Caen - Basse Normandie. *2 citations en pages **70 et 71***

[Sauvage, 2004a] SAUVAGE, S. (2004a). Agent oriented design patterns : A case study. *Proceedings of the Third International Joint Conference*. *Citation en page **70***

[Sauvage, 2004b] SAUVAGE, S. (2004b). Design Patterns for Multiagent Systems Design. MONROY, R., ARROYO-FIGUEROA, G., SUCAR, L. et SOSSA, H., éditeurs : *Mexican International Conference on Artificial Intelligence : Advances in Artificial Intelligence*, volume 2972 de *Lecture Notes in Computer Science*, pages 352–361. Springer Berlin Heidelberg. *Citation en page **70***

[Schmidt *et al.*, 1996] SCHMIDT, D. C., FAYAD, M. et JOHNSON, R. E. (1996). Software Patterns. *Communications of the ACM*, 39(10):37–39. *Citation en page **2***

[Sébastien, 2009] SÉBASTIEN, N. (2009). *Distribution et Parallélisation de Simulations Orientées Agent*. Thèse de doctorat, Université de La Réunion. *2 citations en pages **4 et 12***

[Sébastien *et al.*, 2008] SÉBASTIEN, N., COURDIER, R., HOAREAU, D. et HUGET, M.-P. (2008). Analysis of temporal dependencies of perceptions and influences for the distributed execution of agent-oriented simulations. *European Simulation and Modelling Conference (ESM)*, pages 343–347, Le Havre, France. *Citation en page **4***

[Siebers *et al.*, 2007] SIEBERS, P., AICKELIN, U., CELIA, H. et CLEGG, C. (2007). Using intelligent agents to understand management practices and retail productivity. *Proceedings of the 39th conference on Winter simulation : 40 years ! The best is yet to come*, pages 2212–2220. IEEE Press Piscataway, NJ, USA. *Citation en page **38***

[Siebert, 2011] SIEBERT, J. (2011). *Approche multi-agent pour la multi-modélisation et le couplage de simulations.* Thèse de doctorat, Université Henri Poincaré - Nancy. *Citation en page **38***

[Sierhuis et Selvin, 1996] SIERHUIS, M. et SELVIN, A. M. (1996). Towards a Framework for Collaborative Modeling and Simulation. *Workshop on Strategies for Collaborative Modeling & Simulation CSCW*, volume 9. *Citation en page **19***

[Sinha *et al.*, 2001] SINHA, R., PAREDIS, C. J. J., LIANG, V.-C. et KHOSLA, P. K. (2001). Modeling and simulation methods for design of engineering systems. *J. Comput. Info. Sci. Eng.*, 1(1):84–91. *2 citations en pages **18 et 23***

[Smith, 1987] SMITH, R. (1987). Panel on design methodology. *SIGPLAN Not.*, 23(5):91–95. *Citation en page **67***

[Soulié, 2001] SOULIÉ, J. (2001). Vers une approche multi-environnements pour les agents. *Citation en page **85***

[Soulié *et al.*, 1998] SOULIÉ, J., MARCENAC, P., CALDERONI, S. et COURDIER, R. (1998). GEAMAS V2.0 : an object oriented platform for complex systems simulations. *Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*, pages 230–242. *Citation en page **3***

[Sriram *et al.*, 1992] SRIRAM, D., LOGCHER, R. D., GROLEAU, N. et CHERNEFF, J. (1992). DICE : An Object Oriented Programming Environment for Cooperative Engineering Design. TONG, C. et SRIRAM, D., éditeurs : *Artificial intelligence in Engineering Design*, chapitre DICE : An O, pages 303–366. Academic Press Professional, Inc., San Diego, CA, USA. *Citation en page **19***

[Stelting et Maassen, 2002] STELTING, S. et MAASSEN, O. (2002). *Applied Java Patterns.* Prentice Hall PTR. *Citation en page **81***

[Sullivan et Knight, 1996] SULLIVAN, K. J. et KNIGHT, J. C. (1996). Experience assessing an architectural approach to large-scale systematic reuse. *Proceedings of the 18th international conference on Software engineering*, ICSE '96, pages 220–229, Washington, DC, USA. IEEE Computer Society. *Citation en page **40***

[Szostak, 2002] SZOSTAK, R. (2002). How to Do Interdisciplinarity : Integrating the Debate. *Issues in Integrative Studies*, 122(20):103–122. *Citation en page **17***

[Tarpin-Bernard et David, 1999] TARPIN-BERNARD, F. et DAVID, B. (1999). AMF : un modèle d'architecture multi-agents multi-facettes. *Techniques et Sciences Informatiques*, 5. *Citation en page **80***

[Tesnière, 1965] TESNIÈRE, L. (1965). *Eléments de syntaxe structurale*. Klincksieck, Paris. *Citation en page **127***

[Tranier, 2007] TRANIER, J. (2007). *Vers une vision intégrale des systèmes multi-agents : Contribution à l'intégration des concepts d'agent, d'environnement, d'organisation et d'institution*. Thèse de doctorat, Université Monpellier II. *Citation en page **37***

[Tyler, 2011] TYLER, J. (2011). *App Inventor for Android : Build Your Own Apps - No Experience Required !* Wiley Publishing, 1st édition. *Citation en page **184***

[Weiss, 1999] WEISS, G. (1999). *Multiagent Systems : A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, London, England. *Citation en page **98***

[Wilensky, 1999] WILENSKY, U. (1999). NetLogo. *Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.* *Citation en page **151***

[Wong et Sriram, 1993] WONG, A. et SRIRAM, D. (1993). SHARED : An information model for cooperative product development. *Research in Engineering Design*, 5(1):21–39. *2 citations en pages **16 et 19***

[Woosnam-Savage, 2011] WOOSNAM-SAVAGE, R. C. (2011). The Matériel of Middle-Earth. *Picturing Tolkien : Essays on Peter Jackson's The Lord of the Rings Film Trilogy*, pages 139–167. *Citation en page **2***

[Zannier *et al.*, 2004] ZANNIER, C., ERDOGMUS, H. et LINDSTROM, L. (2004). *Extreme Programming and Agile Methods - XP/Agile Universe 2004 : 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004, Proceedings*. Lecture Notes in Computer Science. Springer-Verlag. *Citation en page **157***

Acronymes et abréviations

aMVC
: **a**gent **MVC**.

DOM
: **M**odélisation **O**rientée **D**ynamique – de l'anglais **D**ynamic **O**riented **M**odeling.

DS
: **D**OMINO **S**MAT : **D**émarche **O**bjet **M**ultisite pour l'étude des **I**nteractions entre **N**iveaux d'**O**rganisation - **S**ystème **M**ulti-**A**gents **T**erritoires.

EDMMAS
: **E**nergy **D**emand **M**anagement by **M**ulti**A**gent **S**imulation.

GEAMAS
: **GE**neric **A**rchitecture for **M**ulti**A**gent **S**imulations.

GEAMAS-NG
: **GEAMAS N**ew-**G**eneration.

LIM
: **L**aboratoire d'**I**nformatique et de **M**athématiques — EA2525.

MMC
: **M**odélisation **M**ulti**C**omportementale.

MVC
: **M**odèle **V**ue **C**ontrôleur.

POO
: **P**rogrammation **O**rientée **O**bjet.

SMA
: **S**ystème **M**ulti-**A**gents.

SMART
: **S**ystème **M**ulti-**A**gents et **R**éseaux de **T**élécommunication.

SMAT
: **S**ystème **M**ulti-**A**gents **T**erritoires.

SOA
: **S**imulation **O**rientée **A**gent.

XELOC
: e**X**tensible **E**diting **L**anguage **Of** **C**onfiguration.

Table des figures

Liste des tableaux

Listes des Définitions