Modélisation et Simulation des systèmes complexes spatialisés. Utilisation de Systèmes Multi-Agents et Multi-composant pour la gestion des pêcheries.

Paul-Henri Martelloni

UNIVERSITE DE CORSE-PASCAL PAOLI
ECOLE DOCTORALE ENVIRONNEMENT ET SOCIETE
UMR CNRS 6134 (SPE)

Thèse présentée pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE
Mention : Modélisation et simulation

Soutenue publiquement par
Paul-Henri
MARTELLONI

le 13 DECEMBRE 2021

Modélisation et Simulation des systèmes complexes spatialisés. Utilisation de Systèmes Multi-Agents et Multi-composant pour la gestion des pêcheries.

Directeurs :
M Paul-Antoine BISGAMBIGLIA, Pr, Université de Corse
M Eric INNOCENTI, Dr, Université de Corse

Rapporteurs :
M Gregory ZACHAREWICZ, Pr, École nationale supérieure des mines d'Alès
M Jean-Christophe SOULIE, DR (HDR), CIRAD

Jury :
M David HILL, Pr, Université Clermont Auvergne
M Gregory ZACHAREWICZ, Pr, École nationale supérieure des mines d'Alès
M Jean-Christophe SOULIE, DR (HDR), CIRAD
M Paul-Antoine BISGAMBIGLIA, Dr-HDR, Université de Corse
M Paul-Antoine BISGAMBIGLIA, Pr, Université de Corse
M Eric INNOCENTI, Dr, Université de Corse

RÉSUMÉ

L'équipe de recherche informatique de notre université travaille en modélisation et en simulation afin d'étudier de manière holistique les systèmes naturels complexes. Dans ce travail, nous nous sommes fixés comme objectif d'utiliser la simulation informatique comme éprouvette virtuelle afin de concevoir un outil d'aide à la décision dans un contexte pluridisciplinaire. L'enjeu central de notre travail est de proposer un modèle exécutable suffisamment générique pour permettre l'aide à la décision et de l'utiliser dans le cas de la modélisation d'une pêcherie. Nous proposons d'appliquer nos travaux à la gestion des stocks de poissons en Corse. Ceci nous a conduits à suivre la démarche suivante : la première étape est la représentation du système complexe. Pour cela nous avons cherché un paradigme pour représenter des composants à structure hiérarchique capables d'interactions et d'autonomie de décision. Le paradigme agent se prête très bien à la représentation de systèmes complexes bio-économiques. Ensuite, la question de la robustesse de l'approche s'est posée. Pour cela nous nous sommes rapprochés de la théorie de la modélisation et de la simulation (TM&S) proposée par B.P. Zeigler. Dans notre cas, le formalisme PDEVS peut être considéré comme un formalisme unificateur et pivot. En effet, celui-ci permet d'inclure d'autres concepts, on parle de multi-modélisation. Nous le complétons notamment avec le paradigme agent afin de bénéficier que son expressivité et de sa faculté à décrire simplement des entités (les agents), leurs interactions mutuelles, ainsi l'environnement dans lequel elles évoluent. Enfin, pour respecter un cadre formel, indispensable au travail préalable de conceptualisation, nous utilisons la formalisation DPDEMAS proposée R. Franceschini et déjà développé dans notre équipe. En effet, cette formalisation offre dans sa forme originale une analogie intéressante entre un Système Multi-Agents (SMA) et les modèles PDEVS

et DSDE (version dynamique de PDEVS). Nous y retrouvons une description de l'agent à partir d'un corps qui modélise sa représentation physique dans un environnement, ainsi qu'à partir d'un esprit qui est le lieu des prises de décision.

Ceci nous a amené à nous pencher sur la problématique de la prise de décision. L'enjeu central était donc de proposer un modèle exécutable suffisamment générique pour permettre l'aide à la décision et de l'utiliser dans le cas de la modélisation d'une pêcherie. Pour cela, nous proposons une approche mettant à disposition des agents des briques inspirées de Soar. Une brique d'optimisation permettant d'affiner les décisions des agents. Une brique d'apprentissage par renforcement permettant aux agents de compléter leurs connaissances dans une situation donnée. Nous complétons cette description par un premier exemple didactique avant d'appliquer notre approche à deux exemples de modélisation d'un système de pêcherie basé sur 50 ans de données. Le premier exemple est un modèle à 5 composants pour simuler des scénarios avec quotas. Le second exemple est un modèle à 6 composants pour simuler lui aussi des scénarios avec quotas, avec en plus la prise en compte de l'espace et des migrations entre zones. Néanmoins, ces deux modèles, n'ont pour objectif que de valider notre approche. En effet, si grâce à ces deux exemples, nous obtenons des résultats cohérents nous permettant de valider leur comportement, ces modèles sont purement théoriques et ne représentent pas suffisamment bien la réalité du système pour pouvoir être utilisés en l'état pour aider à la gestion des ressources halieutiques. Les perspectives que nous envisageons par la suite sont d'une part l'intégration de nouvelles méthodes d'apprentissage et de décision, de développer un modèle de pêcherie plus représentatif de la réalité ou encore d'utiliser notre approche pour traiter d'autres cas d'études.

Mots clefs : DEVS ; modélisation et simulation ; SMA ; aide à la décision ; multicomposant

REMERCIEMENTS

Je remercie en tout premiers lieux mes directeurs et co-directeurs de thèse le professeur Paul-Antoine Bisgambiglia et le docteur Eric Innocenti pour leur aide et leur encadrement durant ces années de doctorat. Il ne faudrait surtout pas oublier le parfait homonyme de mon directeur, Le docteur Paul-Antoine Bisgambiglia (HDR) dont j'ai partagé le bureau toutes ces années et qui a aussi joué un rôle prépondérant dans l'avancement de cette thèse. Merci à tout les trois pour m'avoir poussé quand je stagnais tout en me laissant une grande liberté dans mes recherches.

Merci au professeur Gregory Zacharewicz et le directeur de recherche Jean-Christophe Soulier pour me faire l'honneur d'être rapporteurs cette thèse.

Je remercie également le professeur David Hill pour avoir accepté de faire partie de mon jury de thèse.

Merci au docteure Evelyne Vittori pour m'avoir suivi au cours de cette aventure au sein de mon comité de pilotage et au docteur Gauthier Quesnel pour ses conseils techniques.

Je remercie également les enseignants chercheurs du département informatique de l'université avec qui j'ai pu débuter en tant qu'enseignant, pour leur accueil et leurs conseils. Ainsi que les personnes du 4ème étage, Laura, Stephanie, Laetitia, en particulier qui ont composé avec mes mauvaises habitudes en matière administrative.

Merci à ma chère Anaïs, qui a commencé sa thèse en même temps que moi et qui pour la paraphraser, a su se rapprocher et se faire une place très particulière à mes côtés. Elle aura également su me motiver quand cela s'est avéré nécessaire et de me changer les idées avec, entre autres, sa capacité hors du commun à faire planter un ordinateur d'un simple regard.

Merci à Bruno et Nicolas presque aussi sociables que moi, et tout aussi alambiqués. Il y a eu des grands moments où nous n'avons jamais oublié de rire. Des longues discussions rocambolesques et des heures à redécouvrir nos cours de math devant des tableaux entiers d'équations.

Merci au groupe de doctorant, ingénieurs ou stagiaires et qui pour certain ont été les trois avec qui j'ai partagé de nombreuses pauses, café, belote, etc. Je citerais notamment le prochainement docteure Karina et ses expériences de brûlages épiques où certain se sont consacrés corps et âme, David ou encore Guillaume.

Merci à mes parents pour m'avoir supporté et épaulé surtout durant ces années. À mes amis aussi en particulier Cecile qui ne voulait surtout pas manquer cet évènement mais qui à son grand regret, ne va finalement peut être pas pouvoir se déplacer. Sans oublier Pasqua avec qui nous passons beaucoup moins de temps à geeker depuis le début de ce doctorat.

Merci à Christine et Christophe pour leurs relectures et leur aide pour la traduction anglaise de mon résumé scientifique.

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

TABLE DES MATIÈRES

Remerciements **5**

Introduction **11**

1 Approches de modélisation conceptuelle et simulation **19**

1.1 Modélisation des systèmes complexes 20

1.1.1 Du système complexe au modèle informatique 21

1.1.2 Un processus itératif en quatre phases 22

1.1.3 Modélisation multi-échelle 24

1.2 Simulation et simulateur 27

1.2.1 La simulation à évènements discrets 28

1.2.2 La simulation en temps discret 29

1.3 Les Systèmes Multi-Agents 30

1.3.1 Le paradigme agent 30

1.3.2 Architectures cognitives exécutables 36

1.4 Formalismes de spécification 38

1.4.1 Formalismes DEVS et PDEVS 39

1.4.2 Extensions du formalisme DEVS 42

1.4.3 Analogies DEVS/SMA 46

1.5 Apports conceptuels et analogie soar 50

1.5.1 Fonctionnement de Soar . . . 51
1.5.2 Analogie DEVS/SOAR . . . 54
1.5.3 Nos contraintes . . . 55
1.5.4 Processus d'apprentissage . . . 56
1.6 Conclusion . . . 60

2 Intégration informatique **61**
2.1 Frameworks DEVS/PDEVS . . . 62
2.2 Environnement : Multicomposant dans VLE . . . 65
2.2.1 Description informatique de VLE . . . 67
2.2.2 Intégration du multicomposant dans VLE . . . 69
2.2.3 Déroulement de la simulation . . . 71
2.2.4 Exemple de mise en œuvre . . . 73
2.3 Agent : approche cognitive dans VLE . . . 74
2.3.1 Schéma structurel . . . 75
2.3.2 Nos modules de décision . . . 77
2.3.3 Intégration de notre approche dans VLE . . . 83
2.3.4 Exemple didactique de simulation . . . 91
2.4 Conclusion . . . 98

3 Expérimentations **101**
3.1 Modélisation d'une pêcherie . . . 102
3.1.1 Modèle de Graham-Schaefer . . . 104
3.1.2 Calibration du modèle . . . 105
3.2 Simulation par agents non spatialisée . . . 108
3.2.1 Modèle informatique . . . 108
3.2.2 Modèle exécutable . . . 112
3.2.3 Résultats et discussions . . . 120
3.3 Simulation par agents spatialisée . . . 122
3.3.1 Modèle informatique . . . 123
3.3.2 Modèle exécutable . . . 126

3.3.3 Résultats et discussions 130
3.4 Conclusion 134

Conclusion et perspectives de recherche **137**

Bibliographie **141**

Table des figures **157**

Liste des tableaux **161**

Liste des algorithmes **163**

INTRODUCTION

La simulation informatique s'affirme chaque jour davantage dans un mouvement de décloisonnement des disciplines scientifiques et comme un outil de prise de décision. Ses briques de base, le modèle et le simulateur, nécessitent un travail exigeant à l'intersection de plusieurs domaines.

C'est particulièrement le cas en sciences environnementales où informaticiens, écologues, biologistes et économistes doivent collaborer pour concevoir leurs outils scientifiques et numériques.

Dans ce travail, nous traitons de la problématique de gestion des stocks de poissons en Corse. Nous fondons nos concepts et outils sur la théorie de la modélisation et de la simulation (TM&S) présentée dans [Zeigler, 1976].

L'institut français de recherche pour l'exploitation de la mer a publié un rapport [IFREMER, 2019] montrant que la surexploitation de la ressource halieutique touche environ un quart des stocks de poissons pêchés en France : "27% du volume capturé en France provient de surexploitation". Les océans étant la principale source de protéines de la planète, il est primordial de proposer des stratégies de gestion efficace des stocks. C'est l'un des objectifs de la simulation informatique des modèles bio-économiques de pêcherie.

La simulation d'un modèle bio-économique de pêcherie permet d'évaluer les conséquences environnementales, mais aussi économiques des mesures engagées dans une politique de pêche.

Dans le contexte corse, fournir un modèle bio-économique spatialisé de confiance pour la gestion des pêcheries s'avère être un enjeu majeur pour le territoire. La mise à disposition d'un tel modèle pourra fournir aux décideurs et acteurs politiques un outil de simulation efficace quant à l'évaluation des politiques de pêche. Celui-ci pourra notamment permettre d'estimer les périodes de pêche, d'évaluer des

quotas, de définir les dimensions des maillages des filets, ou encore de décider de la délimitation de zones protégées (ZPR). Bien que l'aspect artisanal des pêcheries et la complexité des courants présents en Corse autour de l'île imposent l'élaboration d'un modèle dédié, des besoins similaires peuvent être également observés dans d'autres régions où intérêts économiques et protection de l'environnement cohabitent [Bruskotter and Fulton, 2013, Gelcich and Donlan, 2015].

De ce fait, les propositions de modèles bio-économiques en pêcherie sont nombreuses dans la littérature [Schaefer, 1957, Béné et al., 2001, Clark, 2010]. Les modèles discrets semblent être de très bons candidats, la diversité des échelles de temps et les caractéristiques à prendre en compte (nouvelles mesures, catastrophes naturelles, périodes d'interdictions) rendent ces approches beaucoup moins coûteuses en calcul.

Contexte

La modélisation et la simulation d'une pêcherie invite les chercheurs en informatique, en sciences naturelles et en économie à s'unir afin de tisser un lien entre sciences fondamentales et sciences humaines. Il s'agit d'une condition sine qua non à la réussite d'un projet interdisciplinaire œuvrant à la compréhension des systèmes complexes naturels.

Comme nous le verrons, la science de la simulation informatique commande pour cela l'usage de paradigmes, de formalismes, de concepts et de techniques en accord avec l'évolution des matériels. Le contexte pluridisciplinaire des sciences environnementales et plus spécifiquement celui des pêcheries soutenant ce travail de recherche, commande un processus de modélisation clair et précis facilitant un dialogue efficace entre les différents scientifiques intervenant dans le projet.

Les personnels de l'équipe d'accueil de l'*UMR CNRS 6134 SPE*[1] sont rompus à ce travail pluridisciplinaire depuis maintenant de nombreuses années, comme en témoignent notamment leurs travaux de recherche dans les projets suivants :

— Energies renouvelables ([Mattei et al., 2006, Voyant et al., 2017]).

— Feux de forêts ([Balbi et al., 2009, Bisgambiglia et al., 2017]).

— Ressources naturelles ([Nothias et al., 2020]).

— Gestion et valorisation des Eaux en Méditerranée ([Koeck et al., 2015]).

1. Sciences Pour l'environnement.

À l'*UMR CNRS 6240 LISA* [2] de l'université de Corse, des travaux de recherche reposant sur le paradigme agent ont été engagés en 2016 dans le but d'expérimenter par la simulation des politiques de pêche, et notamment d'identifier et de quantifier les facteurs environnementaux et économiques d'intérêt dans une pêcherie [Innocenti et al., 2016]. Dans ce contexte, des échanges entre experts des différents domaines ont eu lieu à travers le programme PO-FEDER [3] *MoonFish*.

Problématiques et objectifs

Nous souhaitons dans nos travaux replacer la simulation au centre d'une approche interdisciplinaire qui vise à mieux comprendre les fonctionnements des petits métiers de la pêche en Corse.

Notre objectif est d'utiliser la simulation informatique comme vecteur d'intégration des préceptes et concepts liés aux disciplines intervenant dans le processus de modélisation d'une pêcherie, afin de tester des scénarios et d'effectuer des expériences virtuelles. Le but est d'améliorer la gestion des ressources halieutiques en Corse.

Pour répondre à notre problématique et facilement mettre en place des expérimentations virtuelles, nous souhaitons utiliser le paradigme Agent.

Il s'agit de l'un des domaines de compétence de nos équipes [Urbani, 2006, Mattei et al., 2012, Innocenti et al., 2016, Franceschini et al., 2017].

Il s'avère que le paradigme agent se prête très bien à la modélisation des systèmes complexes bio-économiques, notamment du fait de sa capacité à pouvoir facilement exprimer dans un modèle des sous-composants hiérarchiquement organisés et capables d'interaction, mais également pour son caractère interdisciplinaire. Le paradigme agent offre l'opportunité de processus de modélisation adaptés aux échanges entre experts de différents domaines (en l'occurrence ici, économie, biologie et informatique). Il permet en outre une correspondance naturelle entre le système complexe observé et les objets conceptuels de l'étude lors du processus de modélisation.

Afin de garantir la robustesse de l'approche proposée, ce travail est à contextualiser dans le cadre de la théorie de la modélisation et de la simulation proposée par *B.P. Zeigler* dans [Zeigler, 1976]. Dans son ouvrage, aujourd'hui complété par une troisième édition [Zeigler et al., 2018], les bases de l'utilisation de la simulation comme science sont formellement énoncées.

2. Laboratoire Lieux, Identités, eSpaces et Activités
3. PO-FEDER : Programme opérationnel Fonds européen de développement régional

Dans cette optique plusieurs problématiques sont abordées comme la représentation spatiale d'un modèle, l'articulation et la composition de modèles hétérogènes, ainsi que l'intégration dans des modèles conventionnels de processus de décision.

Notre approche nécessitant l'intégration de plusieurs concepts, elle est donc basée sur le formalisme DEVS dans sa version dite "paralèlle" ou PDEVS [Kim et al., 2009] qui a été défini comme un multi-formalisme par [Vangheluwe et al., 2002].

Dans notre cas, le formalisme PDEVS peut être considéré comme un formalisme unificateur et pivot. En effet, celui-ci permet d'inclure d'autres concepts, nous le complétons notamment avec le paradigme agent afin de bénéficier de son expressivité et de sa faculté à décrire simplement des entités (les agents), leurs interactions mutuelles, ainsi qu'avec l'environnement, au sein d'un Système Multi-Agents (SMA).

À titre d'exemple, un agent pêcheur peut prélever une quantité de poissons dans la mer (interaction agent-environnement) ; ou encore, un agent gestionnaire peut imposer des mesures de restriction de pêche (interaction agent-agent).

Afin de respecter un cadre formel, limitant les ambiguïtés de modélisation, permettant l'usage d'un langage unique entre experts et modélisateurs et donc indispensable, nous compléterons la formalisation DPDEMAS proposée dans [Franceschini et al., 2017]. Celle-ci offre dans sa forme originale une analogie intéressante entre un Système Multi-Agents (SMA) et les modèles PDEVS et DSDE (version dynamique de PDEVS [Barros, 1998]). L'agent y est décrit à partir d'un corps qui modélise sa représentation physique dans un environnement, ainsi qu'à partir d'un esprit qui est le lieu des prises de décision.

L'apport central de notre travail sera de compléter la notion d'esprit des agents DPDEMAS. Il s'agira d'intégrer la notion d'apprentissage caractéristique liée à la fois à l'environnement, via les interactions, ainsi qu'aux expériences de l'agent dans un but d'autonomie. Pour cela, nous proposons de nous baser sur l'architecture cognitive *SOAR pour State, Operator And Result en anglais* [Laird, 2012] et de l'intégrer dans la *description d'un agent cognitif.*

Nous souhaitons proposer un modèle computationnel suffisamment générique pour permettre l'aide à la décision, via l'usage de codes informatiques de simulation modulaires et évolutifs. Dans la pratique, il s'agit de produire un laboratoire virtuel d'expérimentation, pour que le décideur puisse à partir de paramètres judicieusement choisis, produire des données de simulation, en fonction de scénarii bio-économiques probables.

Le sujet central de ce travail est la *modélisation et la simulation du système dynamique complexe de*

la pêcherie, c'est-à-dire une étendue d'eau naturelle ou artificielle (l'environnement) qu'exploitent des pêcheurs (agents) (cf. figure 1).

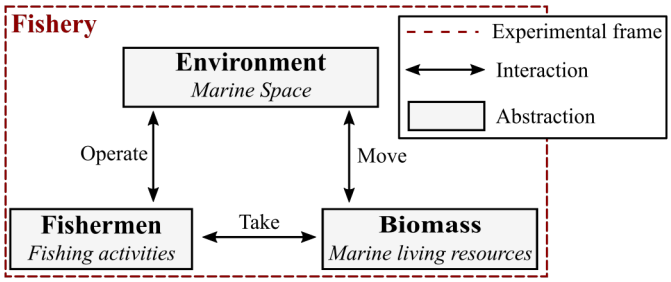


FIGURE 1 – Représentation du système complexe inspiré de [Idda et al., 2020].

A notre connaissance, le système complexe d'une pêcherie fut modélisé pour la première fois en *science halieutique*, ainsi que plusieurs fois en *science économique*. Malheureusement, les modélisations proposées furent trop souvent cantonnées au domaine scientifique des seuls modélisateurs intervenant dans le processus de modélisation. Nous verrons qu'un modèle de pêcherie est d'abord un *modèle bio-économique* et qu'à ce titre, il est nécessaire de prendre en compte aussi bien les aspects économiques, que les aspects biologiques des espèces considérées dans le système. La difficulté principale rencontrée dans ce processus de modélisation résulte en général des méthodes qui sont utilisées par les modélisateurs. Ceux-ci utilisent des méthodes, pour la plupart fondées sur des mathématiques qui décrivent des équilibres stables et homogènes, simplifiant à l'extrême le domaine qu'ils ne maîtrisent pas. Ainsi, les chercheurs en biologie auront tendance à proposer des modèles de population, mono ou multi-espèces, en simplifiant fortement l'écriture des équilibres liés à l'aspect économique ; et à contrario, les chercheurs en économie auront tendance à considérer des modèles économiques à l'équilibre, composés d'entités homogènes, ce qui ne correspond jamais à la réalité.

Dans les faits, le système complexe d'une pêcherie est empreint d'une forte hétérogénéité dans ses composants ; de plus, ceux-ci sont répartis spatialement et selon différents niveaux d'abstraction ; qui plus est, le contexte de l'étude ne peut que rarement être celui d'un équilibre parfait. C'est dans ce contexte de pluridisciplinarité, très cloisonné en général, que nous proposons d'apporter notre contribution. Il s'agit pour nous d'utiliser les préceptes, concepts et formalismes de la science informatique, notamment en définissant et en proposant :

— un *processus de modélisation itératif* adapté au caractère pluridisciplinaire de ces travaux de re-

cherche. Celui-ci doit permettre de consolider les méthodes de modélisation rencontrées en sciences halieutiques et économiques. Il s'agit ici d'adapter le processus de modélisation conceptuelle dans le contexte de la modélisation des pêcheries, via l'usage d'abstractions facilitant l'échange et le dialogue interdisciplinaire. Nous définissons pour cela le *modèle conceptuel* (modélisation conceptuelle), comme un préalable indispensable à la phase de conception des composants informatiques de modélisation. Les formalismes de spécification issus de la théorie de la modélisation et de la simulation seront d'un grand secours dans cette optique.

— un *modèle conceptuel multicomposant* et *multi-échelle* générique basé sur l'usage d'*agents informatiques*[4]. La *méthode d'intégration* des composants informatiques proposée est déduite de la modélisation conceptuelle que nous posons comme préalable. L'usage de composants permet l'obtention dans les codes de simulation informatiques d'entités modulaires et génériques plus faciles à (ré)utiliser, et à faire évoluer, là où l'usage courant commande la définition de simples objets issus de la POO[5]. Enfin, l'usage des concepts agents, nous permet de rompre définitivement avec les approches cloisonnées courantes que l'on rencontre dans la littérature, nous permettant ainsi de modéliser des systèmes hors d'équilibre, et de tenir compte de l'hétérogénéité des principaux acteurs qu'il faut modéliser.

— L'*utilisation d'agents cognitifs* pour la simulation informatique des pêcheries. Parmi les concepts retenus pour modéliser le système complexe d'une pêcherie, les méthodes reposant sur le *paradigme agent* nous semblent adéquates [Ferber and Perrot, 1995]. En effet, comme en atteste la littérature, en écologie et en biologie ([Bousquet and Le Page, 2004], [Bousquet et al., 1994], [Coquillard and Hill, 1997], en sciences sociales [Deffuant et al., 2002], en éthologie ([Corbara et al., 1993]), les *Systèmes Multi-Agents* (*SMA*) font largement consensus quand il s'agit de modéliser des ensembles hétérogènes d'entités capables d'interactions dans un environnement. Comme nous le détaillerons dans un chapitre ultérieur, ce type d'approche permet d'appréhender efficacement les difficultés liées à la prise en compte des nombreux éléments à considérer sur différentes échelles, et cela toujours dans un contexte d'interdisciplinarité.

Le modèle informatique résultant sera implémenté sous la forme d'un framework (modèle exécutable)

4. La précision apportée ici, indique que la référence qui en est faite n'est pas celle ayant court en économie, mais en informatique.

5. POO : programmation orientée objet, nous invitons également le lecteur à prendre connaissance de la programmation orientée composant (POC).

suffisamment générique pour permettre différents scénarios de simulation. Nous détaillerons ses possibilités dans une partie expérimentale.

Organisation de ce mémoire

Dans le chapitre (1), nous présentons les préceptes, concepts et formalismes qui structurent ce travail de recherche. Nous rappelons les principales définitions issues de la théorie de la modélisation et de la simulation informatique, à savoir la notion de système complexe, de modèle, et de simulateur. Nous profitons également de l'écriture de ce chapitre pour présenter les formalismes de spécifications ayant influencé la formulation des modèles conceptuels que nous avons développés, ainsi que les principaux formalismes issus de la formalisation DEVS employés par les membres de l'équipe informatique de l'UMR SPE. Ce chapitre sera également l'occasion de définir ce qu'est un *Système Multi-Agent* (*SMA*), les différents types d'agents, ainsi que les associations qu'il est possible de faire entre les différents formalismes que nous avons utilisés (*PDEVS*, *DPDEMAS*, etc.).

Dans le chapitre (2), conformément au processus de modélisation que nous proposons, nous traitons de l'intégration du *modèle informatique*. Nous détaillons les structures informatiques mise en oeuvre, ainsi que les frameworks testés lors de nos expériences de simulation. Nous expliquons notamment comment il convient d'intégrer notre architecture cognitive dans un environnement de multi-modélisation et de simulation basé sur le formalisme DEVS .

Dans le chapitre (3), nous présenterons deux exemples de mise en oeuvre d'un système complexe de la pêcherie, sujet central d'application de ce travail. Nous détaillons les composants du modèle computationnel et nous présentons différents résultats de simulation faisant suite à diverses implémentations de scénarios.

Enfin, nous concluons ce travail et nous présentons des perspectives futures de recherche dans une cinquième et dernière partie.

Chapitre 1

APPROCHES DE MODÉLISATION CONCEPTUELLE ET SIMULATION

Nous présentons dans ce chapitre les concepts qui structurent ce travail et notamment ceux qui sont utilisés pour définir le modèle conceptuel du système dynamique complexe de la pêcherie.

Nous rappelons dans un premier temps les principales définitions de la théorie de la modélisation et de la simulation informatique (TM&S), à savoir la notion de système complexe et le processus de modélisation et de simulation.

Dans un second temps, nous présentons les formalismes de spécifications ayant influencé la formulation de notre modèle conceptuel. Une attention particulière est portée aux formalismes issus de la TM&S, comme la famille des spécifications formelles DEVS employée par les membres de notre équipe.

Ce chapitre sera également l'occasion de définir plus avant ce qu'est un *Système Multi-Agents* (*SMA*), une *approche cognitive* comme *State, Operator And Result SOAR*[1], ainsi que les liens existant entre le formalisme *PDEVS* et le paradigme agent au travers de la formalisation *DPDEMAS*.

Enfin, nous proposerons un modèle conceptuel basé sur *DPDEMAS* qui intègre l'*approche cognitive SOAR* et constituant l'un des principaux apports de cette thèse.

1.1 Modélisation des systèmes complexes

La simulation informatique implique d'identifier l'objet de l'étude à mettre au centre de la démarche de modélisation, c'est-à-dire l'ensemble des constituants du système complexe cible à l'origine du phénomène observé. L'appréhension par la pensée de ces constituants et de leur dynamique par la simple observation n'est pas chose aisée, aussi la mise en œuvre d'un processus de modélisation est indispensable. L'objectif de la première phase de ce processus est de proposer une représentation simplifiée, suffisamment formelle et contextualisée pour une étude en simulation du système complexe : le modèle conceptuel. Ce modèle conceptuel sera traduit sous la forme d'un modèle informatique, puis sous la forme d'un modèle exécutable dans le but d'étudier et de vérifier des hypothèses lors d'expériences de simulation.

Le modèle exécutable final est donc notre éprouvette ou notre boîte de Pétri qui va nous permettre de multiplier les expérimentations virtuelles pour vérifier et valider notre modèle conceptuel initial. Le lecteur intéressé pourra trouver les définitions usuelles de la vérification et la validation de modèles (conceptuels) écologiques dans le chapitre 7 du livre [Coquillard and Hill, 1997].

Nous allons maintenant revenir sur la notion de système.

1. État, Opérateur et Résultat en français

1.1.1 Du système complexe au modèle informatique

L'objet central d'une étude en simulation informatique est le *système complexe*, c'est-à-dire un *ensemble d'éléments en interaction*. Cet ensemble génère un *comportement* observable globalement qui ne peut pas être expliqué facilement depuis les seules propriétés individuelles de ses constituants.

Les systèmes complexes, qu'ils soient des systèmes écologiques, empiriques, physiques, ou environnementaux, présentent comme point commun d'être constitués d'un grand nombre de constituants hétérogènes et hiérarchiquement organisés.

En science environnementale, l'étude de ces systèmes passe par un travail pluridisciplinaire de modélisation que l'on matérialise dans le *modèle conceptuel*. Le modélisateur doit identifier les constituants et leur organisation.

Le modèle conceptuel permet de représenter le *système complexe* que l'on étudie, sous la forme d'une *abstraction* répondant au contexte de l'étude. Le lecteur intéressé par une discussion plus approfondie sur ce sujet pourra notamment consulter les travaux de [Corum, 2014, Pidd, 2010]. Cette *abstraction* dépend de la méthode formelle que l'on utilise pour décrire le *système complexe*. La plupart du temps, il s'agit de décrire des entités que l'on répartit hiérarchiquement à travers le prisme de paradigmes liés au contexte de l'étude (approches top-down ou bottom-up). Chacune de ces entités est formulée en fonction de caractéristiques que l'on aura judicieusement choisies, et selon le point de vue le plus pertinent possible pour le projet de simulation.

Dans la littérature, il existe de nombreux travaux qui insistent sur le fait que la première abstraction obtenue doit être suffisamment *"simple"* quant à la formulation conceptuelle des constituants.

Nous tâcherons dans ce travail de nous en tenir à cet état de fait, d'autant plus que cela fait maintenant de nombreuses années que ce débat existe dans la communauté scientifique, notamment à cause de son caractère critique [Chwif et al., 2013, Balci, 2011, Mitroff et al., 1974].

Dans [Robinson, 2015], l'auteur fait remarquer que la modélisation est encore aujourd'hui davantage un *"art, plutôt qu'une science exacte"*[2].

Dans ce contexte, grâce aux paradigmes et formalismes de la théorie de la modélisation et de la simulation informatique (TM&S), nous espérons apporter davantage de rigueur et de méthode quant à la formulation des codes de simulation. En effet, la TM&S tend à devenir une science pluridisciplinaire

2. *"conceptual modeling is more an art than a science."*

à part entière, elle offre pour cela pléthore de *concepts* et *formalismes* qui permettent de faire face à la complexité des systèmes naturels.

Nous espérons par ce travail améliorer la tâche du modélisateur, en intégrant clairement notre approche de modélisation dans un processus itératif, suffisamment simple pour être appliqué au domaine des pêcheries, sans qu'il soit nécessaire pour autant de trop simplifier la réalité, et tenir compte du caractère pluridisciplinaire du sujet de l'étude.

1.1.2 Un processus itératif en quatre phases

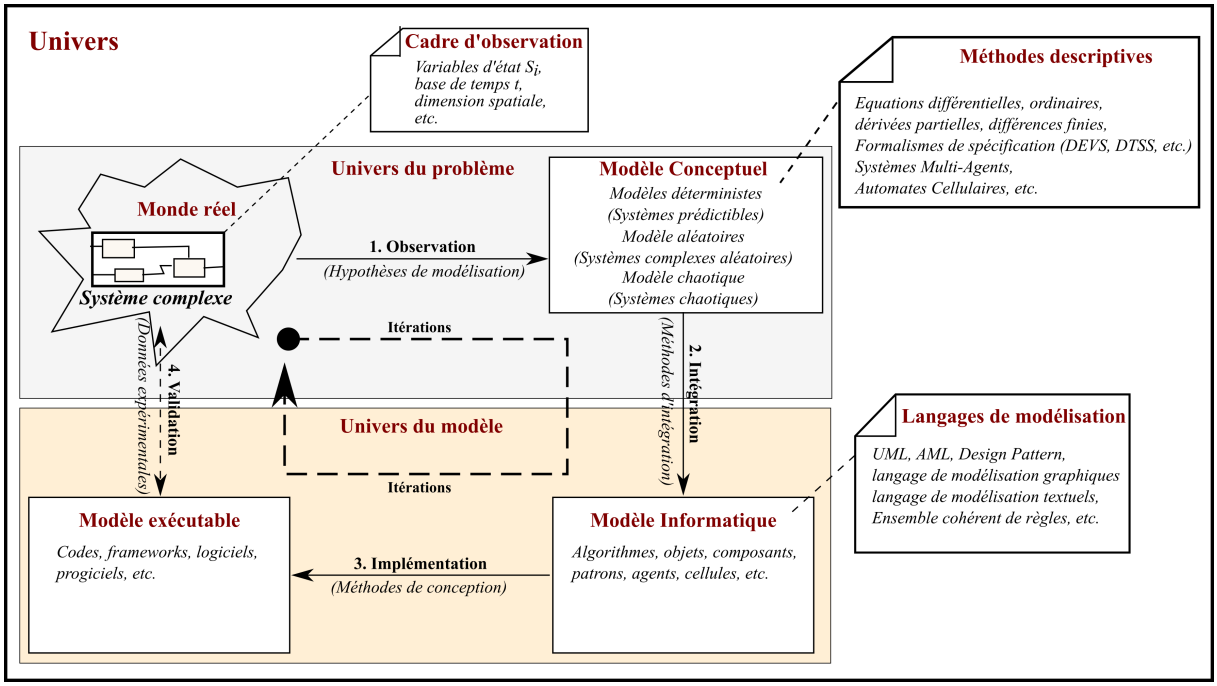


FIGURE 1.1 – Le processus de modélisation itératif en quatre étapes que nous proposons de suivre. Ce processus est en partie issus des travaux précurseurs de [Siegfried, 2014, Chwif et al., 2013, Robinson, 2013].

Le modèle conceptuel résulte d'un processus descriptif de modélisation *itératif*, *hiérarchique* et *multi-vues*.

On trouve dans la littérature de nombreux travaux qui traitent de la formulation des modèles conceptuels. Historiquement, on remarque que la *science mathématique* fut la première à initier la construction de ce type d'abstraction, en fournissant les équations différentielles comme premiers instruments de

conceptualisation.

Les *équations différentielles* s'inscrivent également dans un processus itératif *de modélisation* ayant pour finalité la mise au point de méthodes informatiques permettant de décrire des ensembles de transformations séquentielles (génériques si possibles) menant à la formulation de modèles exécutables.

Toutes ces méthodes œuvrent à déplacer le modèle conceptuel formulé mathématiquement sous la forme d'équations différentielles, dans des structures informatiques (type abstrait de données) sous la forme de modèles exécutables. Il s'agit pour l'informaticien de proposer les structures et instructions adéquates permettant de quitter l'univers continu du réel, que nous appelons *univers du problème*, pour l'univers discret de l'ordinateur que nous appelons *univers du modèle*.

Ainsi, dans ce travail, nous proposons de produire à partir du modèle conceptuel, le *modèle informatique* et son pendant, le *modèle exécutable*, c'est-à-dire le fichier exécutable exploitable par l'ordinateur.

Bien que de nombreuses étapes soit proposées dans la littérature, cherchant un compromis entre détail et simplicité, nous soutenons l'usage d'un processus de modélisation en quatre phases :

1. **Phase d'observation** : définition du modèle conceptuel. Le modèle conceptuel issu de la *phase d'observation* décrit dans un *langage de modélisation* adéquat les éléments-clés constitutifs [3] du système complexe ainsi que leurs interactions. Dans l'*univers du modèle*, cette description intègre les différentes *hiérarchies d'abstraction*, ainsi que les différents *points de vue*. D'ailleurs, dans la littérature, on qualifie souvent le modèle conceptuel de *modèle hiérarchique et multivue*. Quelle que soit la terminologie employée, on constate que le modèle conceptuel exprime dans un langage de modélisation, standardisé ou non, les constituants et leurs relations (interactions) à travers le prisme d'un *cadre d'observation expérimental* du système complexe. Le modèle conceptuel est devenu depuis maintenant quelques années, l'objet de nombreux travaux qui tentent d'aboutir à des standards [Cervenka and Trencansky, 2007].

2. **Phase d'intégration** : après avoir défini le modèle conceptuel, la *phase d'intégration* vise à formuler le *modèle informatique*. Celui-ci traite de l'intégration des contraintes liées à l'usage des outils numériques. Ainsi, ce type de modèle est proche de la nature matérielle des ordinateurs. C'est dans l'*univers du modèle* que le modélisateur (souvent un informaticien) aura la charge de formuler le *modèle informatique*. Celui-ci prend la forme d'une abstraction traduisant la *pensée algorithmique*

3. Plus avant, nous emploierons le terme de composants.

en entités logiques et fonctionnelles, organisées conformément aux règles structurelles et comportementales qui sont décrites dans le modèle conceptuel.

3. **Phase d'implémentation** : à la fin du processus de modélisation, le *modèle informatique* est matérialisé sous la forme d'un framework ou *modèle exécutable* dans un langage de programmation (objet la plupart du temps).
4. **Phase de validation** : chacune des itérations du processus de modélisation a pour but d'améliorer le modèle conceptuel ainsi que le modèle informatique. L'ajout d'itération dans le processus augmente sensiblement la qualité des données obtenues grâce aux simulations [Corum, 2014]. In fine, cela facilite la *modularité*, l'*évolutivité* et la *reproductibilité* des codes du framework du modèle exécutable.

Le *processus de modélisation itératif* en 4 étape que nous avons présenté dans cette section est détaillé figure 1.1 . Il permet de consolider l'usage des méthodes de modélisation rencontrées en sciences halieutique, économique et informatique. Nous avons redéfini le processus de modélisation conceptuelle afin de le dédier à la modélisation des pêcheries, en insistant sur l'usage d'abstractions facilitant l'échange et le dialogue pluridisciplinaire. Notre objectif est maintenant, via l'usage de ce processus, de construire un modèle conceptuel hiérarchique multicomposant à base d'agents.

1.1.3 Modélisation multi-échelle

Notre processus de modélisation pose le questionnement des échelles descriptives et de l'organisation des constituants dans un système complexe, selon différents niveaux d'abstraction et selon différentes vues.

Concernant les niveaux d'abstraction, historiquement c'est l'approche réductionniste qui introduisit la première la possibilité d'une description étagée des constituants dans un processus de modélisation. On la retrouve dans les travaux fondateur datant de 1648 de *René Descartes* [Bertalanffy, 1969] qui posèrent les bases de la méthode de conceptualisation réductionniste.

Selon la méthode de conceptualisation réductionniste, les niveaux d'abstraction sont décrits du haut vers le bas (approche top-down) et le processus de modélisation doit toujours se faire dans le sens d'une diminution du nombre des constituants du système complexe. Pour cela, chacun des niveaux d'abstraction rassemble un ensemble de constituants minimalistes capables de décrire un aspect du système.

À la fin des années 1960 les travaux précurseurs du biologiste *L. von Bertalanffy* sur la dynamique des systèmes [Bertalanffy, 1969] et la systémique firent définitivement émerger le concept de l'interaction [4]. La *pensée systémique* fit alors son apparition en tant que nouveau paradigme de modélisation, où les niveaux d'abstraction furent formalisés du bas vers le haut (approche bottom-up), en insistant sur l'importance de comprendre les relations qui existent entre les différents constituants d'un système complexe. Citons également *H. A. Simon et K. E. Boulding* [Boulding, 1956] qui contribuèrent à la pensée systémique en définissant notamment les concepts de niveau d'organisation et d'arborescence.

Alors qu'initialement, le réductionnisme défendait la croyance qu'un système complexe ne peut être compris qu'en le réduisant en une somme de parties plus simples, plus faciles à comprendre et reposait sur une formulation analytique des composants du système, le paradigme systémique prit le contre-pied. La pensée systémique considère le système complexe comme une entité dynamique globale constituée de sous-systèmes reliés entre eux et interdépendants. Ainsi, le paradigme systémique remet en cause la plupart des présupposés du paradigme réductionniste, il introduit avec la notion de niveaux d'abstraction, la prise en compte des relations entre les constituants du système complexe ; constituants pouvant être à leur tour de nouveaux des sous-systèmes à étudier. La notion de système complexe apparaît ainsi clairement comme étant une entité globale intégrant en un tout cohérent, des sous-systèmes et leurs relations, depuis lesquels émerge un niveau global unitaire d'existence, tel un phénomène physique, biologique voire même un phénomène social. En tant qu'approche globalisante le paradigme systémique est à l'opposé des méthodes mathématiques analytiques au caractère réductionniste. De nos jours, les modèles conçus selon ce procédé sont qualifiés de modèles hiérarchiques [5] ou de modèles multi-échelles [6].

La modélisation multi-vue s'inscrit dans la continuité du paradigme systémique en faisant référence à la formulation de modèles distincts et séparés d'un même système complexe, dans le but de modéliser les différents aspects sémantiques du système. Comme le fait remarquer *M. C. Oussalah* dans ses travaux sur les systèmes numériques [Oussalah et al., 1989], lors de l'observation d'un système complexe, sur un même niveau d'abstraction, différentes vues distinctes peuvent être considérées. Chacune de ces vues représente un aspect particulier du système, et les modèles qui en résultent peuvent être à ce titre qualifiés de modèles multi-vues [7].

4. Dans son ouvrage Ludwig von Bertalanffy [Bertalanffy, 1969] définit les concepts fondateurs de la pensée systémique (systèmes ouverts, homéostasie, équifinalité, etc).
5. en anglais : hierarchical models
6. en anglais multi-level models
7. en anglais multi-view models.

Le paradigme multi-vues est important dans le cadre d'un projet de simulation interdisciplinaire comme celui présenté dans ce travail, car il complète la pensée systémique et l'approche réductionniste. Il s'agit pour le modélisateur d'intégrer les différents points de vue des parties prenantes lors du processus de modélisation, à travers la formulation de modèles conceptuels capables d'intégrer les différents apports descriptifs relatifs aux différentes disciplines qui sont impliquées dans le projet de simulation. D'ailleurs, de nombreux auteurs tels que [Simon, 1990, Von Hanxleden et al., 2012] soutiennent l'importance des vues dans le processus de modélisation. Ils font également remarquer à juste titre, que l'expression des vues ne doit pas se limiter à l'usage quasi-exclusif de formules mathématique, mais intégrer également l'usage de symboles, de mots, de schémas et de dessins, ceux-ci s'avérant dans de nombreux cas beaucoup plus adéquats.

En ce qui concerne le paradigme agents, nous noterons que dans [Parunak et al., 1998] la simulation de *SMA* est définie comme la représentation directe des comportements, des actions et des interactions d'un ensemble de constituants autonomes (les agents), qui évoluent dans un environnement commun. De plus, [Michel, 2007b], nous fait remarquer que dans ce contexte, la dynamique globale dans un *SMA* au niveau macroscopique, est considérée comme le fruit de la dynamique issue des interactions qui se déroulent au niveau microscopique.

À titre d'exemple, cela se traduit en simulation informatique, dans des modèles multicomposants, où les comportements globaux résultent de l'addition des actions d'un ensemble de composants, engendrant même parfois un comportement émergeant, cher à la maxime *"le tout étant plus que la somme des parties"*.

Ainsi, dans notre processus de modélisation, l'articulation du travail de formalisation entre ces deux approches de modélisation globales (top-down vs bottom-up) est donc fondamentale.

Toujours selon [Michel, 2007b], déduire la dynamique macroscopique d'un système en fonction de la dynamique du niveau microscopique soulève de nombreux problèmes, techniques et conceptuels, qui sont le plus souvent ignorés : les modèles agents se limitent le plus souvent aux spécifications du niveau micro (le comportement des agents et l'environnement) ; et très peu d'éléments concernent la manière dont le niveau macro en est effectivement déduit mais de notre point de vue c'est l'un des paradigmes les mieux armés pour travailler sue ces deux niveaux [David et al., 2002].

Ainsi, l'approche que nous mettons en œuvre doit permettre de décomposer de façon modulaire le *SMA*, cela dans un but de simplification de la construction du modèle conceptuel, et cela afin de permettre une meilleure réutilisation des composants du modèle informatique.

Ainsi, le processus de modélisation à base d'agents que nous mettons en œuvre permet d'approcher la complexité du système, en définissant dans une première étape, les constituants du niveau d'abstraction le plus fondamental : approche bottom-up. Nous réservons l'approche top-down à une deuxième étape, afin de valider l'organisation hiérarchique et les relations des constituants retenus pour formuler le modèle conceptuel.

Conscient qu'il est important de combiner plusieurs perspectives d'un même système au sein d'une seule représentation cohérente, pour cela, nous faisons le choix de recourir au formalisme *Multicomposant* (*MC*) proposé initialement par *B.P. Zeigler* [Zeigler et al., 2000]. En effet, le formalisme *MC* est suffisamment générique pour englober la spécification de nombreuses approches de conceptualisation hiérarchiques et multi-vues, pouvant mêler des démarches hybrides de modélisation qui font appel à la fois à l'approche réductionniste, ainsi qu'à la pensée systémique.

Le formalisme *MC* permet de formuler des modèles conceptuels hiérarchiques et multi-vues, afin d'aboutir à des modèles informatiques qu'il sera aisé de connecter aux algorithmes de simulation de la famille *DEVS*. Ces algorithmes de simulation sont détaillés dans [Zeigler et al., 2018] et sont implémentés dans de nombreux environnements de simulation, qualifiés de simulateurs abstraits, ils décrivent précisément le processus de simulation en charge de faire évoluer le modèle informatique sous-jacent. Nous décrivons le processus de simulation dans la section suivante.

1.2 Simulation et simulateur

Le *simulateur* ou *algorithme de la simulation* a pour objet la prise en compte de la dimension temporelle dans le *modèle informatique*. Il consiste à faire fonctionner le *modèle informatique* dans un *temps simulé*. Cette phase complémentaire au processus de modélisation appelée *simulation*, a pour rôle de définir la mécanique du fonctionnement du *modèle informatique*, en intégrant la dimension temporelle discrète. La science de la *simulation informatique* traite pour cela du développement de structures de contrôle adaptées, appelées *simulateurs abstraits*[8]. Les *simulateurs* servent à générer, à partir des *modèles exécutables* des données expérimentales de simulation. Le *système complexe*, le *modèle conceptuel/informatique*, le *simulateur*, sont ainsi liés par un *lien de modélisation* (*Système complexe - Modèle conceptuel/informatique*), par un *lien de simulation* (*Simulateur - Modèle informatique*), ainsi que par un lien

8. c'est-à-dire des algorithmes de simulation spécialisés, issues du modèle conceptuel considéré initialement et formulés dans le modèle informatique.

de validation - *Simulateur/Système complexe*) . Le processus complet de modélisation et de simulation d'un système complexe en simulation informatique est illustré sur la figure 1.2.

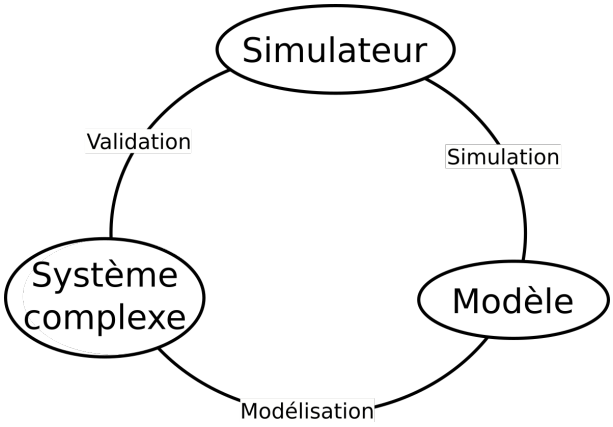


FIGURE 1.2 – Le processus de simulation d'un système complexe.

En nous basant sur le formalisme *MC*, nous disposons de fait du simulateur abstrait proposé par [Zeigler et al., 2000].

La mise en correspondance entre observations et connaissances acquises sur le système complexe permet ainsi de valider ou d'invalider les modèles intermédiaires, depuis le code du modèle exécutable jusqu'au modèle conceptuel. Ainsi, en procédant par validations successives, le modélisateur est en capacité d'améliorer le fonctionnement du modèle exécutable qui est finalement produit. Il existe cependant autant de démarches de modélisation conceptuelles que de simulateurs différents, le lecteur intéressé trouvera des descriptions des simulateurs les plus courants dans les ouvrages [Coquillard and Hill, 1997, Erard and Déguénon, 1996, Jerry, 1984].

Le caractère discret des composants internes des micro-ordinateurs favorise l'usage des techniques de la *simulation à évènements discrets*. Celles-ci mettent en œuvre des modèles informatiques qui décrivent le comportement de changement d'état des systèmes complexes sur la base d'*évènements*.

1.2.1 La simulation à évènements discrets

La mise en œuvre de la *simulation à évènements discrets* nécessite de faire progresser le temps dans les modèles conceptuels par dates d'occurrences d'évènements, c'est-à-dire où seuls les instants significatifs doivent être pris en compte. Ainsi, durant le processus de simulation, seuls les *évènements* qui engendrent

des changements d'état dans le modèle doivent être considérés. L'algorithme du simulateur correspondant nécessite l'usage d'un *échéancier*, dont le rôle est d'ordonner chronologiquement les évènements à traiter. Chaque progression du temps implique un traitement et tous les évènements sont donc productifs. Dans ce type de simulateur, le temps progresse plus ou moins rapidement en fonction des évènements qu'il faut traiter. Le traitement d'un évènement à la tête de l'échéancier peut engendrer l'ajout et/ou la suppression d'autres évènements. L'algorithme d'un simulateur à évènements discrets est illustré sur la figure 1.3.

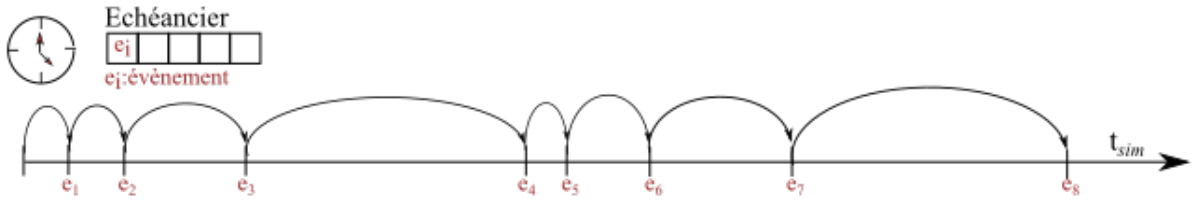


FIGURE 1.3 – Principe de la simulation à évènements discrets.

La simulation à évènements discrets engendre deux catégories d'évènements : les *évènements externes* et les *évènements internes*. Les occurrences des *évènements externes* ne sont en général pas contrôlées par le modèle lui-même. Les occurrences d'*évènements internes*, en revanche, résultent d'évènements préalablement programmés par le modèle. En général, lors d'une *simulation à évènements discrets*, la gestion des différents types d'évènements s'effectue distinctement, à l'aide de deux *fonctions de transition d'état* : une *fonction de transition d'état externe* δ_{ext}, et une *fonction de transition d'état interne* δ_{int}.

On définit également une *fonction d'avancement du temps* $ta(s)$ afin de déterminer la période durant laquelle aucun évènement interne ne se produit. C'est cette démarche de conceptualisation dédiée à la modélisation des *systèmes dynamiques complexes* qui est décrite formellement par le formalisme *DEVS* et ses dérivés.

1.2.2 La simulation en temps discret

La formalisation des modèles conceptuels suppose dans le cas d'une *simulation à temps discret*, une évolution pas à pas des états du modèle durant le processus de simulation. Il s'agit en fait d'un cas particulier de la simulation à évènements discrets, qui implique de faire évoluer le temps du modèle par *pas de temps* h fixes et constants. L'axe des temps est découpés en intervalles de taille identique, et seule une horloge cadence les transitions d'état δ des différents composants du modèle. À chaque pas de temps,

l'algorithme du simulateur exécute les évènements qui sont programmés entre la date courante t et le prochain top d'horloge. Le principe de la *simulation en temps discret* est illustré sur la figure 1.4.

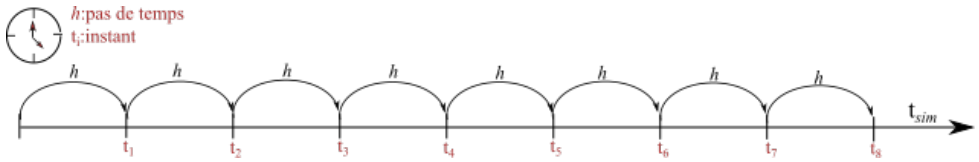


FIGURE 1.4 – Principe de la simulation en temps discret.

L'état suivant S_{t+1} du modèle est calculé à partir de l'état courant S_t du modèle et de l'influence de son environnement. À un instant t_i, le modèle est dans un état S_i et une *règle de transition* définit la façon dont le changement d'état se produira à l'instant suivant $t_{i+1} = t_i + h$. Le simulateur modifie alors l'état du modèle entre ces deux instants d'observation du système, et fait progresser l'horloge d'une unité de temps h. Le temps évolue ainsi de façon discrète, où chaque pas de temps est un multiple d'une période de référence (1 seconde, 1 minute, 1 année, etc.). La spécification des *modèles à temps discret* s'accompagne habituellement de la définition de deux fonctions : une fonction de transition d'état δ, et une fonction de sortie λ. Si l'état du système au temps t_i est $S(t_i)$ et que la valeur des données en entrée au temps t_i vaut $x(t_i)$, alors l'état du système au temps $t_i + h$ est donnée par $S(t_i + h) = \delta(S(t_i), x(t_i))$. La valeur de sortie du modèle vaut alors $y(t_i) = \lambda(S(t_i), x(t_i))$ (*type Mealy*) ou $y(t_i) = \lambda(S(t_i))$ (*type Moore*) [Zeigler et al., 2000].

Après avoir défini le cadre conceptuel général de ce travail, nous présentons dans les sections qui suivent les paradigmes et les formalismes de modélisation qui fondent les travaux présentés dans ce mémoire. Nous débutons cet état de l'art par les *Systèmes Multi-Agents* (*SMA*).

1.3 Les Systèmes Multi-Agents

1.3.1 Le paradigme agent

Après des années dominées par les approches basées sur des équations décrivant le comportement du système étudié surtout d'un point de vue endogène, la simulation des systèmes complexes a connu une profonde évolution avec l'apparition des *Systèmes Multi-Agents* (*SMA*). Les approches basées sur le

paradigme agents[9] permettent de formaliser l'intégration de comportements exogènes.

Selon *J.Ferber* [Ferber, 1995a] le *paradigme agent* repose sur la description d'un système complexe depuis la formulation d'*interactions* entre *agents*, ou *groupes d'agents* et un *environnement*. Nous pouvons l'illustrer à partir de la représentation schématique de la figure 1.5 où les interactions sont limitées aux actions de perceptions et d'actions de l'agent.

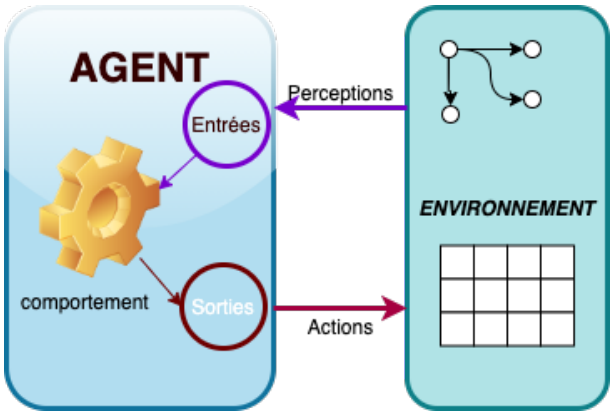


FIGURE 1.5 – Représentation schématique d'un SMA.

Ainsi, l'expression d'un système complexe sous la forme d'un *SMA* implique de formuler un modèle conceptuel dont la dynamique repose sur la spécification de comportements individuels simples et de leurs relations. Les comportements sont formulés dans les agents, c'est-à-dire sous la forme d'entités capables d'interactions avec elles-mêmes à un niveau local.

En science du vivant, on qualifie ce type de modèle conceptuel, où chaque agent possède ses propres caractéristiques et son propre comportement, de *modèle individu centré* (IBM[10]) [Grimm, 1999, Judson, 1994]. En simulation informatique, il s'agit de *modèles à base d'agents* (ABM[11]). Ils sont utilisés depuis plus de trente ans avec succès pour conceptualiser, modéliser et simuler les systèmes dynamiques complexes que l'on rencontre en écologie, en géographie, en économie, en sociologie, en archéologie ou encore en psychologie [Phan, 2014].

Le *paradigme agent* est particulièrement bien adapté à la modélisation des systèmes complexes. Néanmoins, bien qu'il soit utilisé dans de nombreux domaines applicatifs, aussi bien en recherche que dans l'industrie, il peut exister des différences notables quant aux concepts qui lui sont associés. Nous

9. Agent-Based Modelling and Simulation (ABMS) en anglais.
10. *individual-based model* en anglais.
11. *Agent-Based Model* en anglais.

allons dans les sous-sections qui suivent détailler plus avant les principaux concepts du paradigme agent.

Les agents

Dès qu'il s'agit de définir un *SMA*, le concept *d'agent* est bien entendu présent. Les agents représentent au sein d'un *SMA* des entités à part entière qui possèdent une autonomie, ainsi qu'une influence sur l'ensemble du *SMA* de part leurs actions nombreuses et variées. Bien que le terme d'agent soit souvent mentionné dans la littérature, et parfois même de manière assez vague, nous nous référons dans ce document à la définition précise de *J. Ferber* extraite de [Ferber, 1995a], où les agents sont définis comme des entités physiques ou virtuelles possédant :

— une capacité d'agir dans l'environnement ;

— une capacité de communiquer directement avec d'autres agents ;

— une capacité de pouvoir être mû par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'il cherche à optimiser) ;

— une capacité de posséder des ressources propres ;

— une capacité de percevoir (mais de manière limitée) son environnement ;

— une capacité de disposer d'une représentation partielle de son environnement (et éventuellement aucune) ;

— possédant des compétences et offrant des services ;

— une capacité de pouvoir éventuellement se reproduire ;

— une capacité de se comporter dans le but de satisfaire des objectifs, en tenant compte des ressources et des compétences à disposition, et en fonction de la perception, des représentations et communications perçues.

Ces principes ont été notamment développés dans les travaux de [Parunak et al., 1998], où l'accent est mis sur les actions et les interactions d'un ensemble d'entités autonomes (les agents) qui évoluent dans un environnement commun. *F. Michel* dans [Michel, 2007a] souligne le caractère d'émergence des *SMA*, où la dynamique globale qui s'observe au niveau macroscopique résulte des interactions qui se déroulent au niveau microscopique.

De ces travaux sur les agents, il résulte deux principes essentiels qui sont utiles à notre travail :

— le principe de l'*action - interaction*. Dans un *ABM*, un agent ne dispose pas nécessairement de toutes les connaissances requises, ou ne possède pas toutes les capacités nécessaires pour atteindre son/ses objectif(s). Grâce au principe l'*action - interaction*, un agent peut quand même agir en fonction de son/ses objectif(s) à travers sa *zone de perception*, les interactions fournissant une partie des informations nécessaires au calcul de son comportement.

— le principe de l'*action - réaction*. On exprime par ce principe qu'un événement peut avoir une influence sur l'agent ou sur ses actions. En retour, cela engendre sous certaines conditions une modification de son comportement en retour.

Dans ses travaux sur les *ABM*, *M. Wooldridge* [Wooldridge, 2002] considère plusieurs familles d'agents :

— les *agents logiques*, dont le comportement est basé sur des déductions logiques ;

— les *agents réactifs*, dont le fonctionnement est basé sur une simple correspondance entre les situations et les actions. Les agents interagissent avec leur environnement mais sans raisonner dessus ;

— les *agents cognitifs*, qui peuvent décider des actions à entreprendre à partir d'états internes qui sont exprimés sous la forme de *croyances* (*Belief*), de *désirs* (*Desire*) et d'*intentions* (*Intention*) : *approche BDI* ;

— les *agents multi-niveaux*, dont les connaissances internes sont organisées sur les différents niveaux d'abstractions d'un modèle conceptuel hiérarchique, permettant ainsi différents niveaux de traitement.

Ces familles d'agents permettent de répondre à différents des objectifs de modélisation. Il en ressort deux grandes familles qui se différencient sur leur façon d'agir, ou plus précisément sur la manière dont ils engagent leurs actions. Il y a d'une part ceux qui sont capables de *réflexion* et qui tiennent compte de leur état, ainsi que d'objectifs pour décider de leurs actions : ce sont les *agents cognitifs*. D'autre part, il y a les agents qui ne réagissent que par *effet réflexe*, ils ne possèdent pas de mécanisme de réflexion, ils ne réagissent qu'aux changements de situation : ce sont les *agents réactifs*.

Ainsi, l'utilisation d'*agents* dans un modèle à base d'agents signifie qu'il n'est pas nécessaire de conceptualiser des raisonnements. Les entités qu'ils représentent dans le système réel ne font que réagir par *réflexe* au changement de situation. Le comportement d'ensemble de l'ABM découle dans ce cas des réactions individuelles de chacun des agents utilisés pour résoudre le comportement du système complexe. Il est possible d'observer dans ce cas des *comportements émergents*.

L'utilisation d'*agents cognitifs* dans un modèle à base d'agents implique de formuler des règles de sélection. Celles-ci servent à déterminer l'action la plus adaptée au contexte attaché à la situation courante de l'agent et ont pour but d'atteindre un objectif précis. Ainsi, un agent cognitif accède à un "savoir".

On utilise ce type d'agents, quand il s'agit de modéliser des entités qui doivent être capables de prendre des décisions au sein d'un système complexe.

M. Wooldridge dans [Wooldridge and Jennings, 1995] retient également les caractéristiques suivantes pour les agents réactifs et cognitifs :

— l'*autonomie* : les agents agissent et évoluent sans intervention directe humaine et dispose d'un certain contrôle sur ses actions et son état ;

— la *capacité sociale* : à l'aide de différents modes de communications, un agent peut interagir avec d'autres agents (potentiellement humains) ;

— la *réactivité* : un agent perçoit son environnement et réagit en fonction de celui-ci (un utilisateur au travers d'une interface, un ensemble d'autres agents, ou encore une combinaison de l'ensemble) ;

— la *proactivité* : un agent ne se contente pas de réagir à son environnement, il est capable de définir un objectif et de prendre une initiative.

Selon nous, les caractéristiques énoncées par *M. Wooldridge* correspondent davantage aux agents cognitifs, notamment la caractéristique de proactivité qui nécessite d'attacher un objectif précis à l'agent.

L'environnement

L'environnement est un élément essentiel au même titre que les agents au sein d'un *SMA*. Nous présentons dans cette sous-section quelques-unes des caractéristiques qui permettent de le définir.

L'environnement possède un rôle et des responsabilités spécifiques, aussi dans de nombreuses approches ils constituent également un agent ou un super-agent.

Dans [Weyns et al., 2007], les auteurs tentent de définir explicitement ce que représente un environnement dans un *SMA*. Ils décrivent l'environnement comme une *"abstraction de première classe qui fournit les conditions environnementales pour que les agents puissent exister, et un plan utilisable pour la construction du* SMA [12].

12. "a first-class abstraction in MAS that provides the surrounding conditions for agents to exist, and an exploitable design abstraction to build MAS applications."

Dans leur ouvrage, [Russell et al., 2003] recensent un certain nombre de caractéristiques que doit posséder un environnement au sein d'un *SMA* :

— L'*accessibilité* : cette caractéristique est matérialisée par un *degré d'accessibilité* que possède un agent vis-à-vis de son environnement ;

— Le *déterminisme* : il s'agit de déterminer dans quelles mesures les *changements d'état* de l'environnement dépendent de l'état courant et des actions des agents ;

— Le *caractère discret ou continu* : cette caractéristique de l'environnement est dépendante du nombre fini d'actions et des perceptions possibles définies dans le modèle ;

— Le *caractère statique ou dynamique* : il s'agit de formuler si l'environnement doit évoluer indépendamment de l'action des agents ou non.

Dans [Odell et al., 2002], les auteurs distinguent l'*environnement communiquant ou social* et l'*environnement physique*. L'environnement communiquant sert à définir les règles de communication, les différents groupes et rôles de chaque agent. L'environnement physique contient les agents et définit les règles et contraintes qui régissent ceux-ci.

Il ressort de ce qui précède que les approches de modélisation proposées dans la littérature sont plus ou moins dépendantes des caractéristiques liées à l'environnement, et que la formulation d'un *SMA* offre une grande liberté de choix au modélisateur. Les choix que celui-ci opère sur l'environnement influent donc en cascade sur les caractéristiques des agents, ainsi que sur les entités du modèle conceptuel dans lesquelles sont exprimées les règles d'interaction.

Dans le cas particulier d'un *SMA* représenté à partir de *modèles cellulaires* (CAM[13]) l'environnement joue un rôle essentiel en tant qu'agent. Dans ce type de modèle conceptuel la description qui est faite de l'environnement possède un rôle central puisque les agents qui sont considérés ici se limitent aux seules cellules fixes représentant l'espace. La dynamique du phénomène est exprimée au sein de ces cellules, qui constituent l'environnement dans lequel le phénomène est observé. Ainsi, dans un *CAM*, ce sont les agents cellules qui constituent l'ossature de l'environnement, et qui sont le lieu des interactions du *SMA*. Parmi les *CAM*, les *automates cellulaires* (CA), constituent les formulations les plus simples, dont le célèbre *CA* de *von Neumann* [Von NEUMAN, 1966] qui a inspiré le *"jeu de la vie"*.

En l'absence de méthode standardisée, la formulation d'un modèle conceptuel à base d'agents reste

13. *Cellular Automata Models* en anglais.

une étape critique. Elle est à l'origine de la construction du modèle informatique en charge de tracer les plans des futurs codes de simulation. Il existe cependant dans la littérature un certain nombre de modèles conceptuels génériques pour les *SMA* qui permettent de formuler clairement et précisément les constituants d'un système complexe en se reposant sur le paradigme agent.

Nous pouvons citer le *modèle AGR* pour *Agent, Groupe, Rôle* détaillé dans [Ferber et al., 2003], ou encore le *modèle IRM4S* proposé par *F. Michel* dans [Michel, 2007b] est basé sur les approches centrées "action" proposées par [Ferber, 1995a].

Les *ABM* sont aujourd'hui largement utilisés dans de nombreux domaines. Depuis maintenant quelques années, les modélisateurs se sont rapprochés des sciences cognitives pour modéliser les processus décisionnels des agents.Ces différents travaux ont donné naissance à des approches de conceptualisation, aboutissant à la formulation d'*architectures cognitives exécutables*. Il s'agit de modèles conceptuels de la cognition humaine, liés notamment à la prise de décision que l'on spécifie dans des agents cognitifs. Spécifiés sous la forme de composants, ces architectures cognitives exécutables sont organisées de façon à faciliter la modélisation des processus cognitifs dans les agents concernés.

Nous présentons quelques-unes de ces architectures cognitives dans la section suivante.

1.3.2 Architectures cognitives exécutables

Nous nous orientons dans ce travail vers la mise en œuvre d'agents dont les processus décisionnels sont construits à partir d'un raisonnement reposant sur des états mentaux et agissant en conséquence dans l'environnement de l'ABM.

Pour cela, nous envisageons d'utiliser une architecture cognitive exécutable afin de modéliser le raisonnement humain d'*"agents pêcheurs"* qui évoluent dans l'environnement virtuel de la pêcherie, notamment lors de l'exécution des processus décisionnels qui engendrent l'activité de pêche.

Il existe dans la littérature un grand nombre d'architectures cognitives exécutables à destination de tels agents, chacune étant inspirée par des objectifs et des applications différentes.

Dans [Balke and Gilbert, 2014], les auteurs ont publié une étude sur quatorze architectures cognitives à destination de simulations sociales (modèle *ABSS*[14]).

Cette étude liste plusieurs types d'architectures cognitives, de la plus simple à base de règles de transition, à la plus complexe permettant de décrire des caractéristiques émotionnelles dans un agent.

14. Agent Based Social Simulation

Celles-ci sont recensées dans 5 grandes familles :

— celles à base de règles, les plus simples à mettre en œuvre ;

— celles de type *BDI* (Croyance-Désir-Intention) et ses extensions, comme *eBDI* (Emotional BDI de [Pereira et al., 2005] et [Jiang and Vidal, 2006]), *BOID* (Croyances-Désirs-Obligations-Intentions) de [Broersen et al., 2002], et BRIDGE de [Dignum et al., 2008] ;

— celles à base de composants normatifs, comme *EMIL-A* de [Andrighetto et al., 2007] ;

— celles à base de composants cognitifs, comme *Consumat* de [Jager and Janssen, 2002] ;

— enfin, celles inspirées par la psychologie et les neuro-sciences comme CLARION de [Sun et al., 2003], *ACT-R* (Contrôle adaptatif de la pensée-rationnelle) de [Lebiere et al., 2013], et *SOAR* (State, Operator And Result) de [Laird et al., 1986].

L'architecture cognitive *Soar*[15] retient plus particulièrement notre attention pour reproduire le raisonnement de nos agents dans une boucle perception/décision/action. C'est-à-dire la description d'entités sociales capables de prises de décisions reposant sur des interactions *agent-environnement* et *agent-agent*.

Ainsi, nous nous orientons dans ce travail vers la mise en œuvre d'agents dont les processus décisionnels sont construits selon une architecture cognitive *Soar* permettant de raisonner à partir d'états mentaux et d'agir en conséquence dans l'environnement. L'usage et le fonctionnement de *Soar* seront détaillés dans la dernière section où nous proposerons un modèle conceptuel.

Précédemment, nous avons vu que la simulation environnementale qu'implique ce travail de recherche nécessite de mettre en relation des compétences issues de différents domaines scientifiques. Dans le cas de notre problématique, il s'agit d'intégrer des concepts de modélisation empruntés à la fois à des modèles de population issus de la biologie ainsi que des concepts de modélisation empruntés aux modèles économiques afin de décrire le comportement des agents pêcheurs. L'usage des formalismes de spécification formelle vont permettre de faciliter ce travail de modélisation. Nous les décrivons plus avant dans la section suivante.

15. *SOAR* signifiait à l'origine *State, Operator And Result* (*État, Opérateur Et Résultat*). *Soar* n'étant plus considéré comme un acronyme, ce terme ne doit donc plus être écrit en majuscules.

1.4 Formalismes de spécification

Dans le processus de modélisation en quatre étapes que nous proposons, il est recommandé d'utiliser un formalisme de spécification formelle afin de décrire le modèle conceptuel de manière précise et non ambiguë. Il existe pour cela un certain nombre de formalismes, dont les principaux sont recensés dans [Ramat, 2003], sous la forme d'une classification que nous avons reproduit dans le tableau 1.1. Cette classification repose sur la caractéristique de l'état du modèle, la caractéristique de son horloge temporelle, ainsi que sur la caractéristique de la prise en compte de la dimension spatiale.

Etats	Temps	Espace	Formalisme
Continu	Continu	Absent	Équations différentielles ordinaires
		Continu	Équations aux dérivés partielles
		Discret	
	Discret	Absent	Équations aux différences
		Continu	Équations aux différences et spatialisées
		Discret	
Discret	Continu	Absent	Modèles à évènements discrets
		Continue	
		Discret	
	Discret	Absent	Automates à états finis
		Continu	Modèles à temps discret
		Discret	Automates cellulaires

TABLE 1.1 – Classification des formalismes, proposée par [Ramat, 2003] en fonction des caractéristiques de l'état du modèle, de l'espace et du temps.

Dans le cadre de notre problématique plusieurs de ces formalismes semblent convenir, notamment ceux qui permettent de décrire des changements d'états qui s'opèrent de manière discrète. En effet, il faut que l'on puisse s'affranchir de l'aspect continu du temps afin de ne réagir qu'à un ensemble d'évènements particuliers, et d'éviter une surcharge de calculs pour le simulateur.

Le formalisme *Discrete EVent system Specification* (*DEVS*) proposé par *B.P Zeigler* [Zeigler et al., 2000] possède l'avantage de pouvoir représenter l'ensemble des formalismes du tableau 1.1 (aspect multi-formalismes), de pouvoir être étendu à de nombreux cas particuliers, aussi nous le détaillons plus avant dans la section suivante.

1.4.1 Formalismes DEVS et PDEVS

Le formalisme DEVS fut proposé par *B.P Zeigler* dans son premier ouvrage [Zeigler, 1976], puis étendu à de nombreux domaines dans un second ouvrage [Zeigler et al., 2000]. Ce formalisme est à la base d'une méthodologie de conceptualisation universelle servant à modéliser les systèmes complexes sur la base des préceptes de la simulation à évènements discrets. Il est fondé sur la *théorie des systèmes* et il repose sur la définition de deux grands types de composants de modélisation : *le modèle atomique* et *le modèle couplé*.

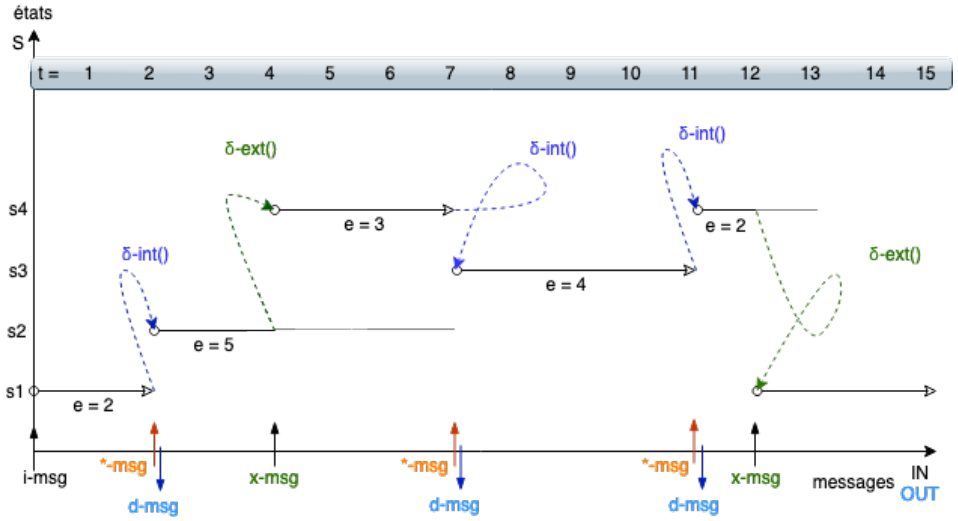


FIGURE 1.6 – Évolution dans le temps des états d'un modèle DEVS.

Les modèles atomiques sont les composants de base servant à décrire de manière formelle le comportement du système à étudier. Les modèles couplés servent à structurer le système complexe comme résultant d'un réseau de composants couplés entre eux. Chaque composant pouvant être lui-même, un modèle atomique, ou bien un modèle couplé. Cette approche fut connue dans une première version sous le vocable de "*DEVS classique*", puis améliorée pour permettre la gestion de possibles conflits d'évènements, ce qui donna naissance à *PDEVS*, environ deux décennies après que le formalisme DEVS fut proposé [Chow and Zeigler, 1994]. Le formalisme de spécification formelle *PDEVS* remplace en général aujourd'hui le formalisme *DEVS* dans la plupart des projets de simulation.

La figure 1.6 donne une représentation de l'évolution des états d'un modèle dans le temps en fonction d'évènements. La simulation est gérée par les évènements $x, *$ et d qui declanchent les fonctions de

transition ($\delta_{ext}()$) et $\delta_{int}()$).

Le formalisme *PDEVS* permet de décrire formellement des structures de données pour les modèles à évènements discrets, capables de gérer de manière efficace la survenue d'évènements simultanés dans l'algorithme de la simulation. Aussi, on distinguera deux catégories d'évènements conflictuels :

— Le premier cas de conflit traite de l'arrivée de plusieurs événements en entrée à une même date. C'est le cas sur la figure 1.7 lorsque plusieurs x-messages sont présent dans la file d'entrée (BAG). L'exemple basique est celui d'un modèle atomique qui à un temps de simulation donnée t_i perçoit deux évènements en entrée. Ces deux entrées ayant potentiellement un effet sur le changement d'état du système, le modélisateur doit définir malgré tout un ordre de priorité, en gardant en mémoire leurs effets, avant de calculer le nouvel état.

— Le deuxième cas de conflit traite de la survenue simultanée d'un évènement interne et d'un évènement externe, c'est-à-dire de l'exécution simultanée des fonctions de transitions interne et externe.

Selon le formalisme PDEVS, le modélisateur a la possibilité de spécifier une fonction $\delta_{con}()$ pour gérer ces conflits.

Ce cas est illustré sur la figure 1.7 où la fonction $\delta_{con}()$ est sollicitée pour gérer les conflits où un évènement externe survient sur l'entrée d'un modèle atomique et au même instant effectue un changement d'état planifié par le modèle (e = ta). Dans ce cas, le formalisme DEVS classique impose d'ignorer l'évènement externe.

Par défaut, sans aucun conflit et afin d'avoir un comportement identique au modèle DEVS classique le comportement de la fonction $\delta_{con}()$ correspond à l'exécution de la fonction $\delta_{int}()$.

Spécification des modèles

Le modèle atomique PDEVS est décrit par un tuple $\{X, Y, S, t_a, \delta_{con}, \delta_{int}, \delta_{ext}, \lambda\}$ avec :

— X est l'ensemble des ports et des valeurs d'entrée ;

— Y est l'ensemble des ports et des valeurs de sortie ;

— S est l'ensemble des états partiels du système ;

— t_a : est la fonction d'avancement du temps ;

— $\delta_{int}(Q \to Q)$: est la fonction de transition interne, déclenchée par l'arrivée d'un *-message ;

— $\delta_{ext}((Q,\omega) \rightarrow Q)$: est la fonction de transition externe, déclenchée par l'arrivée d'un x-message où, ω est l'ensemble des sacs d'entrées appartenant à X, Q est l'ensemble des états totaux, Q = (s, e) avec s ∈ S, $0 \leq e \leq t_a(s)$, e est le temps écoulé depuis la dernière transition ;

— $\delta_{con} = \delta_{ext}(\delta_{int})$: la fonction de conflit qui se déclenche lorsque plusieurs messages sont en attente au même instant ;

— $\lambda(Q \rightarrow Y)$: la fonction de sortie qui génère un d-message.

De la même manière que pour la version classique de DEVS, en l'absence d'événements sur les ports d'entrées, le modèle conserve un état passif jusqu'au prochain évènement déclenchant la transition interne (δ_{int}). Une sortie est alors générée par la fonction de sortie (λ), suivie de l'exécution de la fonction de transition interne (δ_{int}).

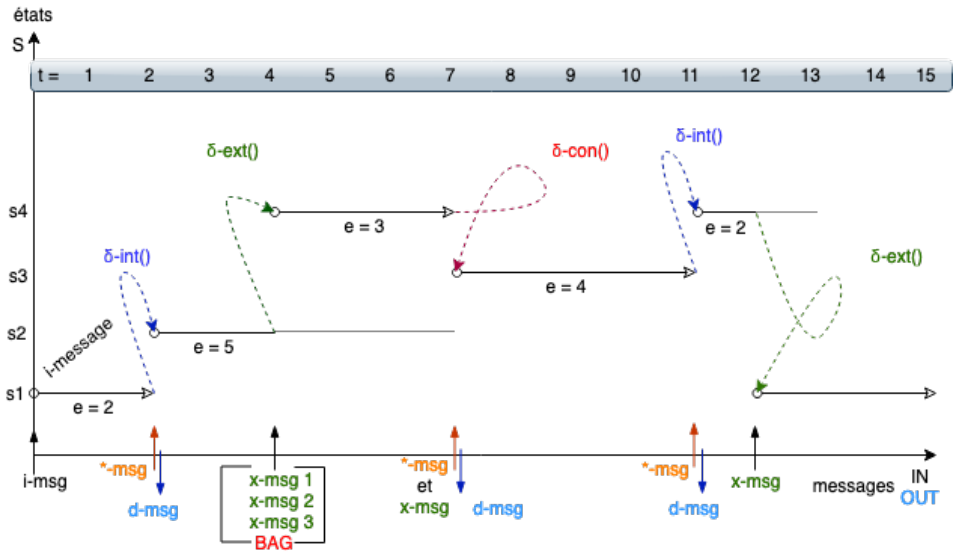


FIGURE 1.7 – Simulation d'un modèle PDEVS.

Le modèle couplé définit comment interconnecter des sous-modèles (atomique ou couplé) afin de former un nouveau modèle selon une hiérarchie de composition.

Un modèle couplé est une structure : $\{X, Y, D, \{Md/d \in D\}, EIC, EOC, IC, Select\}$ avec :

— les définitions de X et Y sont identiques à celles du modèle atomique ;

— D est l'ensemble des noms des composants (modèles) du modèle couplé ;

— Md est un modèle DEVS atomique ou couplé. Les variables représentant les entrées et les sorties

du modèle seront indexées par l'identifiant du modèle. Les entrées et les sorties du modèle couplé sont connectées aux entrées et sorties des modèles composants le modèle couplé ;

— EIC représente l'ensemble des ports d'entrée ipN du modèle couplé connectés aux ports d'entrée ipd des sous-modèles le composants ;

— de la même façon, EOC représente l'ensemble des ports de sorties ipd du modèle couplé connectés aux ports de sorties des sous-modèles qui le composent ;

— IC : à l'intérieur du modèle couplé, les sorties d'un modèle peuvent être couplés aux entrées des autres modèles. Une sortie d'un modèle ne peut pas être couplés à l'une de ses entrées.

Sur cette base, le formalisme DEVS peut-être dérivé ou étendu.

1.4.2 Extensions du formalisme DEVS

Dans cette section, nous détaillons les extensions du formalisme DEVS que nous utilisons dans ce travail.

Cell-DEVS

Le formalisme *Cell-DEVS*[16] est une extension du formalisme *DEVS* qui permet de modéliser les *systèmes spatiaux*, à l'aide des concepts de la simulation à évènements discrets [Wainer and Giambiasi, 2001]. Ce formalisme fut étendu afin de se conformer à *PDEVS* dans [Troccoli and Wainer, 2003].

Le formalisme *Cell-DEVS* est une extension du formalisme DEVS dédiée spécifiquement à la définition de modèles cellulaires ou *CAM*[17]. Les CAM possèdent une *hiérarchie d'abstraction* pouvant s'étendre sur plusieurs niveaux. Bien que théoriquement, une infinité de niveaux peuvent être définis, on considère habituellement deux niveaux d'abstraction : un *niveau global* et un *niveau local*. Cette extension étant propre au domaine des *CAM*, elle nécessite une modification de la structure de base des modèles atomiques et couplés *DEVS* comparativement à l'approche *DEVS classique*, ainsi que des simulateurs abstraits correspondant.

Le lecteur intéressé par davantage de précision sur ce formalisme de spécification système, tirera bénéfice de la lecture des références [Wainer and Giambiasi, 2005, Wainer, 2000, Wainer, 2018].

16. Cell-DEVS pour *Cell-Discrete Event System Specification* [Wainer, 2014, Wainer and Giambiasi, 2005].
17. CAM pour Cellular Automata Models.

DEVS dynamique : DSDEVS, DSDE, M-DEVS, dynDEVS

Nous avons vu que le formalisme DEVS permet de décrire la structure et le comportement d'un système complexe avec des modèles atomiques. Cependant, lors de la simulation la structure d'un modèle DEVS est fixe, le formalisme DEVS classique ne permettant pas son évolution structurelle. La possibilité de modifier cette structure lors de la simulation. C'est-à-dire d'ajouter et de supprimer des sous-modèles ainsi que de modifier les connexions pourrait permettre de mieux représenter la dynamique de certains systèmes complexes, tels que les systèmes vivants ou naturels. Cette caractéristique est capitale pour décrire la dynamique structurelle d'un SMA. En effet, la représentation d'une population, ou plus généralement de systèmes comportant des entités susceptibles de se regrouper et de se désagréger, en sont de bons exemples (dynamique structurelle).

[Barros, 1995] proposa pour cela, Dynamic Structure DEVS (DS-DEVS), un formalisme dans lequel les modèles de base sont identiques aux modèles DEVS classiques, mais à la différence près que la structure des modèles couplés peut varier au cours du temps.

Dans [Barros, 1997, Barros, 1998] une évolution nommée DSDE est proposée, elle se base sur le formalisme PDEVS de [Chow and Zeigler, 1994]. Cette extension offre donc les avantages de gestion des simultanéités de PDEVS.

DS-DEVS puis DSDE introduisent la notion *de composant exécutif* qui formalise une structure évolutive dans un modèle conceptuel. Il s'agit d'un modèle atomique spécifique qui gère la dynamique de couplages. Il est défini par :

$$M_x = (X_x, Y_x, S_x, \delta_{int_x}, \delta_{ext_x}, \lambda_x, ta_x)$$

L'état du modèle exécutif qui contient les informations sur la structure du réseau. Un changement d'état de ce modèle, implique une modification de la structure du modèle couplé.

Une autre approche permettant de gérer une évolution de la structure des modèles atomiques pendant la simulation est dynDEVS proposée par [Uhrmacher, 2001].

Nous pouvons également citer *MDEVS* signifiant *Mobile Discrete Event System Specification*, proposée par [Kim and Kim, 2000]. Cette formalisation propose également des changements de structure pendant la simulation via l'ajout d'entités aux modèles couplés. Ce sont ces dernières qui prendront en charge les

évolutions de la structure.

Ces trois approches permettent de formaliser des modèles dont la structure peut évoluer au cours de la simulation.

Ainsi, dans un premier temps, ce fut le formalisme DSDE qui permit cela en introduisant la notion *de modèle exécutif* ainsi que la fonction γ qui cartographie l'état de la structure et l'ensemble des modèles du réseau. Dans un second temps, le formalisme DynDEVS apporta deux nouvelles fonctions au formalisme DEVS afin de permettre des changements de structure : une "fonction de transition de modèle" qui est incluse dans le modèle atomique dynamique ; une "fonction de transition de réseau", qui est incluse dans un modèle dynamique couplé. Enfin, le formalisme MDEVS proposa d'ajouter une dynamique comportementale aux modèles couplés en les dotant d'une fonction de transition et d'un état de la structure du réseau.

Dans le cadre de notre collaboration de recherche (notamment l'usage de l'environnement VLE) et par extension du travail réalisé au sein de l'UMR SPE d'accueil, nous avons fait le choix de privilégier le formalisme DSDE quant à la définition du modèle conceptuel de la pêcherie. DSDE est le formalisme utilisé par VLE.

Les spécifications Multicomposant

Historiquement, c'est *B.P. Zeigler* qui proposa le premier, dans son ouvrage [Zeigler et al., 2000], un formalisme de spécification système *Multicomposant*[18] (*MC*) pour décrire formellement les *systèmes dynamiques complexes spatiaux* et poser les premières bases d'une réflexion portant sur le *modèle conceptuel multicomposant*.

Ce formalisme sert à spécifier le modèle conceptuel d'un *système complexe spatial*, à travers le prisme de la composition de sous-systèmes de complexité moindre et capables d'interactions.

La figure 1.8 est illustration du *modèle conceptuel multicomposant* guidant la pensée logique des processus de modélisation mis en œuvre dans les travaux de recherche qui sont présentés dans ce document.

Il convient de remarquer que l'usage d'un *modèle multicomposant* est souvent lié aux problématiques environnementales. Cela implique la mise en correspondance de nombreuses compétences scientifiques dès la définition des éléments conceptuels. Le but final est d'intégrer dans une seule et même entité, c'est-à-

18. Multicomponent System Specification.

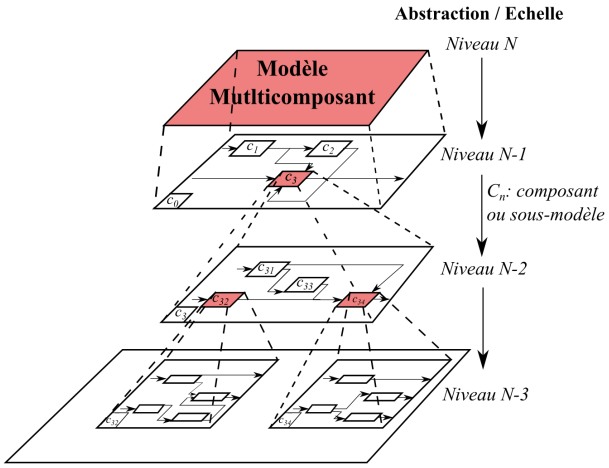


FIGURE 1.8 – Illustration d'un *modèle conceptuel multicomposant*.

dire dans le *modèle multicomposant*, les nombreuses informations [19] provenant des différentes disciplines (points de vue).

Dans ce type de modèle, le modélisateur en charge de l'intégration doit traiter avec méthode, les données du *niveau individuel* ou *niveau local* le plus bas (*niveau N-i*), jusqu'au *niveau global* (*niveau N*).

Le modèle conceptuel du multicomposant facilite l'appréhension de la complexité inhérente aux différents éléments constituant le système complexe, notamment grâce à l'usage de paradigmes, de formalismes et de composants, pour la plupart issus de la *théorie de la modélisation et de la simulation* [Zeigler et al., 2000, von Bertalanffy and Sutherland, 1974].

Avec le formalisme DEVS, les composants d'un système interagissent par l'intermédiaire des ports d'entrées/sorties de façon modulaire. En aucun cas, un modèle atomique n'a accès aux variables d'état d'un autre modèle [20]. Dans le cas du formalisme DEVS-Multicomponent [Zeigler et al., 2000], il est possible de définir des systèmes non modulaires, des systèmes couplés comportant des composants possédant chacun leur fonction de transition, leurs états propres, et pouvant également influencer ou être influencés par les autres composants [21]. Ici, contrairement à un modèle couplé, les composants ne possèdent pas d'interface d'entrée/sortie.

Dans un modèle Multicomposant de type DEVS, chaque composant gère individuellement sa fonction

19. Ces informations peuvent être très diverses, à savoir de simples données environnementales, mais également des instruments plus formels tels que des outils mathématiques spécialisés dans un domaine spécifique.
20. La notion de variable partagée est presque une nécessité pour les SMA.
21. Nouvelle notion importante pour les SMA orientés d'après de paradigme influence/réaction

d'avancement du temps et ses évènements internes. Lorsqu'un évènement interne survient, la fonction de transition interne est exécutée ; une sortie est éventuellement générée, ce qui entraîne un changement d'état du composant. Il est a noter que le nouvel état du composant dépend des états des influenceurs I_d et modifie l'état total des composants influencés Ed. Si plusieurs composants doivent s'activer au même moment, c'est la fonction $Select$ qui joue le rôle d'arbitre. Un évènement externe en entrée d'une entité DEVS-Multicomponent peut être géré par n'importe quelle fonction de transition externe $\delta_{ext,d}$ des composants. Tous les composants n'ont pas nécessairement de fonction de transition externe définie.

L'utilisation de la spécification système Multicomposant se prête à la représentation des systèmes complexes spatialisés, tel qu'une pêcherie. Comparativement au formalisme *Cell-DEVS*, le formalisme Multicomposant est plus simple à mettre en œuvre et beaucoup plus générique. Cependant, dans sa version de base le formalisme Multicomposant ne permet pas de décrire précisément des entités autonomes mobiles. Afin de pouvoir spécifier ce concept dans notre processus de modélisation, il nous faut maintenant considérer les paradigmes des *Systèmes Multi-Agents*(*SMA*). Nous les présentons dans les sections suivantes.

1.4.3 Analogies DEVS/SMA

De nombreuses proposition d'association des *SMA* à partir des formalismes unificateurs *DEVS* et *PDEVS* existent. Une première approche propose l'utilisation du formalisme *PDEVS* pour lier un simulateur externe à un modèle PDEVS. Nous pouvons notamment citer MECSYCO [Camus, 2015] qui permet d'intégrer des simulateurs SMA, tels que NetLogo dans des composants DEVS (warpping). Ce framework permet de coupler des simulateurs différents.

D'autres travaux proposent une analogie entre un agent et un modèle DEVS (atomique et/ou couplé) [Uhrmacher and Arnold, 1994, Uhrmacher and Schattenberg, 1998]. Parmi ces travaux, nous retrouvons deux approches basées sur DSDEVS [Duboz et al., 2002, R. Duboz et al., 2005] puis sur DSDE [Quesnel, 2006].

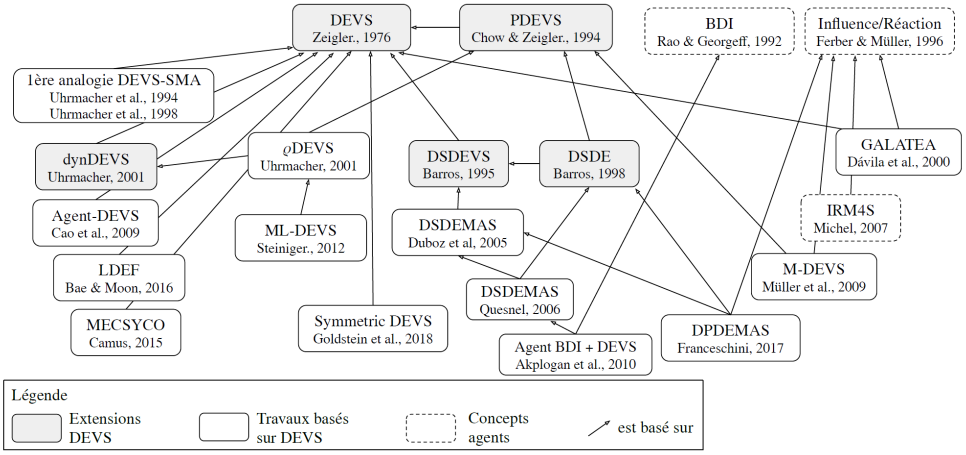


FIGURE 1.9 – Positionnement dans l'existant [Franceschini and Duboz, 2020]

DSDEMAS Nous retrouvons dès les années 1990 des travaux portant sur la spécification de modèles SMA à l'aide du formalisme DEVS.

La spécification DSDEMAS, pour Dynamic Structure Discrete event Multiagent Systems [Duboz et al., 2006] est une formalisation du paradigme agent basée sur le formalisme DEVS et l'extension DSDEVS [Barros, 1995].

[Quesnel, 2006] complète ensuite ces travaux de spécification, en se basant sur la variante parallèle des formalismes (i.e. PDEVS et DSDE).

DPDEMAS Plus récemment, la spécification DPDEMAS pour Dynamic Parallel Discrete event Multiagent Systems [Franceschini, 2017] basée sur DSDE et influencé par l'approche *influence/réaction* [Ferber and Müller, 1996]. Cette spécification formalise de façon générique la structure et le comportement des entités du SMA.

Cette approche vise à améliorer la reproductibilité des expériences numériques dédiées aux systèmes multi-agents. Le positionnement de DPDEMAS dans l'existant est illustré dans la figure 1.9.

La spécification DPDEMAS [Franceschini et al., 2017] définit la structure d'un modèle SMA accompagné de sa sémantique pour la simulation à évènements discrets. Dans cette optique, elle respecte l'ensemble des propriétés exposées dans la littérature [Ferber, 1995b, Michel, 2007a, Soulié, 2001].

En ce qui concerne les agents, elle respecte les principales caractéristiques d'un agent [Wooldridge and Jennings, 1995] : *proactivité*, *réactivité*, *sociabilité* et *autonomie*, en respectant la *contrainte d'intégrité interne* [Michel, 2007a] (l'agent dispose du contrôle total de son état).

En ce qui concerne les environnements, (1) elle permet la définition d'environnements multiples [Soulié, 2001] (physiques et/ou sociaux), (2) elle permet une dynamique endogène, (3) elle respecte la contrainte de localité (les agents perçoivent leur voisinage en fonction d'une position) et enfin, (4) elle respecte la contrainte d'intégrité de l'environnement, qui consiste à s'assurer que les agents n'effectuent pas de modification directe de l'environnement.

La spécification DPDEMAS autorise une dynamique structurelle afin de matérialiser les aspects dynamiques d'un SMA (création/destruction d'entités, etc.). Elle permet la gestion d'actions simultanées afin d'éviter les problématiques liées à l'ordonnancement des agents [Lawson et al., 2000], qui peut influer sur les résultats de simulation.

Enfin, dans DPDEMAS, le système multi-agents (SMA) est décrit par des modèles d'environnements (physiques et/ou sociaux), eux-mêmes composés par le corps des agents, et des modèles correspondant aux esprits des agents. L'esprit est vu comme le système cognitif de l'agent, mais aucune spécification n'est proposée.

Le fait de décomposer architecturalement l'agent en deux parties pour distinguer l'esprit et le corps n'est pas une approche courante dans la communauté SMA [Saunier, 2015]. Cette approche peut être associée à la théorie de l'approche incarnée de la "cognition" [Wilson, 2002]. Celle-ci considère que l'esprit, le corps et l'environnement jouent un rôle dans le processus cognitif global, par opposition à l'approche "cognitiviste" classique [Haugeland, 1989] où le corps est entièrement déconnecté du siège de l'intelligence.

[Saunier, 2015] considère le corps comme une abstraction de premier ordre : bien que l'esprit puisse prendre n'importe quelle décision, les limites à la réalisation de ces décisions sont imposées à la fois par les capacités du corps et par les règles de l'environnement.

Cette décomposition de l'agent présente plusieurs intérêts conceptuels. Elle permet à un agent d'exister au sein de plusieurs environnements. Le corps est alors la manifestation physique de l'agent au sein d'un environnement. C'est à travers lui que l'agent peut exposer certaines caractéristiques observables par les autres agents, qu'il peut percevoir des informations et agir. Les travaux de [Soulié, 2012] sur l'approche multi-environnement sont les premiers à séparer de manière explicite le corps de l'esprit de l'agent.

On retrouve ensuite cette idée dans les modèles conceptuels AGRE [Ferber, 1995b] ou encore MASQ [Dinu et al., 2012]. Plus qu'un simple intermédiaire entre l'environnement et le processus décisionnel de l'agent, le corps permet également de satisfaire la contrainte d'intégrité interne de l'agent [Michel, 2015], qui consiste à s'assurer que les agents réalisent leur processus de délibération en disposant du contrôle de

leur état.

Ainsi, cette distinction permet de considérer la partie physique de l'agent comme une entité à part entière de l'environnement. Un agent ne perçoit pas le système cognitif de ses pairs, il perçoit uniquement leurs corps. La description d'un agent à partir du couple corps/esprit est très important. L'esprit est donc le siège des décisions et le corps la représentation de l'agent dans un environnement.

Le système multi-agents DPDEMAS est un modèle DSDE ; le système cognitif est un modèle, tout comme l'environnement.

En ce qui concerne l'environnement, il réalise une topologie à travers la fonction de distance, qui place les positions les unes par rapport aux autres. Comme l'environnement, l'agent est un concept central. Chaque système cognitif est associé à un agent, et chaque corps appartient à un agent. C'est à travers son corps que l'agent est plongé dans l'environnement.

Afin de spécifier l'agent, DPDEMAS s'appuie sur l'analogie proposée par [Uhrmacher and Arnold, 1994], reprise par [Duboz et al., 2002] entre un modèle PDEVS et un agent, laquelle permet de mettre en évidence la capacité d'un modèle PDEVS (couplé ou non) à satisfaire les propriétés définissant un agent selon [Wooldridge and Jennings, 1995] : autonomie, proactivité, réactivité, sociabilité. Cette approche, nous semble donc tout indiquée pour concevoir notre modèle de décision et simuler nos pêcheries.

Cette section met fin à notre état de l'art, sans avoir la prétention d'être exhaustive, nous avons du présenter beaucoup de notions. L'objectif étant de poser les bases de nos travaux. En guise de bilan, nous proposons une approche conceptuelle qui complète DPDEMAS et intègre SOAR dans un modèle atomique PDEVS représentant l'esprit d'un agent.

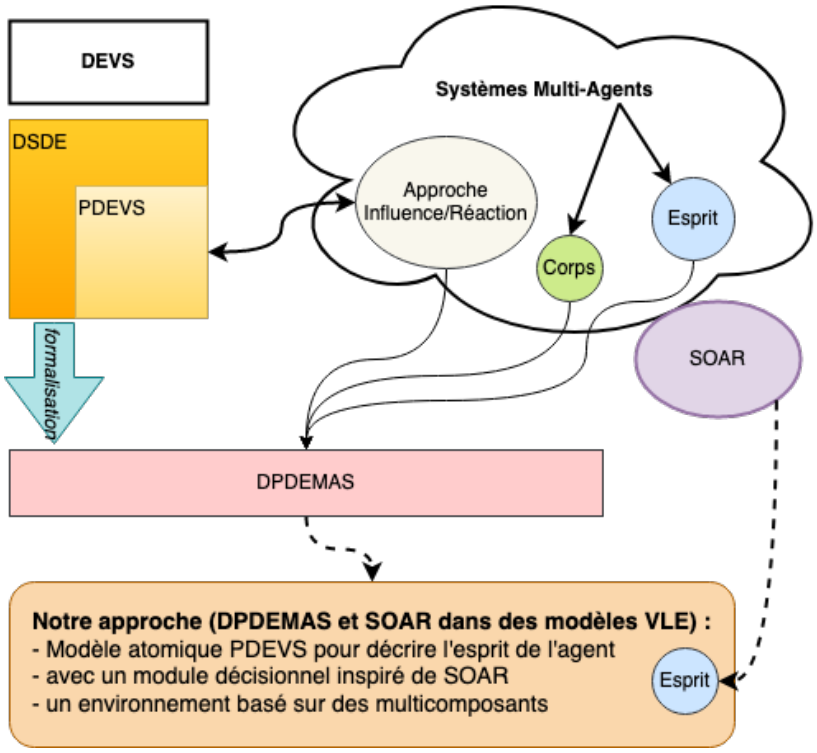


FIGURE 1.10 – Concepts généraux de l'approche proposée.

1.5 Apports conceptuels et analogie soar

Notre objectif est d'utiliser la simulation informatique pour modéliser le système complexe d'une pêcherie dans le but de pouvoir tester des scénarios et d'effectuer des expériences virtuelles. Pour cela, il faudra modéliser les interactions entre les pêcheurs (agents mobiles et autonomes), les stocks de poissons (agents fixes), et l'environnement (agent omniscient), et intégrer des prises de décisions afin d'engendrer des comportements de pêche réalistes. Pour cela, nous proposons d'étendre les travaux existants sur la formalisation DPDEMAS à la définition d'agents autonomes capables de *prise de décision* sous la forme d'*agents cognitifs*. Nous nous reposerons pour cela, sur une architecture cognitive computationnelle que nous décrirons plus avant dans ce document. Notre proposition de modélisation est illustrée sur la figure 1.10.

Pour cela, nous complétons dans un premier temps la spécification de l'agent *DPDEMAS* afin que l'on puisse intégrer dans un second temps, une architecture cognitive computationnelle.

Nous spécifions dans cette optique, des modèles atomiques et couplés auxquels nous ajoutons les méthodes et les attributs nécessaires pour prendre en charge différents modules inspirés de l'architecture cognitive *Soar* que nous présenterons plus loin dans ce document.

Un agent réactif possède un comportement assez simple. Nous pouvons le résumer à perception/action, avec l'approche influence/réaction utilisée dans DPDEMAS ce cycle en deux étapes est conservé mais tient compte de plus de facteurs car la phase d'influence permet de collecter, à l'image du bag PDEVS, plusieurs perceptions concurrente dans le temps. Comme nous avons pu le voir, la bascule d'agent réactif à agent cognitif vient compléter ce dispositif en deux phases en ajoutant de nouvelles étapes. Nous pouvons le résumer en *influence/raisonnement/réaction*.

Dans la *phase de raisonnement*, qui représente l'un de nos apports, nous allons pouvoir appliquer des règles simples de croyance ou de désir de l'agent, mais également des processus plus complexes d'apprentissage ou d'optimisation. Cette phase va regrouper les étapes de mise à jour des croyances et des intentions, amélioration des solutions et enfin sélection d'une solution (étapes un peu modifiées et empruntées à BDI [Rao and Georgeff, 1991]).

D'après [Edmonds and Moss, 2004], les architectures *BDI* et les architectures cognitives en général comme *Soar*, permettent aux modélisateurs de spécifier les processus cognitifs des agents.

Le lecteur intéressé pourra trouver plus de détails sur chacune des approches dans [Balke and Gilbert, 2014]. Une comparaison entre BDI et Soar est aussi proposée. Elle souligne que les deux approches tendent vers le même but à partir d'idées assez similaires et qu'elles sont en faite plus complémentaires que concurrentes. Soar est généralement utilisé en sciences cognitives et adopte une approche empirique alors que BDI est plutôt utilisée par des logiciens et des philosophes.

Dans notre cas, nous avons privilégié Soar.

1.5.1 Fonctionnement de Soar

Nous présentons ici l'architecture cognitive computationnelle *Soar*, son fonctionnement et comment nous souhaitons l'utiliser. Soar fait partie des architectures parmi les plus utilisées et est inspirée des sciences sociales et cognitives ce qui correspond bien aux besoins de notre problématique de modélisation et de simulation.

L'architecture cognitive *Soar* fut décrite initialement dans [Laird et al., 1986] et [Rosenbloom et al., 1993].

Elle est issue des sciences cognitives et de la recherche en Intelligence Artificielle (IA) dans le but de modéliser le comportement humain. Ce type d'architecture repose sur la formulation de processus de décision à base de règles comportementales qui expriment des comportements motivés par un but à atteindre, et par la possibilité d'apprendre en retour des résultats obtenus. Elle est notamment utilisée pour exprimer les interactions *agent-environnement* et *agent-agent* dans un contexte d'apprentissage (SOAR 9).

Soar a déjà été utilisée pour des simulations à grande échelle du comportement humain. La version courante est la 9.6, elle est décrite sur le web [22]. Le site des auteurs propose également au téléchargement différent IDE pour le développement d'agents SOAR, ainsi que des exemples de code et un grand nombre de documents et de ressources. En plus de ces documents, il existe des forums et des groupes de discussions pour les utilisateurs de SOAR ainsi qu'une liste de diffusion SOAR où les utilisateurs peuvent demander de l'aide.

La figure 1.11 est une illustration de l'architecture cognitive *Soar* dont nous nous inspirons dans ce travail. Les parties de couleur orange décrivent l'architecture initiale de [Laird, 2012]. Le schéma dans son ensemble décrit la dernière version (SOAR 9) proposée dans [Laird, 2008].

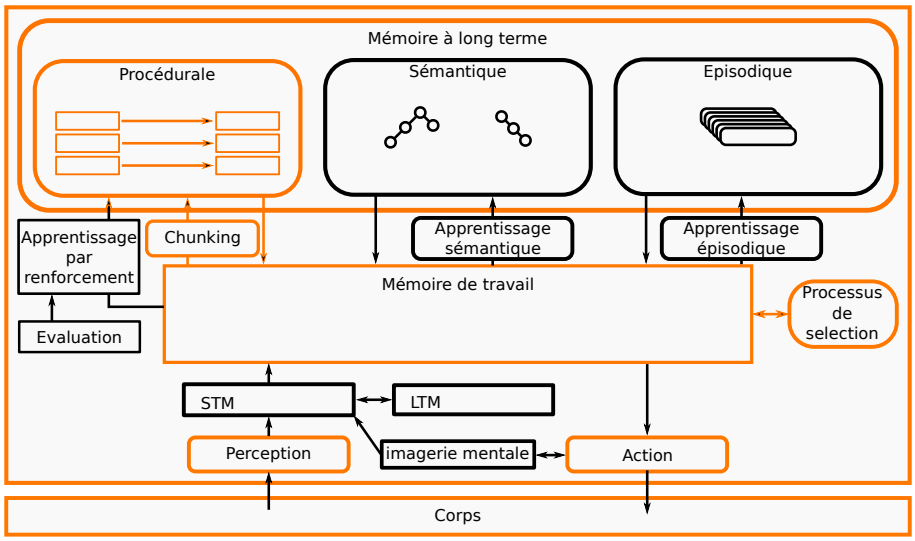


FIGURE 1.11 – L'architecture cognitive *Soar* d'un agent décisionnel.

Au sein d'un agent décisionnel, l'architecture *Soar* nécessite deux types de mémoires : la *mémoire à long terme* et la *mémoire de travail*. La *mémoire à long terme* (symbolique) stocke un ensemble de règles

22. soar.eecs.umich.edu

décisionnelles ainsi que l'historique des décisions de l'agent (actions/récompenses). La *mémoire de travail* est structurée sous la forme d'un graphe, elle stocke les caractéristiques et les relations de l'agent avec les autres agents, ainsi que les relations de l'agent avec l'environnement. La mémoire de travail sert à évaluer la situation courante de l'agent via un processus de sélection. Ce processus consiste à appliquer des règles de sélection fondées sur les connaissances stockées dans la *mémoire à long terme* et dans la *mémoire de travail*. Plus généralement, il s'agit de mettre en correspondance les informations que l'agent perçoit sur ses entrées, avec les informations stockées dans la mémoire à long terme. Il s'en suit la génération de commandes ou d'actions motrices, depuis la sélection d'opérateurs. Ainsi, l'état courant d'un agent du modèle est calculé à partir de règles de sélection qui impliquent les actions à exécuter lors de la transition d'état. Ces règles reposent sur l'usage d'une structure associative depuis laquelle il est possible de générer des connaissances en fonction de l'état courant de l'agent. En général, la plupart des algorithmes de prise de décision reposent sur une seule règle conditionnelle, aussi la prise de décision correspond à une instruction conditionnelle (Si condition Alors proposition). Ainsi, la mise en œuvre d'une architecture cognitive *Soar* nécessite de construire le processus décisionnel à partir d'un ensemble de règles, capables d'interpréter plusieurs éléments de connaissance au même instant.

Depuis la version 9, l'approche cognitive *Soar* offre en outre la possibilité de spécifier des modules d'apprentissage et de mémoire. Cette caractéristique permet de spécifier plus facilement les connaissances grâce à un module d'apprentissage par renforcement. Celui-ci est lié aux connaissances procédurales ainsi qu'à la mémoire de travail comme cela est illustré sur la figure 1.11. Le module d'apprentissage permet d'ajuster la sélection des actions en fonction d'un *"système de récompenses"* en rapport avec l'environnement et la situation de l'agent. Les récompenses spécifient la valeur attendue par un opérateur pour un état. Lorsqu'un opérateur est sélectionné, toutes les règles qui déterminent les valeurs attendues sont mises à jour en fonction de nouvelles récompenses. Ce processus d'évaluation des opérateurs est appliquée à tous les objectifs et sous-objectifs du système, ce qui permet de sélectionner les opérateurs adaptés à la situation. Le processus d'évaluation est liée à la phase d'apprentissage par renforcement, ainsi qu'à la mémoire de travail, comme cela est illustré sur la figure 1.11.

Ces modules nous seront particulièrement utilent afin de reproduire le raisonnement des agents économiques dans une boucle perception/décision/action, à savoir le raisonnement d'entités sociales capables de prises de décisions reposant sur des interactions *agent-environnement* et *agent-agent*.

Plus particulièrement, nous utilisons l'architecture cognitive computationnelle *Soar* afin de reproduire

un raisonnement humain réaliste des agents pêcheurs évoluant dans l'environnement virtuel de la pêcherie lors du processus de la prise de décision dans l'activité de pêche.

Dans cette optique, nous allons proposer une analogie des modules d'intérêt de *Soar* à partir de notre modèle PDEVS représentant l'esprit d'un agent.

1.5.2 Analogie DEVS/SOAR

Les modules de l'approche SOAR qui nous intéressent le plus sont les mémoires et les briques d'apprentissage et de sélection. Nous allons les utiliser pour définir les processus de décision de nos agents.

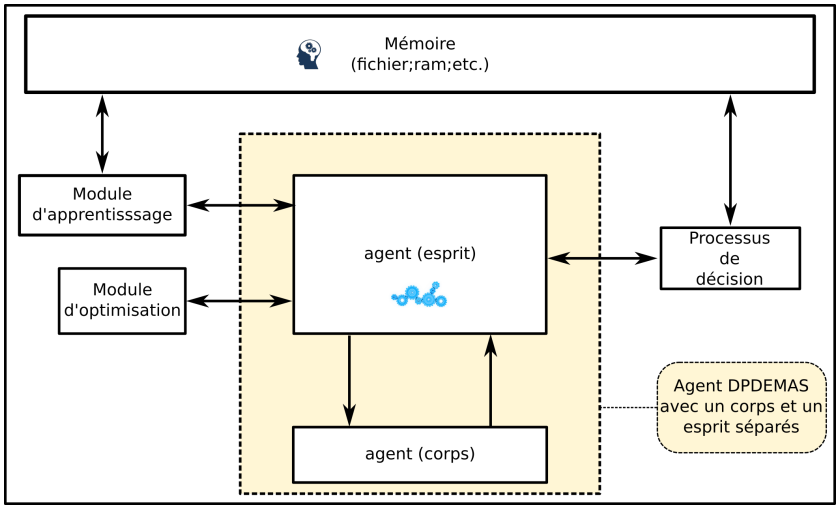


FIGURE 1.12 – Concepts généraux.

Comme nous l'illustrions dans la figure 1.12, nous représentons l'équivalent des deux types de mémoires :

— une mémoire permettant de stocker les informations qui doivent être conservées et qui constituera notre base de connaissances. Dans les chapitres suivant nous utiliserons cette mémoire stoker les données d'apprentissage sous la forme d'un dictionnaire ;

— le module central qui correspond à l'esprit de l'agent et qui si on regarde Soar correspond une mémoire de travail. C'est le coeur du modèle, elle va contenir l'état courant de l'agent et lui servira à effectuer les différentes actions possibles.

Nous retrouvons également trois modules liés à la décision :

— un module contenant les mécanismes de sélection, celui-ci contient les algorithmes permettant de

prendre une décision simple. C'est-à-dire faire une sélection entre plusieurs solutions selon les critères donnés et les connaissances actuelles ;

— un module d'apprentissage, qui contient les méthodes permettant d'effectuer un apprentissage qui devra être défini par le modélisateur. L'apprentissage permettant à l'agent d'acquérir des connaissances pour s'adapter à la situation. À l'heure actuelle seul l'apprentissage par renforcement a été implémenté ;

— un module d'optimisation, celui-ci comprend une bibliothèque d'optimisation externe ainsi qu'une interface permettant à un agent d'utiliser cette bibliothèque. Ainsi un agent pourra faire appel à différent processus d'optimisation pour affiner ses décisions.

L'ensemble de ces éléments ont été sélectionnés car ils composent les briques de base de nos agents cognitifs. Ces dernières sont indispensable pour représenter l'évolution et la dynamique de décision des agents dans notre application.

1.5.3 Nos contraintes

Dans le cas des deux modules les plus important, l'apprentissage et l'optimisation, nous modifions le moins possible la définition structurelle des modèles atomiques décris par DPDEMAS.

L'intérêt est de garder la structure du modèle atomique ainsi que de séparer les processus d'apprentissage et d'optimisation du comportement générale du modèle.

Bien entendu ces processus sont intrinsèquement liés à la dynamique du modèle. De ce fait, l'esprit de l'agent comme nous pouvons le voir sur la figure 1.12 garde la structure d'un modèle atomique tout en ayant la possibilité de lancer des processus externes.

Ils sont généralement utilisés comme un pré-traitement dans le but d'acquérir des connaissances qui seront nécessaires aux futures décisions prises par l'agent. De plus il doit être possible d'effectuer facilement différents types d'apprentissages ou d'optimisations dans le même contexte.

La contrainte principale est de garder la structure formelle des modèles PDEVS. Pour cela, comme nous le montrerons au chapitre suivant, les briques que nous avons ajouté sont implémentées dans une librairie de fonctions accessibles à partir du modèle. Un des éléments intéressants de la librairie est le module apprentissage par renforcement.

1.5.4 Processus d'apprentissage

D'après [Kaelbling et al., 1996] l'apprentissage consiste, pour chaque entité (agent dans notre cas) ayant à faire un choix, à interagir avec son environnement pour maximiser une récompense (reward en anglais) résultante de ses actions.

L'apprentissage par renforcement [23] (RL) est né dans les années 1960. On y retrouve des travaux dans lesquels un algorithme apprend une tache par "essai-erreur" comme dans le jeu du morpion [Michie, 1961]).

D'après [PDMIA, 2008] l'apprentissage par renforcement tel que nous le connaissons aujourd'hui a été développé depuis la fin des années 1980 avec l'ajout de la notion de processus de décision *markovien* (MDP) et de fonction d'optimalité de *Bellman* [Bellman, 1966]. C'est aujourd'hui considéré comme une branche de l'apprentissage machine.

Les processus de décision Markovien (MDP [Puterman, 2014]) sont une extension des chaînes de *Markov*. Cette extension comporte l'addition des actions choisies par l'agent et des récompenses gagnées. Il est composé de cinq éléments principaux, l'espace d'état S, l'espace d'action A, l'environnement ϵ, la fonction de récompense r et un agent de renforcement qui interagit avec l'environnement via une politique π. Le terme politique dans ce contexte ainsi que dans le reste de notre document peut-être considéré comme une stratégie à suivre, que l'agent utilise afin d'atteindre son but et/ou ses objectif(s). Plus concrètement, c'est ce qui permet à l'agent de choisir une action plutôt qu'une autre en fonction de l'état courant du système. Ainsi, le processus d'apprentissage est illustré par la figure 1.13 et se déroule de la façon suivante :

— l'agent reçoit l'état de l'environnement s_t et sélectionne une action a_t

— l'environnement répond à l'action en produisant un nouvel état s_{t+1}

— une récompense r est déterminée en fonction de s_t, a_t et s_{t+1}

— la politique π est mise à jour en fonction de la récompense générée. Le but étant de maximiser l'ensemble des récompenses sur le long terme.

23. Reinforcement Learning (RL) en angalis

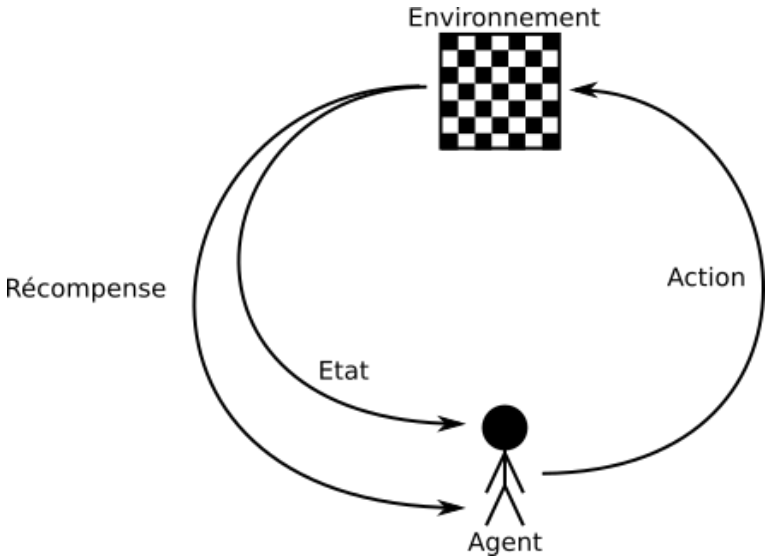


FIGURE 1.13 – Principe d'apprentissage par renforcement.

Pour mettre en œuvre cette décision, ou dans certain cas pour préciser une décision, les algorithmes d'apprentissage par renforcement se divisent en deux grandes catégories :

— Les méthodes *basées sur les valeurs*[24] qui sont basés sur une fonction de valeur mise à jour à chaque itération. Celle-ci associe une valeur à une action pour chaque agent en fonction de son état. Sur cette base une politique est améliorée à chaque itération. Nous avons principalement concentré nos travaux sur ces méthodes.

— Les méthodes *basées politiques*[25] qui se concentrent plutôt sur des méthodes de "gradient de politique"[26] comme dans [Sutton et al., 2000]. Nous pouvons y associer des approches hybrides telles que l'actor-critic présentée dans [Konda and Tsitsiklis, 2000].

Pour se rapprocher de l'une ou l'autre de ces approches en priorité nous nous sommes penchés sur ce qu'elles peuvent nous apporter.

Une synthèse est présentée dans le tableau 1.2.

En reprenant les éléments du tableau 1.2, dans un premier temps, nous avons choisi de nous concentrer sur les méthodes d'apprentissage *basées sur les valeurs* parce qu'elles permettent une exploration explicite des couples *etat/action*. Cela permet aussi une évaluation explicite de l'apport, ou de l'intérêt de chaque

24. *value-based methods*
25. *policy-based methods using policy gradient method*
26. *policy gradient method*

TABLE 1.2 – Comparaison entre les méthodes *basées valeurs* et *basées politique*.

Value-based	**Policy-based**
Exploration explicite de l'espace d'action	Exploration innée et stochasticité
Évaluation de l'apport des couples *etat/action*	Plus adapté à des espaces d'action continus
Plus adapté a un apprentissage sans politique précise ou sans	compatible avec des méthodes d'apprentissages supervisés

couple *etat/action*. En d'autres termes, le calcul d'une valeur au sens qualitatif d'une action en fonction d'un état courant.

Dans un second temps, nous avons intégré les méthodes *basées sur une politique* afin te tester et essayer d'améliorer le comportement global de l'agent, c'est-à-dire ça politique d'action. Cette seconde voie est très intéressante pour notre domaine d'application à savoir l'évaluation des politiques de pêche.

Plusieurs algorithmes permettant de mettre en œuvre ces mécanismes. Nous pouvons citer les algorithmes de "différence temporelle" (TD) qui consistent à évaluer l'état courant puis à fixer le plus précisément possible la récompense espérée à la prochaine itération (c.f. [Sutton, 1988]). La récompense obtenue ne dépendant que de l'état, cela nécessite de pouvoir déterminer à l'avance quelle action effectuer pour aboutir à l'état souhaité. On associe alors une valeur à un couple (état, action). Un premier algorithme utilisant ces couples est nommé SARSA [27] [Rummery and Niranjan, 1994].

Les informations nécessaires sont $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$. Ceci implique d'être capable de déterminer l'action a_{n+1} qui aura lieu au temps de simulation suivant. Une simplification de cela est l'algorithme de Q-Learning [28] décrit dans [Watkins and Dayan, 1992]. Il permet de ne plus avoir à déterminer l'action à l'avance mais de l'estimer. Les informations nécessaires sont alors : (s_n, a_n, r_n, s_{n+1}).

Quel que soit l'algorithme utilisé le principe d'apprentissage reste le même.

Nous pouvons voir sur la figure 1.14 les grandes lignes de l'apprentissage par renforcement.

Nous avons une simulation qui contient toute la dynamique du modèle que nous voulons mettre en œuvre. Ce modèle doit au cours de la simulation prendre un certain nombre de décisions. Pour prendre ces décisions, il dispose de connaissances.

27. Pour State–action–reward–state–action en anglais
28. Le Q-learning est une technique d'apprentissage par renforcement.

L'apprentissage à pour objectif de compléter ces connaissances pour que les décisions prises pendant la simulation soient le plus pertinentes possible. Pour cela, le processus d'apprentissage va lancer une simulations et en fonction des résultats obtenu ainsi que des décisions prises, il va déterminer à quel point les décisions prises étaient bonnes ou mauvaise et ainsi mettre à jour ses connaissances.

Pour qualifier cet ensemble d'étapes nous parlerons d'un **épisode** d'apprentissage.

Puis il va à relancer une simulation qui encore une fois prendra des décisions et générera de nouveaux résultats qui seront évalués par le processus d'apprentissage qui affinera encore les connaissances. Ce processus fait intervenir plusieurs simulations dans un grand nombre de d'épisodes d'apprentissage.

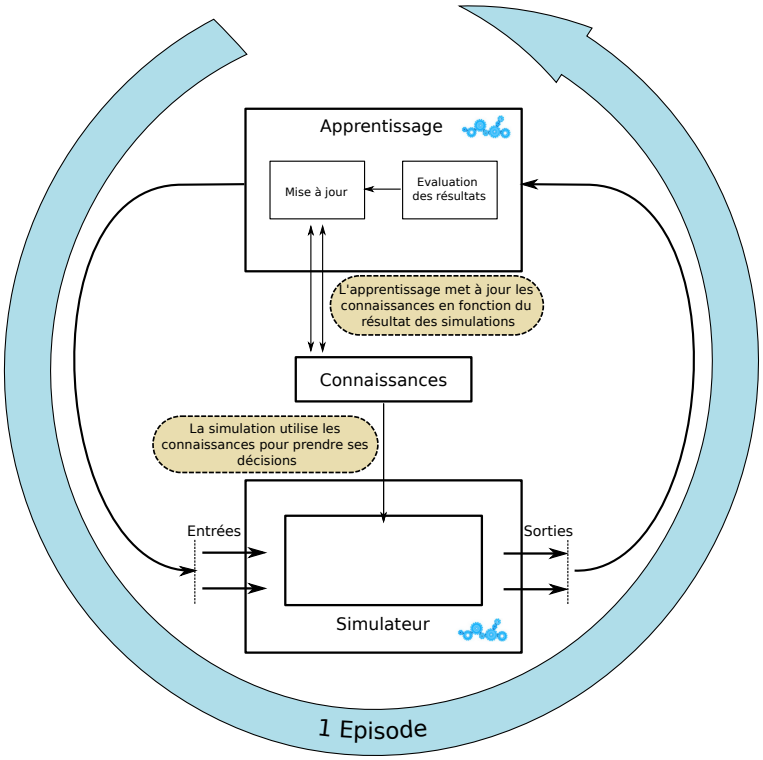


FIGURE 1.14 – Principe d'apprentissage par renforcement

À la fin de l'apprentissage, les connaissances ont évoluée et les décisions prises devraient avoir plus de pertinence par rapport aux objectifs initiaux du modèle.

Nous nous sommes intéressés dans ce paragraphe au module d'apprentissage qui est une façon de gagner de la connaissance dans une situation de prise de décision par un agent.

Nous verrons dans le chapitre 3 un exemple concret d'utilisation.

1.6 Conclusion

Nous avons introduit dans ce chapitre les formalismes et les concepts de la science de la simulation informatique que nous allons développer par la suite.

Afin de traiter un problème pluridisciplinaire tel que celui de la modélisation et la simulation d'une pêcherie Corse, nous avons proposé un processus de modélisation et de simulation en quatre étapes afin de permettre à tous les acteurs d'interagir tout au long des phases de conception, d'implémentation et de simulation.

Notre problématique lève plusieurs questions et chaque section de ce chapitre a pour objectif de présenter une partie de la solution. Tout d'abord, l'intégration des connaissances de plusieurs domaines est possible grâce à notre modèle conceptuel qui va reposer sur une approche formelle de modélisation et de simulation tel que PDEVS. Ensuite, la représentation spatiale (multi-niveau) de nos objets (les pêcheurs) et leurs interactions dans leur environnement (la mer) sont possibles grâce au paradigme agent et à l'approche multicomposant. Apres cela, l'évaluation des actions des agents (décision), l'adaptation de leur comportement à un environnement dynamique, l'amélioration de leur but est facilité grâce à l'utilisation de l'approche SOAR. Pour tout cela, la réalisation d'un modèle informatique à partir d'une sémantique non-ambiguë telle que DPDEMAS devrait aider à la reproductibilité de nos simulations Enfin, l'implémentation d'un modèle computationnel à partir de VLE est une première étape de validation de l'approche [29].

Nous avons explicité notre approche par rapport à une approche formelle tel que DPDEMAS d'un point de vue conceptuel et proposé une évolution au niveau de la description des agents.

Dans le chapitre suivant, nous allons présenter et détailler notre modèle informatique.

29. Il ne sera pas question dans nos travaux de validation des résultats du modèle par observation.

Chapitre 2

INTÉGRATION INFORMATIQUE

Conformément aux paradigmes, formalismes et concepts que nous avons retenus pour notre modèle conceptuel, nous présentons dans ce chapitre le modèle informatique générique et les composants de décision que nous mettons en œuvre dans le modèle computationnel. Nous profitons de ce travail pour intégrer le cadre formel *DPDEMAS* défini dans [Franceschini et al., 2017] en ajoutant les éléments requis par l'architecture *Soar* présentée précédemment. Rappelons que l'intérêt que nous portons à étendre le formalisme *DPDEMAS* repose sur la possibilité de pouvoir spécifier des agents cognitifs afin de pouvoir modéliser les agents économiques de la pêcherie.
Dans une première section, nous présentons les différents frameworks de simulation permettant de mettre en œuvre les extensions des formalismes que nous avons sélectionnés. Notre choix se porte sur le framework *VLE*, notamment parce qu'il est fondé sur le formalisme *DSDE*, mais également pour des raisons de performances. En outre, ce choix nous a permis d'entamer une collaboration avec les membres de l'équipe MIAT [1] de l'INRA [2] de Toulouse.

Dans une deuxième section, nous détaillons notre proposition d'intégration du modèle conceptuel générique *Multicomposant* (*MC*) dans *VLE*. Conscients que la problématique de l'échange des messages entre composants dans ce type d'architecture (approche modulaire vs approche non modulaire) est critique en termes de performance, il s'agit ici d'expérimenter les algorithmes impliqués dans la dynamique spatiale de notre modèle. Notre objectif est de valider dans ce travail préalable, la capacité du framework *VLE* à supporter l'intégration du modèle *MC* que nous proposons dans sa version générique, ainsi que sa mise en œuvre au sein du modèle exécutable qui en résulte.

Dans une troisième section, nous détaillons l'intégration des agents cognitifs et les méthodes décisionnelles correspondantes, conformément aux travaux de conceptualisation présentés dans le chapitre précédent. Nous implémentons pour cela des modules adaptés dans le modèle exécutable *MC* que nous avons intégré dans le framework *VLE*. Nous détaillons également les principaux algorithmes utilisés dans ces modules. Nous concluons ce travail d'intégration dans une quatrième et dernière section.

2.1 Frameworks DEVS/PDEVS

Dans cette section nous présentons différents frameworks permettant d'intégrer et de simuler des modèles informatiques conceptualisés selon le formalisme DEVS ou ses extensions telles que PDEVS ou

1. Unité de Mathématiques et Informatique Appliquées de Toulouse
2. Institut national de la recherche agronomique

DSDE. Il s'agit pour nous de sélectionner un framework adapté à notre approche afin d'éviter la réécriture de modules génériques DEVS connus, validés et déjà éprouvés. La mise en œuvre de nos expériences de simulation au sein de ces frameworks nécessite :

— que le formalisme DSDE soit mis en œuvre, ou puisse être intégré facilement, afin que nous puissions modéliser la dynamique structurelle de l'environnement des agents ;

— des composants de modélisation et de simulation modulaires et efficients à l'exécution, car le processus d'apprentissage nécessite de nombreuses boucles de rétroaction (feedback) ;

— des composants évolutifs, car nous fondons notre réflexion sur un processus itératif, nécessitant de fait de pouvoir modifier et faire évoluer ces derniers en fonctions des résultats que nous obtenons.

— La mise en œuvre de QSS [3] [Kofman and Junco, 2001] pour permettre de traiter les équations différentielles

La sélection du framework repose sur une étude préalable de la littérature scientifique du domaine que nous synthétisons dans le tableau ci-dessous.

Il est intéressant de remarquer que dans [Franceschini et al., 2014] les auteurs utilisent DEVStone [Wainer et al., 2011] pour tester les performances de chacun des frameworks en se fondant sur la documentation et les formalismes utilisés. Nous retrouvons aussi cette démarche dans [Van Tendeloo and Vangheluwe, 2017] où des tests de performances sont effectués à partir de différentes expériences de simulation.

Dans [Franceschini et al., 2014, Van Tendeloo and Vangheluwe, 2017] les auteurs soulignent les excellentes performances du framework VLE. Après avoir intégré un modèle multicomposant afin de vérifier cet état de fait, nous soutenons également que VLE est un outil performant. Ce travail est présenté dans [Martelloni et al., 2018] et a été effectué en collaboration avec *Gauthier Quesnel* l'un des principaux contributeurs du framework VLE. De plus, remarquons également que VLE est basé sur le formalisme DSDE et propose QSS pour la modélisation des équations différentielles.

Ainsi, eu égard aux critères d'intégration que nous avons formulés précédemment, le framework VLE apparaît clairement comme étant le candidat idoine.

Notre première contribution s'attacha à étendre le framework VLE dans le but de représenter l'environnement des agents (espace) sous forme d'un modèle cellulaire conceptualisé préalablement selon une démarche de conceptualisation multicomposant. Rappelons que ce type de conceptualisation permet

3. QSS :Quantized State Systems

Nom	**Lien / Publication**	**Formalismes**	**Avantages**	**Inconvénients**
ADEVS	lien	PDEVS DSDE	En C++ Très bonnes performances	Peu intuitif ; Réservé à des utilisateurs habitué au C++ et à la programmation.
CD++/DCD++	lien [Troccoli and Wainer, 2003]	DEVS Classique PDEVS DSDE Cell-DEVS	Adapté principalement à cell-DEVS.	
DEVS-Suite	lien	PDEVS		Performances un peu faibles ; Pas de structure dynamique.
MS4 ME	lien [Seo et al., 2013]	PDEVS	Fournis un environnement intuitif pour les non-informaticiens.	Sous licence propriétaire ; Performance assez faibles.
Quartz	lien [Foures et al., 2018]	PDEVS DSDE DPDEMAS multiPDEVS Cell-DEVS DPDEMAS	Permet de base l'utilisation de tous les formalismes qui nous interessent.	Peu utilisé dans la litérature ; Utilise un langage trop spécifique.
VLE	lien [Quesnel et al., 2009]	PDEVS DSDE Cell-DEVS	En C++ ; bonne performances ; Open source ; Proximité de l'équipe de développement ; Simulation parallelisable.	Pas de multicomposant.
PowerDEVS	lien	DEVS classique	Bonnes performances.	Uniquement DEVS classique ; Convient à des utilisateurs habitués au C++.
PythonPDEVS	lien	DEVS classique DS-DEVS PDEVS DSDE	Performances très correctes	Utilise un langage interprété.
JDEVS	lien [Filippi and Bisgambiglia, 2004]	DEVS classique Cell-DEVS DS-DEVS		Utilise un langage interprété ; Peu utilisé dans la litérature

TABLE 2.1 – Tableau récapitulatif des différents framework DEVS

sous certaines conditions, comme nous l'avons explicité précédemment, d'améliorer les performances des expériences simulations, ce qui constitue le dernier critère de sélection.

2.2 Environnement : Multicomposant dans VLE

Nous présentons dans cette section l'intégration du modèle conceptuel générique *Multicomposant* (*MC*) dans le framework *VLE*.

Dans notre approche, il est indispensable que la notion d'environnement corresponde à un composant intégrant des éléments spatiaux (cellules), et que sa dynamique soit mise en œuvre dans un espace bi-dimensionnel.

Parmi les modèles conceptuels que nous avons retenus, ceux reposant sur le formalisme Cell-DEVS décrit dans la section 1.4.2 permettent de décrire des modèles cellulaires comme des extensions des formalismes DEVS et PDEVS [Troccoli and Wainer, 2003].

En outre, ce modèle conceptuel est celui proposé par défaut pour intégrer la spatialisation dans le framework VLE. Cependant, l'intégration proposée nativement dans le framework ne permet pas le partage de données entre composants cellulaires (approche modulaire), ce qui constitue un facteur limitant en terme de performances, car la modélisation que nous envisageons nécessite de nombreuses cellules. En effet, cela implique que l'échange de données entre les cellules ne peut se faire que par l'intermédiaire des ports d'entrées-sorties conformes à ceux des modèles atomiques DEVS. Ainsi, le nombre conséquent de messages qu'implique le formalisme Cell-DEVS quand les cellules du modèle sont nombreuses pèse considérablement sur les performances de la simulation.

La figure 2.1 est une illustration de cette approche.

Afin de disposer d'une structure de données plus adéquate quant à notre problématique, qui nécessite rappelons-le, l'usage d'un nombre conséquent d'éléments spatiaux (cellules), nous faisons le choix d'utiliser le formalisme Multicomposant que nous avons présenté dans la section 1.4.2.

La figure 2.2 permet de visualiser le modèle Multicomposant envisagée.

Dans le modèle Multicomposant, à la différence du modèle Cell-DEVS, les structures modélisant l'espace n'interagissent plus par l'intermédiaire de ports d'entrées-sorties ; ses composants permettent une interaction directe avec les composants du voisinage par un système d'influence.

Ce système d'influence entre composants n'est possible qu'à l'intérieur d'un même modèle multicom-

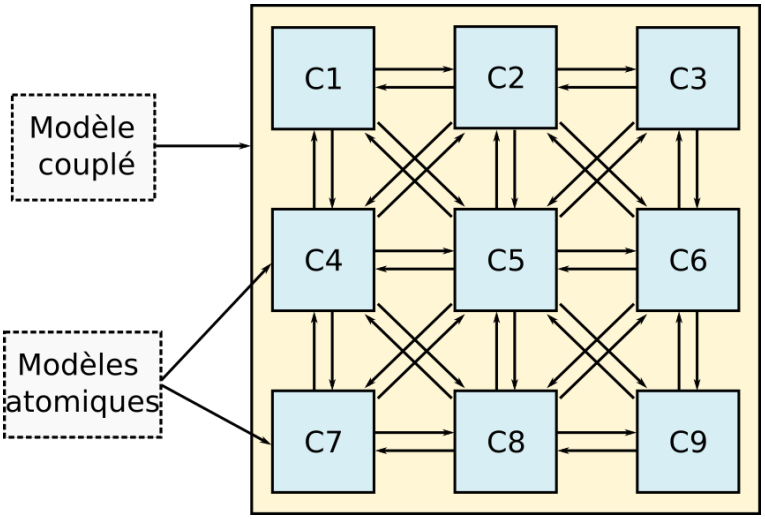


FIGURE 2.1 – Modèle PDDEVS.

posant. Mais cela permet de se passer des mécanismes de propagation par messages ce qui pourrait dans le cas où il y a beaucoup d'informations qui doivent se propager entre les cellules nous faire gagner un temps non négligeable lors des simulations.

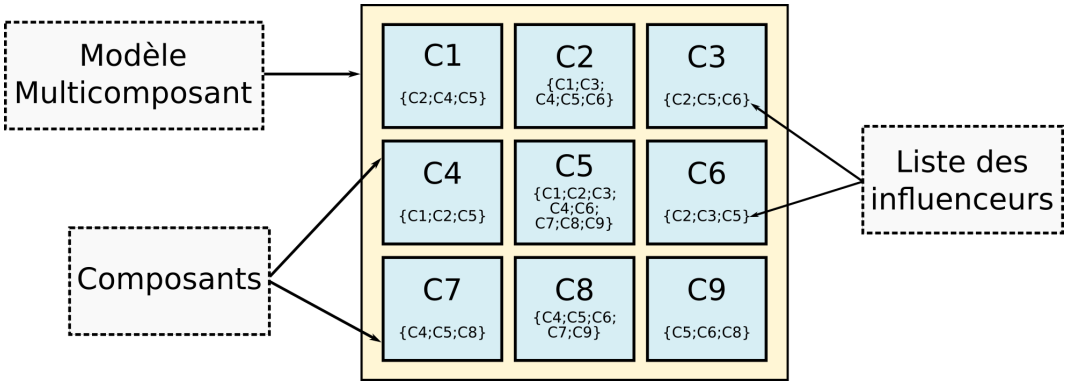


FIGURE 2.2 – Modèle multicomposant.

VLE n'intégrant pas le modèle conceptuel du Multicomposant, nous nous proposons de l'implémenter.

Nous avons présenté la structure décrivant l'environnement générique que nous avons développé afin de modéliser l'espace d'un modèle à base d'agent. Cette structure, décrite dans [Martelloni et al., 2018], a pour objectif de permettre la modélisation de l'espace à partir d'une approche cellulaire pour représenter l'environnement.

2.2.1 Description informatique de VLE

Avant de décrire les apports que nous avons proposés dans VLE, nous allons dans un premier temps nous pencher sur son fonctionnement. Il est néanmoins nécessaire de préciser que nous parlerons dans tout ce chapitre de VLE 2.0.

La figure 2.3 montre la structure informatique du coeur de VLE. Cette structure est scindée en deux grandes parties. Une partie modélisation (ou structure sur le schéma) qui sert à représenter la structure du modèle DEVS. Une partie comportementale (ou simulation sur le schéma) qui sert à décrire le comportement de chaque modèle atomique. Nous retrouvons ici une dichotomie entre structure et comportement.

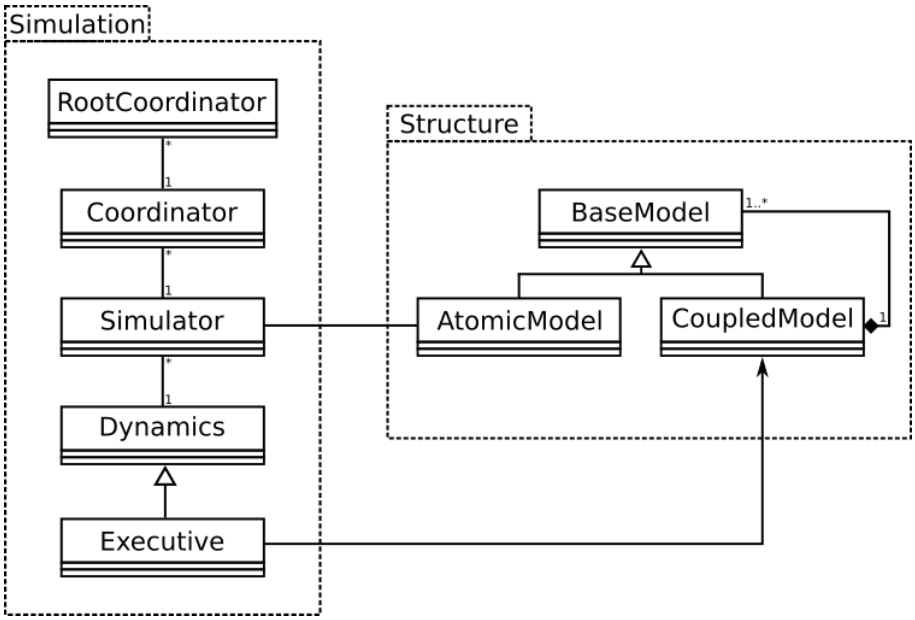


FIGURE 2.3 – Diagramme de classe schématisant le cœur de VLE 2.0

En effet, pour effectuer une simulation avec VLE, le modélisateur doit d'une part implémenter le comportement de ses différents modèles et d'autre part décrire la structure du modèle qu'il veut simuler.

Modélisation dans VLE La partie modélisation permet de décrire la structure du modèle, c'est-à-dire les différents modèles atomiques et couplés, leurs agencements ainsi que les liaisons entre eux. Cette description se fait dans un fichier XML utilisant l'extension *.vpz*. Ce fichier contient également d'autres paramètres tels que le temps complet de la simulation. Ceci peut être fait directement dans le XML ou

par l'intermédiaire de l'interface graphique de VLE. Ce fichier sera lu par un parseur et permettra de créer toute la structure du modèle. Nous retrouvons la classe *BaseModel* qui structure au niveau du code tous les modèles. Les classes *AtomicModel* et *CoupledModel* héritent de la classe *BaseModel*. Ils représentent respectivement un modèle atomique et un modèle couplé.

Ces classes décrivent une coquille vide comme une interface ou une classe abstraite, aucune description comportementale n'y est directement faite. Pour attribuer un comportement à un modèle, il faut lui lier une dynamique.

Enfin à la manière d'un pattern *factory*, nous retrouvons une classe *ModelFactory* qui permet de créer tous les éléments de modélisation. Elle va aussi permettre de créer les éléments nécessaires à la simulation ainsi que la liaison entre les parties modélisation et simulation.

Simulation dans VLE Dans cette partie, nous retrouvons les éléments de simulation, des classes coordinateur, simulateur ainsi que dynamique. Conformément aux algorithmes de simulation du formalisme DEVS, nous allons retrouver :

— La classe *RootCoordinator* qui est le coordinateur de plus haut niveau, il va gérer toute la simulation. Directement ou par l'intermédiaire de sous-coordinateurs

— La classe *Coordinator* a la charge de gérer les simulateurs et leur permettre de communiquer, c'est cette classe qui va permettre le lien entre les différents simulateurs. C'est le pendant du modèle couplé au niveau de la simulation.

— La classe *Simulateur*, a la charge d'exécuter la simulation d'un modèle atomique ainsi que d'envoyer et réceptionner des événements

— La classe *Executive* qui est une dynamique particulière permettant la mise en œuvre du formalisme DSDE. Elle va permettre de modifier la structure du modèle en cours d'exécution, de ce fait elle est liée à un modèle couplé auquel elle appartient.

— Enfin la classe *Dynamics*, elle va contenir tout le comportement d'un modèle. Il s'agit du code permettant de simuler chacune des entités du modèle. Nous y retrouvons les fonctions comportementales de base (δ, λ) d'un modèle atomique.

Nous pouvons noter que sur la figure 2.3 une dynamique (classe *Dynamics*) peut être liée à plusieurs simulateurs. Cela est dû au fait que si plusieurs modèles atomiques ont le même comportement, ceux-ci

exécuteront des instances différentes de la même dynamique.

Ainsi la classe dynamique est composée des méthodes nécessaires à l'application du formalisme PDEVS. Nous pouvons observer sur la figure 2.4 un diagramme UML de cette classe.

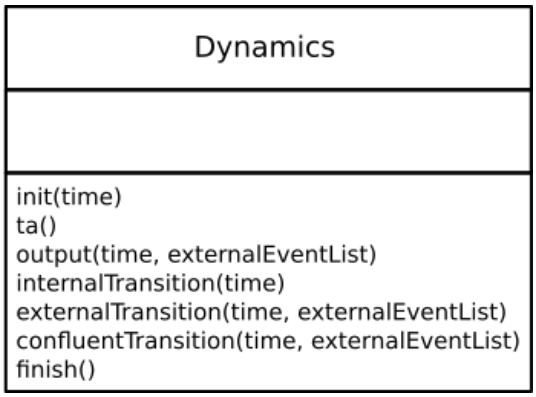


FIGURE 2.4 – Classe Dynamics de VLE

Pour ajouter dans VLE de nouveaux types de modèles avec des comportements légèrement différents, nous avons dû modifier ces mécanismes pour leur faire prendre en compte l'approche multicomposant.

C'est dans ces 2 parties qu'interviennent la majorité de nos modifications. On retrouve également le parseur ainsi qu'une classe *ModelFactory* qui permet de créer les modèles à partir du fichier XML du modélisateur et de les lier avec le simulateur.

2.2.2 Intégration du multicomposant dans VLE

Pour intégrer notre approche, nous avons ajouté dans la partie modélisation deux nouveaux types de modèles. Les modèles de type multicomposants et les modèle de type composants. Nous avons donc créé 2 classes *Component* et *MultiComponent* à l'instar des modèles atomiques, celles-ci héritent de *BaseModel* comme le montre la figure 2.5.

Ces deux classes reprennent la même structure que *AtomicModel*. Les principales différences se retrouvent dans le fait qu'un modèle multicomposant doit contenir des composants comme le ferait un modèle couplé mais qu'il n'y a pas de transmissions de messages. En effet les composants possèdent une fonction de sortie mais celle-ci est directement reliée aux ports de sortie du modèle multicomposant. Dans le cas du multicomposants, ce mécanisme ne sert qu'à interagir avec des modèles externes au multicomposant. Il en va de même avec les messages entrants qui sont collectés par le modèle multicomposant.

Chaque composant est ensuite libre d'utiliser ou non l'entrée reçue.

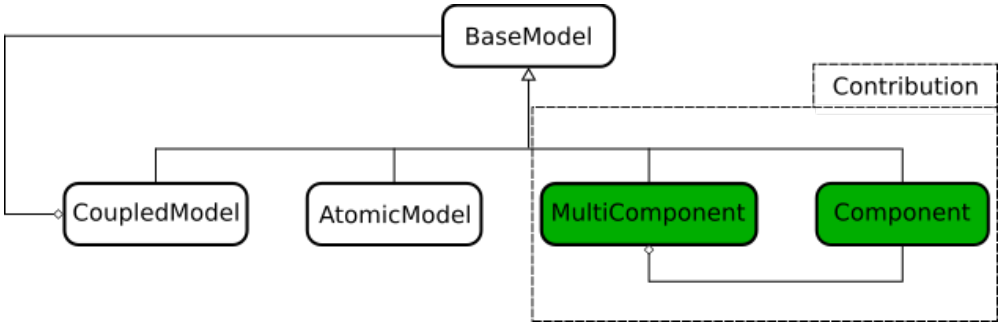


FIGURE 2.5 – Principaux éléments de modélisation.

Côté Simulation, le principe de la simulation multicomposant selon *B.P. Zeigler* nous oblige à implémenter un simulateur spécial multicomposant qui diffère un peu du simulateur PDEVS présent dans VLE.

Nous avons donc pour un souci d'évolutivité du code créé une classe abstraite que nous avons nommée *Simulator* et renommé le simulateur PDEVS *SimulatorAtomic* qui se retrouve être une classe fille de *Simulator*.

Nous avons finalement créé une classe "*SimulatorMulti*" également classe fille de "*Simulator*" et implémenté notre simulateur multicomposant.

Il mêle les rôles d'un simulateur atomique dans la mesure où il est exécuté comme un seul simulateur et d'un coordinateur dans la mesure où il gère les comportements ou dynamiques des composants qui le composent.

L'illustration de la figure 2.6 détaille ce comportement. Nous remarquons notamment sur cette figure que contrairement à un simulateur atomique qui est associé à une seule dynamique, un simulateur multicomposant est associé à un ensemble de dynamiques. En effet chaque composant ayant respectivement un comportement propre, chacun d'eux possède une dynamique. Cette dynamique de composant diffère des dynamiques des modèles atomiques. Nous en avons également ajouté un nouveau type "*multicomposantDynamics*".

Les principales différences que nous retrouvons entre "*Dynamics*" et "*multicomposantDynamics*" sont liées au principe même du multicomposant qui utilise un système d'influenceur/influencé pour faire passer des informations au lieu d'utiliser les ports d'entrées/sorties. En effet un composant comme décrit dans la section 1.4.2 comprend des mécanismes très proches d'un modèle atomique. Il est également à noter

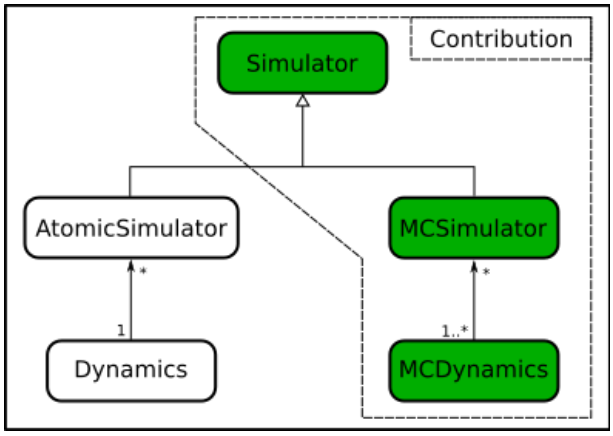


FIGURE 2.6 – Principaux éléments de simulation.

ici que la notion d'influenceur/influencé est très proche du modèle agent IRM4S (1.3.1) qui repose sur la dichotomie d'actions influence/réaction et qui est l'un des rares modèles agent permettant la prise en compte de la simultanéité dans les actions des agents.

Nous retrouvons donc dans ce nouveau type de dynamique les mêmes fonctions que dans la dynamique de base d'un modèle atomique. Les fonctions d'initialisation, d'avancement du temps, de sortie ainsi que les 3 fonctions de transitions prévues dans PDEVS, pour rappel : $\delta_{int}, \delta_{ext}, \delta_{con}$. Elles ont été décrites 1.4.1.

À côté de celà, il a bien sûr été nécessaire de modifier le coordinateur ainsi que l'échéancier pour intégrer ces nouveaux éléments. Pour assurer la bonne liaison entre la partie structure et la partie comportementale du modèle général, nous avons également dû modifier le parseur "SaxStachvpz" qui doit maintenant reconnaître de nouvelles descriptions de modèles dans le fichier vpz ainsi que ModelFactory qui devra, en plus de ce qu'elle fait déjà, permettre de créer et de relier correctement les modèles multicomposant avec l'ensemble des dynamiques des composants associés.

2.2.3 Déroulement de la simulation

Pour mieux expliquer le déroulement du processus de simulation suivant notre approche et pour bien illustrer les différences entre les deux, nous allons nous placer dans un cas particulier simple et observer deux simulateurs : l'un atomique et l'autre multicomposant dans le cas où chacun n'est associé à aucun autre modèle. C'est le cas d'un modèle qui évolue tout seul sans interaction externe. D'un point de vue

algorithmique, nous ne trouvons donc que le processus de transition interne δ_{int}.

La figure 2.7 représente ce processus. Nous rappelons que dans notre exemple il n'y a pas de couplage et donc aucun évènement externe pouvant déclencher δ_{ext} La grosse différence qui en ressort est que pour un modèle atomique PDEVS, le simulateur permet l'exécution d'une seule dynamique alors que pour le multicompsant, le simulateur doit permettre l'exécution de l'ensemble des dynamiques des composants qu'il contient et le composent.

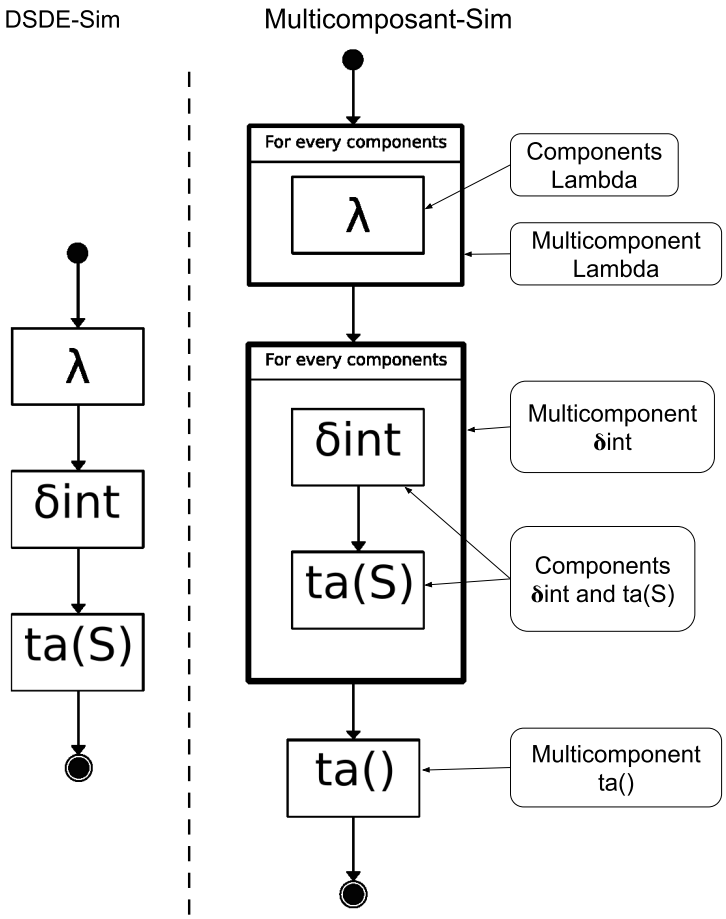


FIGURE 2.7 – Comparaison des algorithmes de simulation de notre exemple.

Modèle atomique Dans le cas d'un modèle atomique, le simulateur garde l'échéancier à jour grâce à la fonction $ta()$, et elle fixe la durée entre les transitions internes du modèle. Puis, elle fait évoluer le modèle en permettant le déclenchement puis l'exécution de la fonction de transition interne (δ_{int}) au moment prévu.

Modèle multicomposant Dans le cas d'un modèle multicomposant, l'algorithme du simulateur doit parcourir l'ensemble des composants du modèle, mettant ainsi à jour l'échéancier. Il utilise la fonction de transition interne du multicomposant. Comme expliqué dans la figure 2.7, elle va permettre d'exécuter la fonction de transition interne de tous les composants imminents à chaque événement (pas de temps).

Toutes les interactions possibles entre composant étant gérées grâce aux effets d'influences, elles sont directement prises en compte par la fonction de transition interne. Il faut donc particulièrement se pencher sur la manière d'influencer un composant à partir d'un autre surtout quand nous voulons modéliser des effets de diffusion où il peut y avoir des temps ou des ordres particuliers de propagation d'une information.

En sortant de notre cas particulier, l'interaction d'un modèle multicomposant avec d'autres modèles se fait de la même manière qu'entre tout modèle PDEVS par l'intermédiaire d'événements d'entrées et grâce à la fonction de sortie λ.

2.2.4 Exemple de mise en œuvre

Afin de montrer l'intérêt en termes de performance de l'approche, nous présentons un modèle simple qui décrit une grille carrée avec une information se propageant du centre vers les bords. L'objectif est de comparer les temps de calcul entre une modélisation cellulaire à partir de Cell-DEVS et une modélisation à base de composants.

Nous avons comparé trois expériences de simulations à partir de VLE : (1) une grille modélisée avec l'approche cellulaire à partir de modèles Cell-DEVS dans VLE classique, c'est-à-dire avant modification. Ce cas d'étude doit nous servir de simulation étalon. (2) une grille modélisée avec l'approche cellulaire à partir de modèles Cell-DEVS dans VLE après l'ajout dans la hiérarchie de classe de notre approche Multi-composant. Et enfin, (3) Une grille constituée de composants à partir de notre approche Multicomposant. Ceci respectant le schéma des figures 2.1 et 2.2.

Le tableau 2.2 présente la comparaison des temps de calcul entre les différents cas.

Nous observons une différence non négligeable des temps de calcul entre les deux approches. Quand

TABLE 2.2 – Comparaison des temps moyen de calcul en secondes.

Taille de la grille	Cell-DEVS Original	Cell-DEVS	Multicomposant
100×100	10.13 s	13.00 s	1.386 s
200×200	59.08 s	62.76 s	13.10 s
300×300	187.60 s	194.60 s	31.33 s
400×400	405.33 s	426.66 s	80.65 s

la taille de la grille augmente, le temps de calcul augmente très fortement dans le cas d'une approche modulaire. Ce phénomène est fortement atténué dans le cas d'un multicomposant non modulaire.

Néanmoins on observe une augmentation du temps de calcul entre la même approche avant et après nos modifications, ceci s'explique par le fait que nous avons alourdi l'architecture de l'api en ajoutant plusieurs éléments d'héritage au niveau du coeur de simulation. Cette approche et les modèles auxquels cette approche a été appliquée ont fait l'objet d'une publication [Martelloni et al., 2018] dans laquelle le lecteur trouvera plus de détails. Nous avons par la suite utilisé cette approche pour plusieurs modèles.

Pour conclure, nous avons présenté et intégré dans VLE une approche de modélisation Multicomposant. Nous proposons une rapide comparaison de performances à partir d'un exemple de propagation d'un message dans une grille. VLE ne disposant pas nativement de l'extension multicomposant, nous l'avons donc ajoutée. L'environnement VLE a encore évolué suite à ce premier travail et il intègre maintenant le concept de multicomposant. L'architecture logicielle utilisée a été améliorée depuis notre première proposition pour pallier la baisse de performance observée en réduisant notamment les effets de couplage et d'héritage que nous avions proposé.

Ce premier travail a été fondateur. Il nous a permis de maîtriser VLE, c'était l'un des objectifs. Il nous a également permis de déployer l'approche multicomposant que nous avions ciblé pour représenter l'environnement de notre SMA. Nous pouvons également noter que les gains en termes de temps de simulation nous serons très utile par la suite. En effet nous allons maintenant nous intéresser à l'aspect cognitif des agents qui dans le cadre de l'apprentissage (cf section 1.5.4) par exemple demande d"effectuer un grand nombre d'épisodes de simulation. Nous allons maintenant présenter nos travaux sur la composante agent.

2.3 Agent : approche cognitive dans VLE

Nous allons à présent revenir sur la partie décisionnelle. Comme nous l'avons déjà indiqué, pour faciliter la modélisation des processus de décision des pêcheurs, nous avons décidé d'utiliser certaines

briques de l'architecture *Soar*. Elles permettent de décrire dans un agent les processus de décision et donc son système cognitif. Nous présentons ici la structure décrivant l'agent cognitif générique que nous avons développé afin d'utiliser plusieurs méthodes de prise de décision. L'ensemble de ces éléments est concentré dans l'esprit du modèle agent pour respecter la dichotomie corps/esprit.

2.3.1 Schéma structurel

La description d'un modèle conceptuel au chapitre précèdent était une première étape, nous proposons ici une architecture informatique afin d'intégrer un processus de décision dans nos modèles VLE.

Sur la figure 2.8, nous décrivons l'architecture de classe de notre approche.

Elle se compose d'une première brique inspirée de DPDEMAS [Franceschini, 2017].

Nous pouvons distinguer les classes :

— *Model* qui décrit un modèle PDEVS ;

— *ModelAtomique* et *ModelCouplé* qui décrivent respectivement le comportement des modèles atomiques et couplés selon le formalisme PDEVS ;

— *Multicomposant* qui permet l'utilisation du formalisme multicomposant ;

— et enfin, *DSDE* qui permet de définir des modèles à structure dynamique.

— La classe *Executive* décrit le *ModelAtomique* qui gère le graphe de modèles dans DSDE

Dans cette architecture, nous schématisons notre apport à partir des classes du package encadré de rouge. Nous avons rajouté le système cognitif de l'agent.

Dans notre approche, un agent est défini comme étant un modèle couplé PDEVS contenant plusieurs modèles atomiques afin de représenter son esprit et ses corps (un corps par environnement).

C'est dans l'esprit que réside tout le système cognitif que nous proposons.

Dans notre *système cognitif*, nous retrouvons tout ce qui est lié à la méthode de prise de décision, et notamment :

— un module de décision qui contient les algorithmes permettant de prendre une décision simple. C'est-à-dire faire un choix (une sélection) entre plusieurs solutions selon les critères donnés et les connaissances actuelles. Cela peut être sous la forme d'une méthode qui sélectionne une action dans une liste en fonction d'une valeur précédemment estimée de cette action comme la méthode *greedy*

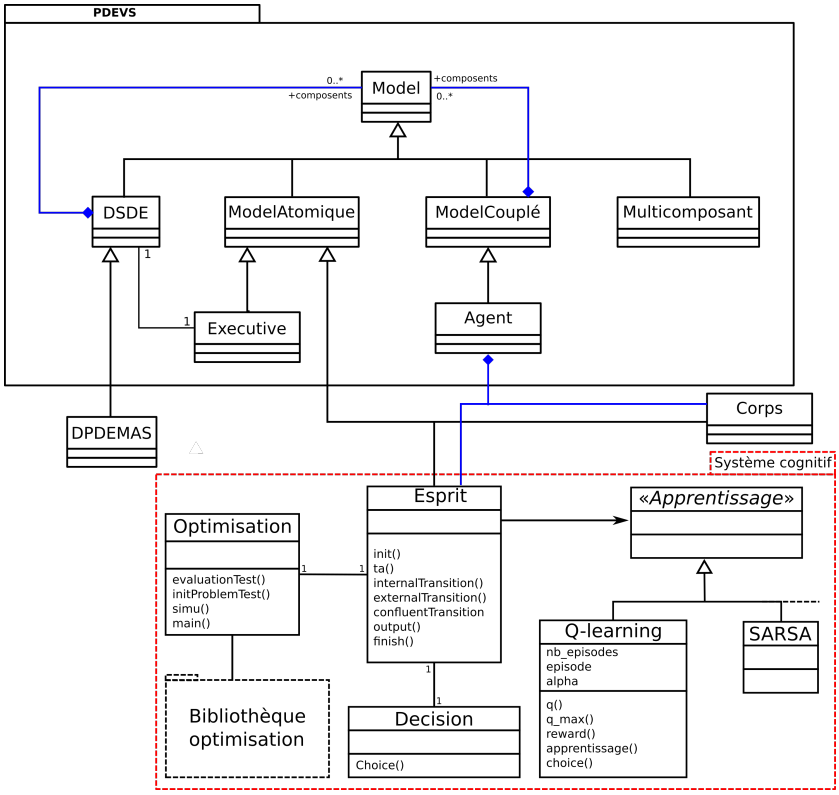


FIGURE 2.8 – schémas UML utilisé par l'approche proposée extrait de [Franceschini, 2017]

[Sutton and Barto, 2018] dont nous reparlerons dans la section 2.3.2. Cela peut également être sous la forme d'une méthode *MinMax* ou *AlphaBeta* qui sont des algorithmes depuis longtemps utilisé dans le domaine des jeux (tictactoe, échec, etc.) [McAllester, 1985],[Rivest, 1987] bien que depuis ils aient été souvent remplacés ;

— un module d'apprentissage dont le processus doit être défini par le modélisateur. À l'heure actuelle seul l'apprentissage par renforcement a été implémenté, nous y reviendrons. Nous retrouvons sur la droite de la figure une classe abstraite *Apprentissage* dont peuvent hériter plusieurs types particuliers d'apprentissages. Celui que nous avons implémenté actuellement étant le Q-Learning [4], nous retrouvons la classe correspondante. Cette architecture nous permettra d'ajouter ultérieurement d'autres types d'approches. Les connaissances acquises pendant le processus d'apprentissage permettent à l'agent de s'adapter en fonction de la situation.

— Un module d'optimisation qui comprend une bibliothèque d'optimisation externe ainsi qu'une in-

4. Q : pour qualité car l'on cherche la meilleure action à entreprendre

terface permettant d'utiliser cette bibliothèque a partir d'un agent. Ainsi sur le schéma, notre classe *Esprit* est liée à une classe statique *Optimisation* qui nous sert d'interface avec la bibliothèque d'optimisation. Cette classe statique contient les méthodes nécessaires pour définir une optimisation, la paramétrer et lancer son exécution. Le principal objectif de ce module est de fournir une capacité à calculer une tendance optimale, une direction vers laquelle se diriger. C'est du moins la façon dont nous l'utiliserons par la suite. C'est également un constat qui est ressorti de nos premières utilisations de ces types de méthodes [Martelloni et al., 2019].

Nous allons maintenant nous pencher plus profondément sur le fonctionnement de ces différents modules.

2.3.2 Nos modules de décision

Nous pouvons résumer les processus de décision simple par le principe suivant : faire un ou des choix d'action entre différentes options connues à partir de connaissances sur soi même et son environnement.

Dans notre cas, l'agent a une connaissance des possibilités qui lui sont offertes ainsi que de la situation vers laquelle il souhaite se diriger. Il existe de nombreux cas de figures possibles de prise de décision simple à partir de données connues. La principale utilisation que nous en faisons actuellement dans nos simulations sera d'utiliser les données acquises au cours d'un apprentissage. Il s'agira dans ce cas de choisir à partir d'une table de connaissance, l'action qui sera associée à la situation courante et apportera selon cette table le meilleur résultat. Nous reviendrons plus en détail sur ce processus dans les exemples que nous allons présenter dans le chapitre 3.

Apprentissage

Notre module d'apprentissage comprend pour l'heure uniquement des méthodes d'apprentissage par renforcement basées sur le Q-learning.

Comme nous l'avons vu dans le chapitre précédent (cf.1.5.4), nous rappelons que le Q-learning est un algorithme d'apprentissage par renforcement qui cherche à trouver la meilleure action à entreprendre en fonction de l'état du système. C'est donc particulièrement indiqué pour répondre à nos objectifs.

Pour cet algorithme, nous avons besoin de connaître à chaque étape d'apprentissage les éléments suivants : (s_n, a_n, r_n, s_{n+1}) où,

— s_n est l'état courant, donc l'état de l'ensemble des variables d'état ;

— $a_n \in \mathbb{N}$ est l'action à effectuer ;

— r_n est la récompense engendrée par l'action choisie ;

— s_{n+1} est l'état résultant de l'action choisie.

Comme nous l'avons indiqué dans la section 1.5.4, l'algorithme de *Q-learning* est plus simple que *SARSA* et plus complet que *TD*. Il s'impose donc comme le premier algorithme à tester.

L'implémentation d'autres méthodes comme celles de *programmation dynamique*[5] (DP) [Munos, 2000], celles de Monte Carlo [Kalos and Whitlock, 2009] ou encore celle de *gradient de politique*[6] [Sutton et al., 1999] qui font partie des méthodes *basées politiques* fait partie de nos perspectives. Nous souhaitons également, dans un prochain travail, comparer des résultats obtenus et les performances de chacune de ces différentes méthodes.

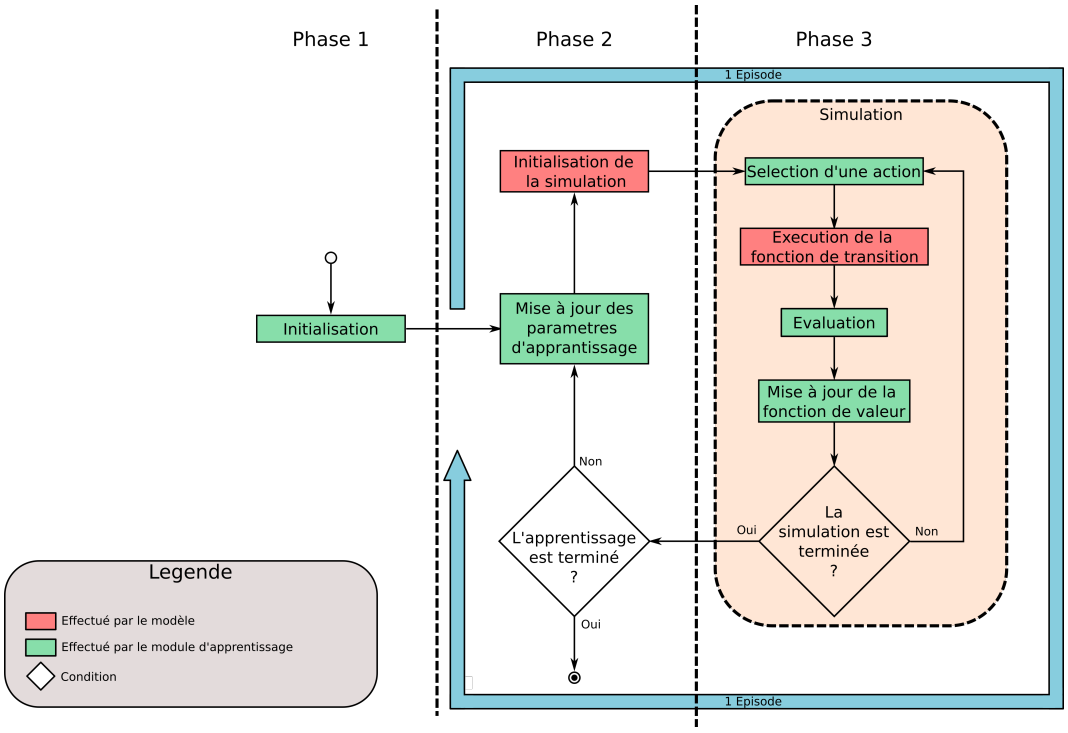


FIGURE 2.9 – Algorithme d'apprentissage

Dans notre cas, le processus d'apprentissage consiste à essayer des actions et en observer les conséquences.

5. dynamic programming
6. Policy gradient

Comme le montre la figure 2.9, l'apprentissage se déroule en plusieurs étapes.

Une phase 1 d'**initialisation** de l'apprentissage permet de définir et de mettre en place le système. Ensuite vient le déroulement de l'apprentissage en lui-même, phases 2 et 3. Celui-ci comprend de nombreux épisodes complets de simulation du modèle, cela se déroule ainsi : (2) en début de cycle, nous fixons les paramètres d'apprentissage α et ϵ. (3) Puis on lance la simulation du modèle.

À chaque pas de simulation, l'algorithme va suivre le processus suivant :

La **sélection** (phase 3.1) consiste à activer le module d'apprentissage pour choisir l'action à effectuer. L'étape de sélection au cours de l'apprentissage est très importante. Elle dépend de ϵ et se déroule en deux temps :

1. (a) l'exploration, qui consiste à essayer des actions dont les conséquences sont encore peu estimées afin d'en augmenter les connaissances. Ce processus se fait souvent de façon aléatoire ;
2. (b) l'exploitation, qui consiste à exécuter l'action que nous connaissons pour avoir les conséquences les plus favorables. Cela permet d'affiner les connaissances déjà acquises.

Il existe de nombreuses méthodes de sélection, dans leur ouvrage [Sutton and Barto, 2018] en présentent plusieurs pour l'apprentissage par renforcement. La méthode *greedy* [Sutton and Barto, 2018], consiste à sélectionner la meilleure action connue possible. Elle peut être définie par :

$$A_t = Arg \max_a Q_t(a)$$

C'est une méthode de sélection qui convient parfaitement pour une décision simple, une fois l'apprentissage terminé. Néanmoins, pour la prise de décision pendant l'apprentissage, elle n'est pas tout à fait adaptée. Elle ne permet pas l'exploration, pour remédier à cela, il existe une variante ou plutôt une généralisation $\epsilon - greedy$ [Sutton and Barto, 2018], qui consiste à rajouter à la méthode *greedy* un taux d'exploration $1 > \epsilon > 0$.

L'algorithme aura donc une probabilité ϵ de ne plus choisir la solution optimale connue, mais de choisir une action au hasard. C'est donc cette solution ($\epsilon - greedy$) que nous avons retenue pour notre implémentation.

Ces deux phases ne sont pas indépendantes, un équilibre entre les deux est nécessaire. Pour cela, nous utilisons ϵ, un des paramètres d'apprentissages qui évoluent au cours du processus.

Nous retrouvons dans la littérature plusieurs variantes quant à ce paramètre ϵ [Raykar and Agrawal, 2014], [Strehl and Littman, 2004], [Watkins, 1989]. La première propose de fixer ce paramètre le plus souvent entre 1% et 10% d'exploration ($0.01 \leq \epsilon \leq 0.1$). Une valeur trop élevée fixe nous donnerait trop d'aléatoire et une trop faible, risque de se fixer plus facilement sur un optimum local.

Selon [Sutton and Barto, 2018] un taux plus élevé permet une exploration plus rapide mais peut avoir un résultat moins efficient (si certaines actions ont des valeurs très proches en particulier), et un taux plus faible sera plus lent à explorer en donnant des résultats plus efficients mais augmentera également le risque de tomber sur des optimums locaux.

Une solution pour allier exploration et exploitation est de faire décroître le paramètre ϵ au fur et à mesure de l'apprentissage. C'est la solution que nous avons choisi d'implémenter. En début d'apprentissage, nous privilégions **la phase d'exploration** pour tester le plus de solutions possibles. En fin d'apprentissage, par contre, c'est **la phase d'exploitation** qui est privilégiée afin de renforcer les connaissances acquises.

Passer l'étape de **sélection**, il est nécessaire d'**activer le modèle** pour exécuter son comportement grâce aux fonctions de transitions. Celles-ci vont déterminer le nouvel état en fonction de l'action choisie. Nous pouvons nommer cette étape : **l'étape comportementale 3.2**.

Vient ensuite l'étape d'**évaluation** pour activer le module d'apprentissage afin d'évaluer le nouvel état du modèle et l'action choisie, étape 3.3. Cela permet de déterminer la récompense. Le module d'apprentissage met à jour *la fonction de valeur*, étape 3.4.

Dans le cas du Q-Learning, cette fonction prend la forme de l'équation suivante :

$$Q : S \times A \rightarrow R$$

avec :

$$Q(s_n, a_n) = Q(s_n, a_n) + \alpha(r + \max_a Q(s_{n+1}) - Q(s_n, a_n)) \tag{2.1}$$

La fonction de valeur associe une valeur au couple $(etat, action)$ et fixe la nouvelle action la plus

adaptée en fonction de l'état. Le processus est ainsi répété.

Une fois que la simulation est arrivée à son terme, on passe au cycle suivant en bouclant sur l'étape de mise à jour des paramètres d'apprentissage. Enfin si l'ensemble des cycles a été exécuté l'algorithme sauvegarde la fonction de valeur puis se termine. *La fonction de valeur* pourra alors servir de base de connaissance pour effectuer des choix dans le futur.

Nous nous sommes intéressés dans ce paragraphe au module et processus d'apprentissage. C'est un moyen très intéressant pour que nos agents gagnent en autonomie. Ils sont ainsi en capacité d'accroître leurs connaissances et donc de prendre des décisions adaptées à leur situation et leur objectif.

Notre second module offre la possibilité d'utiliser des méthodes d'optimisation. Ces méthodes permettent d'obtenir des tendances à suivre pour atteindre les objectifs des agents. Avec ces méthodes, nous n'allons pas directement fournir la capacité à sélectionner une action mais plutôt donner une direction à suivre. L'approche est très complémentaire de l'apprentissage par renforcement. En fonction du contexte, il sera possible de basculer d'une méthode à l'autre. Il sera également très intéressant de pouvoir coupler les approches afin d'utiliser l'optimisation pour améliorer l'apprentissage et par exemple affiner la *fonction de valeur*.

Nous présenterons un exemple dans le chapitre suivant où un agent effectue un apprentissage en ayant connaissance des tendances à suivre. Ces dernières sont obtenues par optimisation. Cela donne une connaissance supplémentaire et permet d'orienter l'apprentissage.

L'utilisation de l'apprentissage par renforcement connaît également des limites pratiques. Un espace *etat/action* trop vaste oblige à fortement augmenter le nombre d'épisodes d'apprentissage et donc le temps d'apprentissage. En effet si une partie de l'espace n'a pas été explorée, une solution plus intéressante peut rester cachée. Dans ce type de situation, d'autres méthodes telle que l'optimisation peuvent remplacer ou aider l'apprentissage.

Optimisation

Le module d'optimisation est issu des travaux d'optimisation par simulation (OvS) présentés dans [Poiron-Guidoni, 2018]. Ces travaux sont développés en parallèle dans le cadre de la thèse de *Nicolas*

Poiron-Guidoni. Le principe de ces optimisations est illustré dans la figure 2.10. Comme nous pouvons le voir, nous avons un modèle comportant un ensemble de paramètres dont nous voulons trouver une valeur viable selon un certain nombre de critères qui dépendent des cas d'utilisations et des données dont nous disposons.

Le but est de maximiser ou minimiser une ou plusieurs fonction(s) objectif(s). Les méthodes d'optimisation peuvent être déterministes ou stochastiques. Lorsque la complexité du problème devient trop importante, telle que l'optimisation via simulation de modèle biologique, on privilégiera des méthodes stochastiques, approchées telles que les métaheuristiques [Siarry, 2014]. Celles-ci permettent d'obtenir des solutions acceptables en temps limité à la différence des méthodes déterministes qui demandent souvent un grand temps de calcul. Elles ne peuvent cependant pas assurer l'optimalité de la solution proposée.

De façon générale, nous pouvons définir l'optimisation comme un processus cherchant à déterminer le meilleur ensemble de paramètres possible pour un problème et des critères donnés. Pour cela nous prenons notre modèle, l'algorithme d'optimisation va fixer ses paramètres et lancer la simulation. A la fin de la simulation, les résultats sont évalués au regard des données connues et aux critères que nous souhaitons remplir. Si les résultats sont conformes, le jeu de paramètre est sauvegardé, sinon l'algorithme d'optimisation met à jour les paramètres du modèle et relance une nouvelle simulation.

Ce principe peut être exécuté plusieurs fois ou utiliser des méthodes d'optimisation multimodales pour obtenir un ensemble de jeux de paramètres viables. Nous parlons de multimodalité pour des méthodes capables de traiter des situations où il existe des cas ou l'optimisation peut converger vers plusieurs optimums locaux. Dans la mesure où pour de nombreux systèmes nous n'avons pas de connaissances suffisantes pour limiter la solution à un seul jeu de paramètres.

En conclusion, notre approche permet d'une part de développer ses propres méthodes de décision. Ce que nous avons fait avec le Q-learning et d'autre part d'intégrer des méthodes déjà développées ce qui était le cas des méthodes d'optimisation que nous avons intégré à partir d'une bibliothèque existante.

Ceci permet de renforcer la généricité de l'approche.

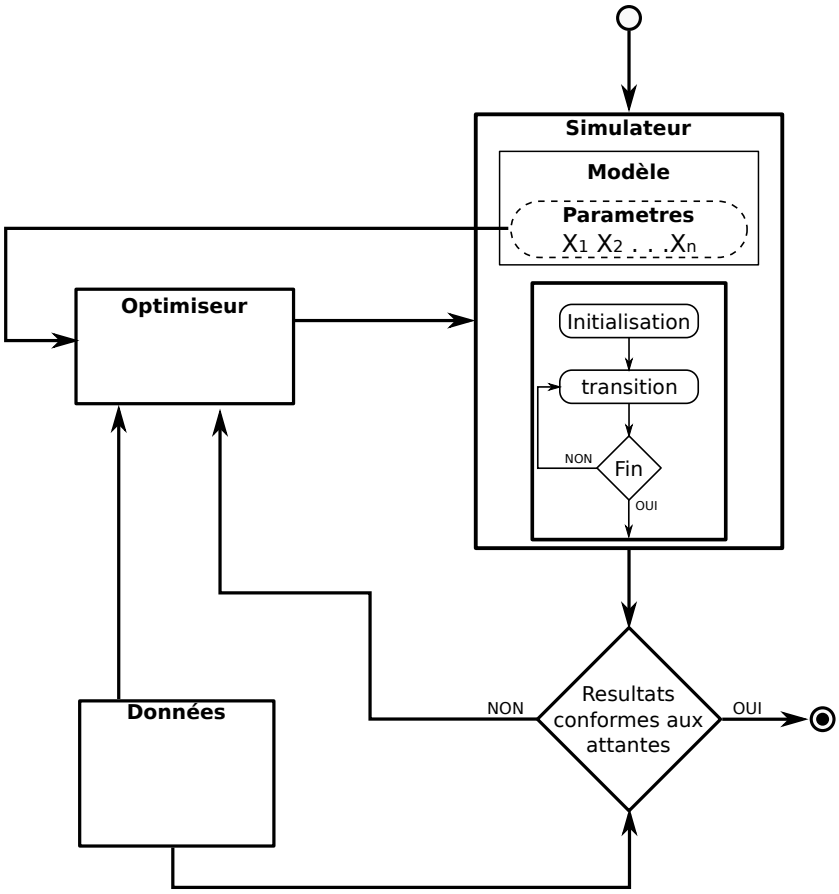


FIGURE 2.10 – Processus d'optimisation

2.3.3 Intégration de notre approche dans VLE

Dans la section précédente, nous avons exposé les briques ainsi que la structure que nous proposons pour décrire un agent cognitif générique, le rendant ainsi capable d'utiliser plusieurs méthodes de décision que nous avons détaillé. Nous allons à présent montrer comment nous avons assemblé tout cela, comment cela fonctionne et comment cela peut être utilisé dans VLE

Dans VLE, le modélisateur souhaitant utiliser notre approche aura accès à des classes et des méthodes facilitant sa prise en main. Sur la base de ce code, il devra dans un premier temps réaliser son modèle DEVS c'est-à-dire implémenter les fonctions comportementales. Puis, il devra utiliser la classe *Esprit* (cf figure 2.8) pour lier son modèle DEVS à nos modules et compléter les méthodes telles que $q()$, $qmax()$, $choix()$, $recompense()$, ou $apprentissage()$ dans le cas du Q-learning pour définir les paramètres des

épisodes de simulation. Le modélisateur a également la possibilité de développer ses propres méthodes d'apprentissage et de les lier à notre approche en les faisant dériver de la classe abstraite *Apprentissage*.

Pour les mécanismes que nous souhaitons compléter, nous avons rajouté quelques méthodes qui feront elles-mêmes appel à des bibliothèques externes. Nous avons choisi d'utiliser des modules externes pour ne pas surcharger la définition d'un modèle atomique avec des méthodes indépendantes du formalisme DEVS. De plus ces modules pourront par la suite être complétés ou améliorés sans nécessiter de modifications au sein de la classe *Modele*.

Application d'une méthode d'apprentissage par renforcement : Q-learning

Nous avons déjà décrit comment le module d'apprentissage permettait à un agent d'apprendre et de mettre à jour les données de la mémoire pour prendre de bonnes décisions.

Pour effectuer un processus d'apprentissage, un modèle doit posséder une variable membre pour stocker un objet de type apprentissage. Cet objet est de type *abstrait*, il permet d'utiliser différentes méthodes d'apprentissages. Elles devront hériter de la classe abstraite *Apprentissage*.

De la même manière d'autres types d'apprentissages tels que SARSA ou TD (cf. section 1.5.4) sont intégrables et utilisables.

Comme nous pouvons le voir sur la figure 2.8 nous retrouvons différents éléments :

— **table** : la table sauvegardée qui permet de garder en mémoire ce que le modèle apprend. Il s'agit des résultats de la fonction de valeur ; La table associe un couple *etat/action* du modèle avec une valeur représentative de l'utilité [7] de cette action quand le modèle se trouve dans l'état donné ;

— **nb_episodes** : le nombre d'épisodes total d'apprentissage. Comme nous l'avons vu dans le chapitre précédent, (1.5.4, un épisode est une simulation complète du début à la fin. Dans le cas d'un jeu par exemple, il s'agit d'une partie complète. Ainsi dans ce cas là *nb_episodes* sera le nombre de parties à jouer pendant l'apprentissage ;

— **épisode** : le nombre d'épisodes effectués depuis le début de l'entraînement ;

— enfin, les paramètres d'apprentissage tel que α ou γ dans le cas du Q-learning (voir équation 2.1) qui serviront à mettre à jour la fonction de valeur et la table associée.

Nous pouvons également identifier les méthodes :

7. L'utilité représente ici à quel point une action satisfait les objectifs, ou à quel point la récompense engendrée est élevée

— **récompense()** : une méthode qui permet de calculer la récompense/pénalité d'une action. C'est une fonction très importante dans le fonctionnement de l'apprentissage. Celle-ci doit être capable de juger du résultat d'une action ou d'un ensemble d'actions. Cette fonction est très dépendante des objectifs du modèle. Elle doit prendre en paramètre le ou les couples état/action exécutés ainsi que les informations nécessaires sur le résultat.

— **choix()** : une méthode permettant de choisir une action à effectuer en fonction des paramètres d'apprentissage et permettant de retrouver le mécanisme d'*exploration/exploitation*. Durant l'apprentissage le processus de choix de l'action à effectuer est différent de celui fait par un agent lors d'une simple simulation. En effet celui-ci n'est pas encore sûr de ses choix. Il faut donc faire des essais pour déterminer quelles actions sont les plus adaptées. Comme nous l'avons souligné précédemment, il faut d'une part essayer des actions que nous n'avons pas encore testées et d'autre part vérifier les connaissances déjà acquises. Pour cela suivant l'un des paramètres d'apprentissage, il y aura une probabilité ϵ d'effectuer une des actions possibles aléatoirement et la probabilité complémentaire $1-\epsilon$ de choisir la meilleure action connue pour l'état courant. Nous pouvons voir dans l'algorithme 2 le déroulement de cette fonction. La tâche *trouverchoixpossibles*() à la ligne 4 est dépendante du modèle et retourne une liste correspondant à tous les choix possibles pour l'agent. Le résultat de cette tâche est affectée à la variable *actionsPossibles*. Enfin, *randdom*(*actionsPossibles*) à la ligne 13 retourne aléatoirement un élément de la liste *actionsPossibles* ;

— $q()$ **et** $q\_max()$ des méthodes de mise à jour de la fonction de valeur. Elles dépendent grandement du type d'apprentissage que nous voulons effectuer. Dans le cas du Q-learning que nous avons développé sur la figure 2.8, elles utilisent la formule 2.1. Nous aurions pu faire cela en utilisant une seule fonction, mais pour des raisons de clarté de code nous avons choisi de séparer les deux parties. Il s'agit du processus qui va permettre de mettre à jour la table de valeurs. En fonction du type d'apprentissage développé, elle aura besoin de différents paramètres présentés dans la section 2.3.2. C'est-à-dire $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$ pour l'apprentissage SARSA ou (s_n, a_n, r_n, s_{n+1}) dans le cas du Q-learning. À ces paramètres s'ajoutent les paramètres d'apprentissage α et γ dans l'équation 2.1.

— **apprentissage()** : méthode qui va gérer les paramètres et les étapes de l'apprentissage. Elle est intrinsèquement liée au modèle développé. Les étapes à effectuer restent identiques, en revanche elles ne se passent pas toujours au même moment suivant le système modélisé. En effet, certaines

simulations doivent se terminer avant de pouvoir juger des résultats, tel que l'exemple que nous exposerons par la suite (cf. section 2.3.4), d'autres peuvent juger du résultat d'une action une fois celle-ci effectuée. De ce fait il faudra effectuer l'étape de calcul de la récompense à un instant différent. L'algorithme 1 montre un déroulement possible de cette méthode.

Toutes les méthodes listées ci-dessus sont dépendantes du type d'apprentissage ainsi que du modèle. Nous fournirons donc un prototype et une base qui peut être commune à un type d'apprentissage, mais il sera nécessaire de les adapter aux besoins.

Algorithm 1 Fonction apprentissage

```
1: function apprentissage()
2: booleen finEntrainnement ← vrai
3: episode ← episode + 1
4: reward()
5: if finEntrainnement then
6:    saveTable()
7: end if
8: return finEntrainnement
```

Algorithm 2 Fonction choix

```
1: function choix(état)
2: list actionsPossibles
3: resultat ← 0
4: actionsPossibles ← trouverchoixpossibles()
5: Booleen test ← random()
6: if ϵ > test then
7:    for all i ∈ actionsPossibles do
8:       if table[etat][resultat] ≤ table[etat][i] then
9:          resultat ← i
10:      end if
11:   end for
12: else
13:   resultat ← random(actionsPossibles)
14: end if
```

La figure 2.11 montre un diagramme de séquence d'un apprentissage comme nous l'avons implémenté. Nous y retrouvons donc toutes les phases que nous avons précédemment évoquées ainsi que les appels des différentes fonctions que nous venons de décrire. Ceci illustre le processus d'apprentissage et nous voyons bien que les simulations effectuées lors d'un apprentissage sont faites par l'agent.

Nous avons ici proposé une méthode d'apprentissage que nous avons développé. Notre architecture

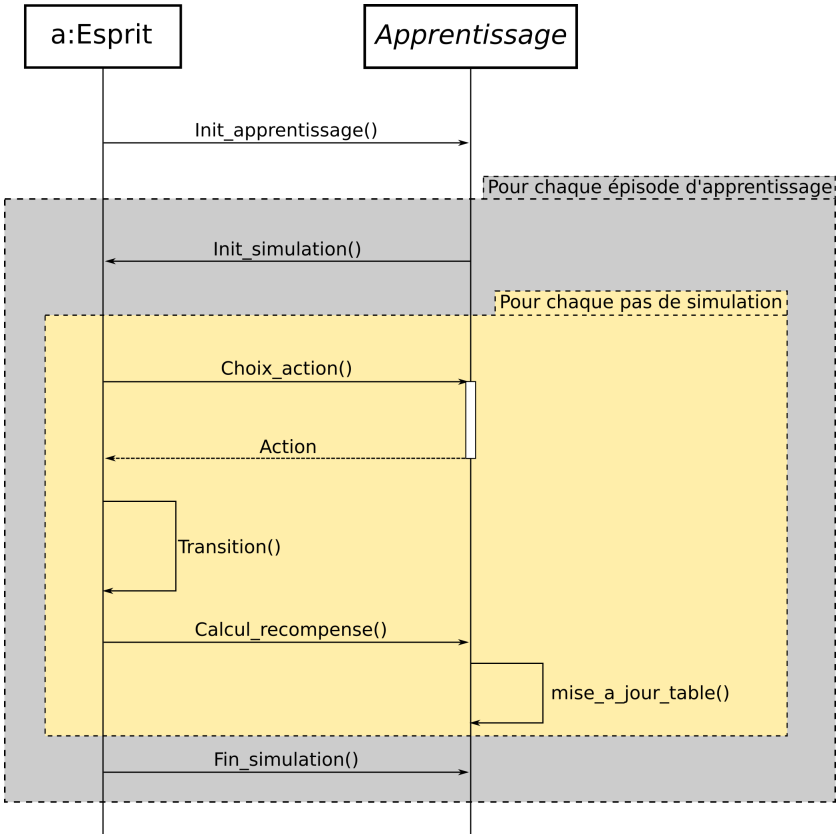


FIGURE 2.11 – Diagramme de séquences du processus d'apprentissage.

logicielle permet d'une part de facilement utiliser une des méthodes déjà développées et d'autre part d'en intégrer de nouvelles.

Nous allons maintenant nous pencher sur le deuxième type de méthode que nous avons abordé et qui est elle issue de l'intégration d'une bibliothèque externe.

Application d'une méthode d'optimisation

L'autre élément que nous proposons est un module d'optimisation. Contrairement au module d'apprentissage, celui-ci utilise une bibliothèque externe développée par un tiers et qu'il faut intégrer dans l'architecture. Nous l'utilisons grâce à la classe *Optimisation* (c.f. figure 2.8). Elle nous sert d'interface entre le modèle atomique *esprit* et la bibliothèque d'optimisation.

Nous allons dans un premier temps nous pencher sur la structure de cette bibliothèque (figure 2.12) qui contient différents types d'objets dont les plus importants sont :

— **Problème** : la classe centrale, qui représente la classe de problème que nous souhaitons traiter. Nous devrons donc définir un problème qui prendra en compte des paramètres à optimiser ainsi que des contraintes et des critères pour trouver une solution.

— **Contrainte** : classe qui nous permet de gérer les contraintes que nous souhaitons attribuer pour l'optimisation.

— **Critère** : classe permettant de gérer les critères d'arrêt d'une optimisation ou dans le cas d'une optimisation hybride les critères de changement d'algorithme.

— Et enfin, nous avons les algorithmes d'optimisation qui héritent d'une classe abstraite qui permet au problème d'utiliser de la même manière tous ces algorithmes ou de rajouter des algorithmes d'optimisation. Ceci de la même façon que pour les algorithmes d'apprentissage dans la section précédente.

Maintenant nous allons regarder de plus près la liaison entre cette bibliothèque et notre architecture. Comme nous l'avons souligné, nous avons une classe statique *Optimisation* qui nous sert d'interface avec la bibliothèque. Nous avons choisi d'utiliser une classe statique car elle n'a pour objectif que de servir d'intermédiaire. Elle sert de conteneur pour des ensembles de méthodes qui opèrent simplement sur des paramètres d'entrée et n'ont pas à obtenir ou définir de champs d'instance internes. De plus, dans le cas d'une classe statique, le programme appelle le constructeur statique une seule fois et la classe statique reste en mémoire pendant toute la durée de vie (exécution) de programme. Nous n'aurons donc pas besoin d'utiliser d'instance de cette classe, mais uniquement de faire appel à elle pour interagir avec la bibliothèque. Dans cette situation, Quel que soit le type d'optimisation que nous voulons utiliser, l'interface restera la même, seuls les appels à la bibliothèque changeront. Ce qui fait qu'il devient inutile d'utiliser un mécanisme d'héritage.

Cette classe contient des méthodes permettant au modèle de lancer une optimisation. Ces méthodes sont :

— **initProblem()** : cette fonction initialise le problème, lui associe des paramètres, une fonction d'évaluation ou encore des critères à respecter ;

— **evaluation()** : cette fonction est comparable à la fonction de récompense (reward) de l'apprentissage. Elle permet d'évaluer une solution. Comme dans le cas de l'apprentissage, cette fonction est très fortement dépendante du modèle. Elle permet de fixer les critères d'optimisation, c'est-à-dire

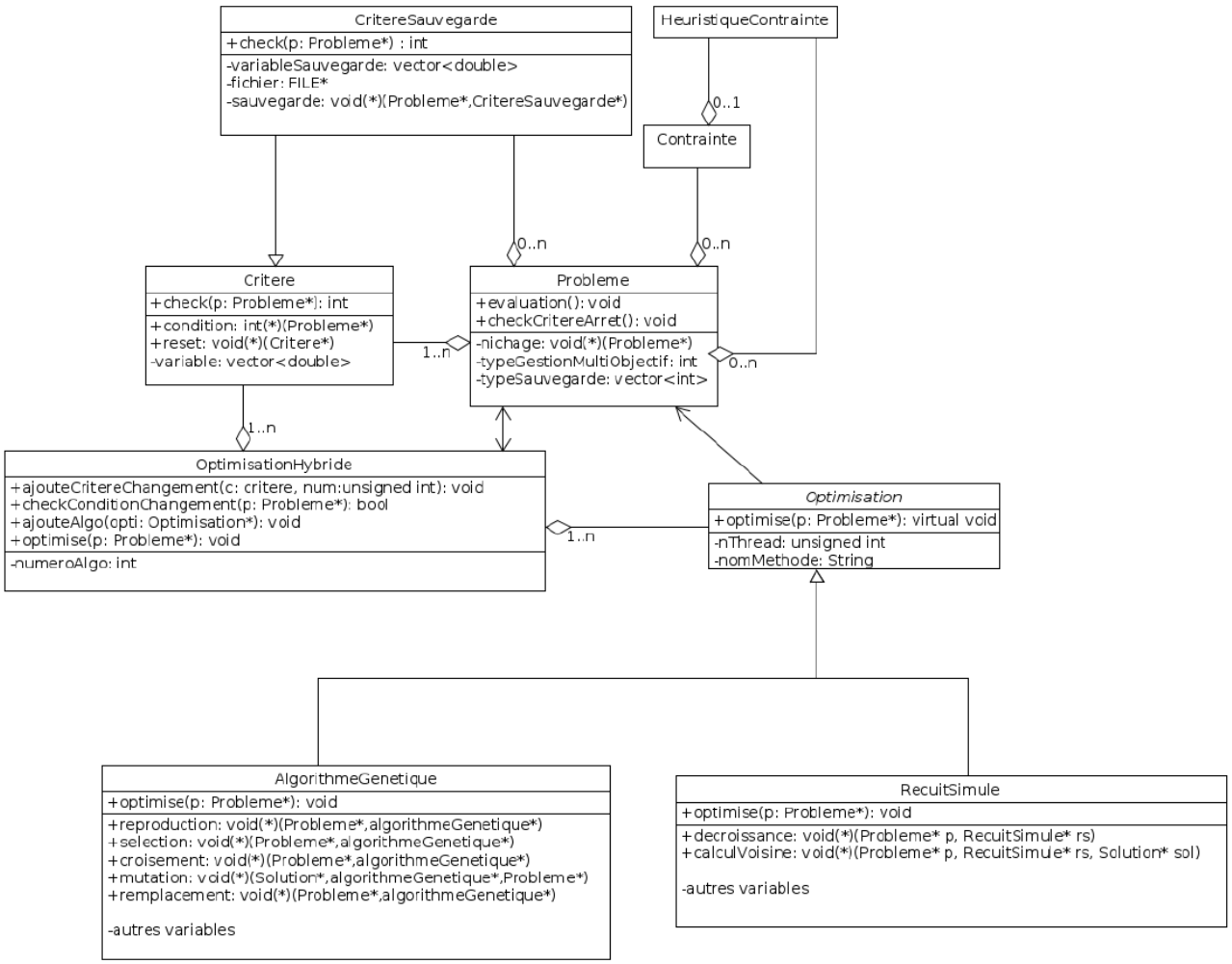


FIGURE 2.12 – Représentation générale de la bibliothèque d'optimisation issue de [Poiron-Guidoni, 2018].

ce qui fait qu'une solution est meilleure qu'une autre. Elle consiste à mettre une note la solution proposée. Dans la version actuelle, plus une note est proche de zéro meilleure est la solution ;

— **simu()** : cette fonction permet de faire évoluer la simulation au sein de l'optimisation. Elle doit correspondre au modèle qui est développé. Cette fonction est vouée à être directement liée au modèle atomique dans une prochaine version. Nous avons choisi de garder ce fonctionnement pour le moment dans la mesure où il nous faudra modifier le fonctionnement de la bibliothèque pour faire ce lien.

— **main()** : enfin, cette fonction permet de lancer les optimisations voulues et de récupérer les résultats.

Algorithm 3 fonction main d'utilisation de l'optimisation

```
1: function main(temps, listeParametres)
2: Probleme p
3: retour
4: initProblem(p)
5: ColonieAbeilleArtificielle abeille
6: p.simulation = simu
7: abeille.optimise(p)
8: for i = 0 to temps do
9:    retour.add(p.solution.param[i])
10: end for
11: return retour
```

Nous pouvons voir sur l'algorithme 3 le déroulement d'une demande d'optimisation à partir d'un modèle atomique. Dans cet algorithme nous définissons un *probleme* ligne 2, une liste *retour* ligne 3 que nous remplirons à la fin de l'optimisation avec les résultats de celle-ci. À la ligne 4 le problème est initialisé. Ensuite, ligne 5 nous définissons un objet *abeille* de type $ColonieAbeilleArtificielle$ qui est un type d'algorithme d'optimisation.

On attribue à la ligne suivante la dynamique de simulation au problème p.

Ligne 7 nous lançons l'optimisation. Enfin des lignes 9 à 11, nous mettons dans la liste *retour* l'ensemble des résultats de l'optimisation.

La figure 2.13 décrit un diagramme de séquence d'une optimisation comme nous l'avons implémenté. Nous y retrouvons donc toutes les phases que nous avons précédemment décrites ainsi que les appels des différentes fonctions que nous venons de voir. Ceci illustre le processus d'optimisation et nous voyons bien ici que contrairement à l'apprentissage, l'agent n'intervient pas au cours du processus. Pour l'optimisation l'agent ne fait qu'appeler une méthode et reçoit le résultat. Tout le processus étant géré par la bibliothèque externe, la classe *Optimisation* ne servant que d'interface.

Nous venons de présenter une structure permettant à un agent de prendre des décisions s'il en est capable, ou de se donner les moyens de le faire. Pour cela nous lui avons permis d'utiliser des méthodes d'apprentissage par renforcement afin de construire une base de connaissance dans le but de prendre des décisions. Toujours dans l'optique de donner une meilleure adaptabilité a nos agents, cette structure permet de résoudre un problème en utilisant des méthodes d'optimisation issues d'une bibliothèque développée par notre équipe. Nous allons maintenant présenter un exemple simple d'utilisation afin de montrer la mise en œuvre de notre approche.

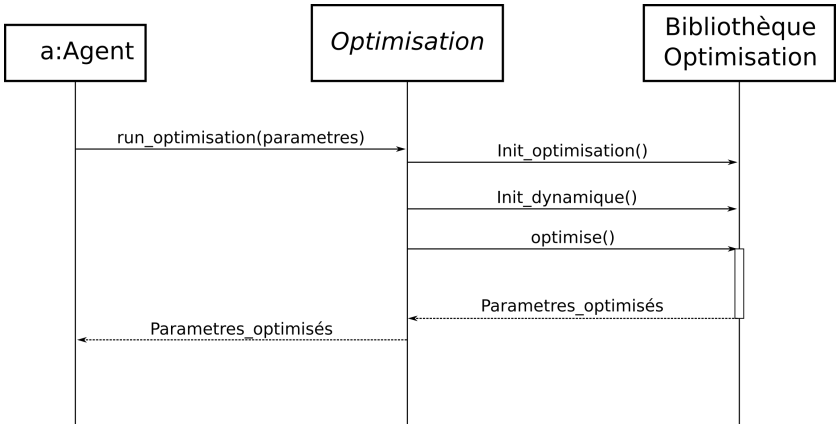


FIGURE 2.13 – Diagramme de séquence du processus d'optimisation

2.3.4 Exemple didactique de simulation

Dans cette section, nous allons présenter un cas simple d'application qui consiste à jouer au jeu du morpion ou tictactoe. C'est un cas d'utilisation qui se prête à l'usage d'apprentissage automatique, dans la mesure où les espaces d'états et d'actions sont relativement restreints. Il se prête également très bien à des résolutions par des méthodes déterministes pour des raisons similaires. En effet l'arbre de possibilités d'une partie est calculable et relativement restreint.

Rappelons rapidement le principe de ce jeu. Une partie met en action deux agents et une grille de jeu carrée 3×3 comme sur la figure 2.14. A tour de rôle, chaque agent inscrit respectivement une croix ou un cercle dans une case. Celui qui réussit à remplir toute une rangée (ligne colonne ou diagonale) avec sa marque remporte la partie.

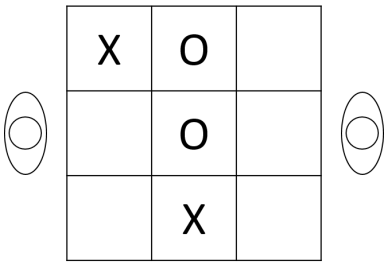


FIGURE 2.14 – Jeu de morpion

Il existe plusieurs algorithmes permettant à un agent de jouer automatiquement. Nous avons utilisé ici deux méthodes différentes : l'une utilisant notre module d'apprentissage afin de permettre à un agent

d'apprendre à jouer. L'autre un algorithme *MinMax* présenté dans la section 2.3.4 et qui nous servira de référence, car capable de parfaitement résoudre ce jeu.

Nous allons détailler comment nous avons modélisé le système, puis comment se déroule le processus d'apprentissage et enfin quels sont nos résultats et conclusions.

Modèle

Le jeu du morpion entre deux agents modélisés avec notre approche d'un point de vue PDEVS est représenté dans la figure 2.15. Il se compose de deux agents cognitifs qui sont également des modèles atomiques, ainsi que d'un modèle atomique qui est le plateau de jeu. Comme il n'y a pas d'environnement où positionner les agents la modélisation est très simple et la représentation des corps n'est pas nécessaire.

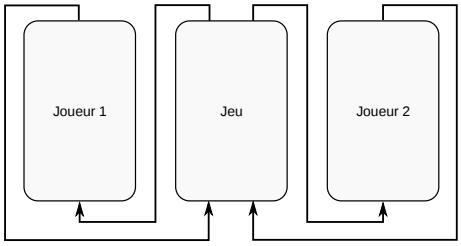


FIGURE 2.15 – modélisation DEVS du jeu de morpion

À chaque tour, l'agent qui doit jouer a un état courant qui est l'état de la grille de jeu. Connaissant cet état, l'agent doit décider de sa prochaine action. L'ensemble des actions possibles correspondent à l'ensemble des cases encore vides dans lesquelles l'agent peut inscrire sa marque.

Modèle plateau de jeu Le plateau de jeu est un simple modèle atomique PDEVS codé à partir de VLE. Il a la structure que nous pouvons observer sur la figure 2.16

Nous retrouvons donc notre classe *Jeu* qui hérite de la classe *Dynamics*. Nous retrouvons ainsi des attributs tels que la grille de jeu : représentée par une chaîne de caractères, un état : qui sert à gérer l'ordre des opérations à effectuer, le tour : un entier qui retient où nous en sommes dans la partie (qui doit jouer) et enfin un attribut gagnant qui à la fin d'une partie garde le résultat en mémoire. Nous voyons aussi quelques méthodes telles que *checkEnd*() qui se charge de détecter la fin de la partie ainsi que le vainqueur et *display*() qui est un affichage de la grille de jeu.

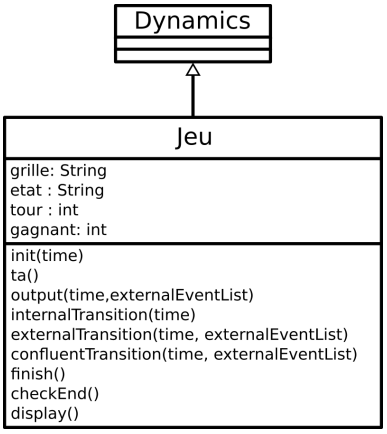


FIGURE 2.16 – Classe du modèle atomique Jeu

Modèle joueur Ce modèle est un agent cognitif qui va utiliser le module d'apprentissage. Il a donc la structure que nous pouvons observer sur la figure 2.17.

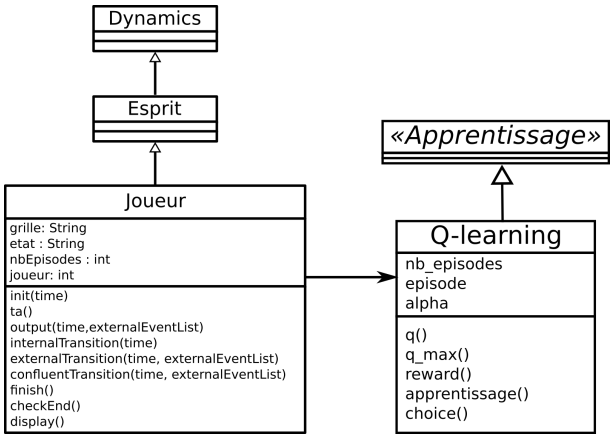


FIGURE 2.17 – Classe de l'agent Joueur

Comme nous pouvons le voir, notre classe *Joueur* qui est notre agent, hérite de la classe *Esprit* dont nous avons donné les fondements précédemment (cf. section 2.3).

D'une part les fonctions héritées de la classe *Dynamics* qui sont celles nécessaires au fonctionnement d'un modèle atomique dans VLE et d'autre part un objet de type *Q-learning* qui possède tout ce qui permet d'utiliser cette forme d'apprentissage.

Chacun des joueurs est un agent cognitif défini selon notre approche et illustré par la figure 2.18.

Une fois ce système modélisé il faut apprendre à nos agents à jouer.

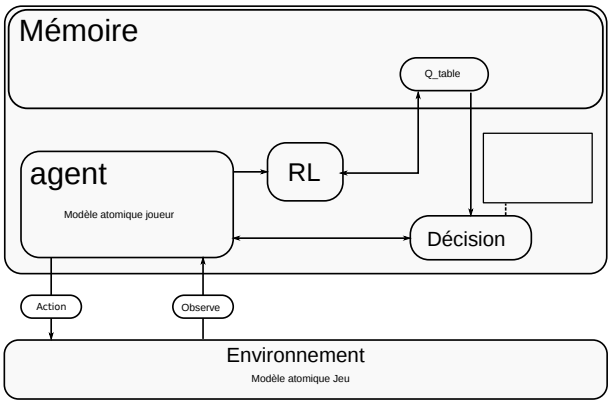


FIGURE 2.18 – modélisation DEVS du jeu de morpion

L'apprentissage

Nous utilisons une méthode d'apprentissage par renforcement de type Q-learning pour permettre aux deux agents d'apprendre à jouer.

Une initialisation qui met en place une partie définit l'ordre de jeu et met en place une grille vierge. C'est également à cette étape que sont définis les paramètres d'apprentissage. La partie commence alors et chaque agent joue jusqu'à la fin de la partie en respectant le principe d'exploration/exploitation.

Les actions ainsi que les états courants sont gardés en mémoire. Une fois la partie terminée les récompenses de tous les choix faits durant la partie sont calculés et la fonction de valeur est mise à jour.

À ce moment là :

— soit l'entraînement n'est pas fini. Dans ce cas, nous recommençons une nouvelle partie en mettant à jour les paramètres d'apprentissage ;

— soit l'entraînement est terminé auquel cas le processus se termine.

Une fois l'apprentissage terminé, nous pouvons laisser l'agent jouer, il utilisera donc les connaissances acquises pendant l'entraînement, et choisira à chaque tour le coup à jouer grâce au processus de décision.

Implémentation

Agent joueur L'agent joueur va pouvoir décider de débuter une partie, dans ce cas il devra envoyer un message pour initialiser le jeu. Lorsqu'une partie est en cours, il va recevoir une mise à jour de la grille de jeu à chaque fois qu'il doit jouer et retourne en sortie le coup qu'il veut jouer. En fin de partie il va

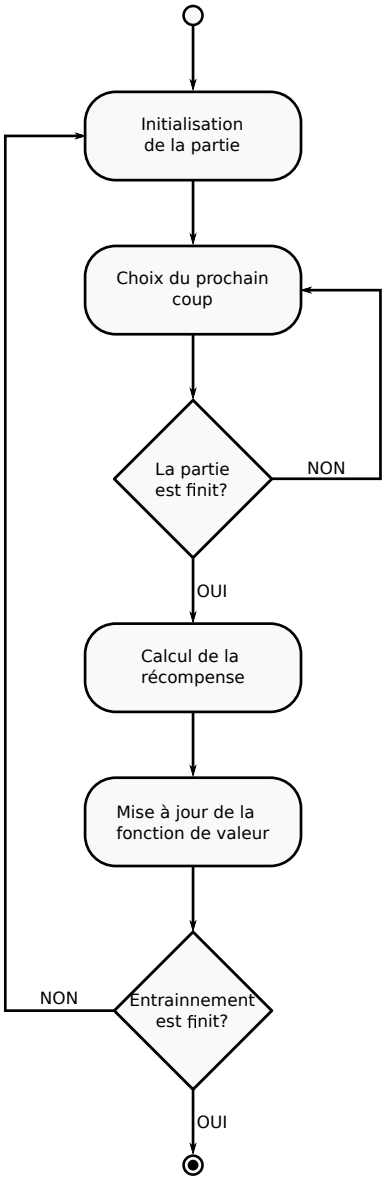


FIGURE 2.19 – Algorithme d'apprentissage du morpion

recevoir sur ses ports d'entrées un évènement qui lui dira s'il a gagné, perdu ou si la partie se termine sur un match nul.

Lors d'un entraînement, l'agent va donc relancer un certain nombre de parties et utiliser un objet du type $Q-learning$ pour démarrer un apprentissage. Durant cette phase d'apprentissage, c'est l'objet $Q-learning$ qui dictera les choix d'actions à faire ainsi que le nombre de parties à lancer. L'algorithme 4

représente une implémentation du comportement de la classe *Q-learning* que l'on peut voir dans la figure 2.17. Nous retrouvons donc les principales méthodes nécessaires à l'apprentissage. Nous noterons avant tout que la fonction de valeur est représentée par la map *q_table*. Dans la suite de ce paragraphe nous parlerons donc de la *q_table* et non de la fonction de valeur. *q* et *qmax* qui sont des méthodes intermédiaire permettant la mise à jour de la *q_table*.

reward est ici la fonction qui permet le calcul de la récompense et permet la mise à jour de la *q_table*.

choice est la fonction permettant de choisir quel coup l'agent va jouer dans le cadre de l'apprentissage.

Enfin *apprentissage* est la fonction qui permet de gérer le déroulement de l'apprentissage.

Résultats

Pour évaluer les résultats de ces entraînements, nous avons confronté notre agent cognitif entraîné à un autre agent cognitif entraîné à partir de l'algorithme *MinMax*. Un algorithme que l'on retrouve dans le domaine des jeux depuis longtemps et qui a connue des déclinaisons comme nous pouvons le voir dans [McAllester, 1985] ou [Rivest, 1987].

Nous allons expliquer le fonctionnement de l'algorithme *MinMax* utilisé ici.

Pour choisir l'action la plus intéressante à effectuer, cet algorithme va calculer toutes les issues possibles à partir de la situation courante. Pour cela il va générer un arbre avec la situation courante comme racine qui aura un fils par action possible et ainsi de suite jusqu'à la fin de la partie, ou jusqu'à une profondeur maximale. La figure 2.20 illustre un exemple d'arbre utilisé par l'algorithme.

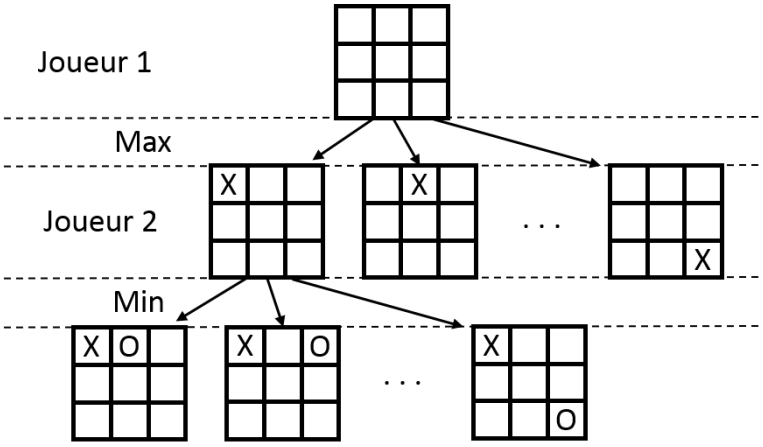


FIGURE 2.20 – Arbre de possibilité de l'algorithme *MinMax*

L'algorithme va considérer qu'il doit sélectionner l'action la plus avantageuse pour lui mêmelui-même Dans le cas d'un jeu de morpion il est possible de calculer cet arbre en totalité à chaque tour de jeu. Ceci fait que l'algorithme *MinMax* devrait toujours obtenir la victoire ou le match nul si son adversaire est suffisamment habile pour ne pas lui laisser une possibilité de victoire.

Nous avons obtenu les résultats suivants lors d'un ensemble d'apprentissage d'un agent confronté à un autre agent également en cours d'apprentissage ou utilisant l'algorithme *MinMax*.

La figure 2.21 présente le pourcentage de victoire de deux agents apprenant tous les deux à jouer l'un contre l'autre.

FIGURE 2.21 – Apprentissage du morpion par deux agents

La figure 2.22 présente le pourcentage de victoire des deux agents. Un agent apprenant à jouer (agent 1) contre l'autre utilisant l'algorithme *MinMax* (agent 2). Nous pouvons voir ici que l'agent utilisant *MinMax* remporte un maximum de victoires en tout début d'apprentissage. Nous pouvons clairement

voir la progression de l'agent qui apprend à contrer l'algorithme *MinMax* pour obtenir de plus en plus de matchs nuls. À la fin de l'apprentissage le pourcentage de match nul est de 100%

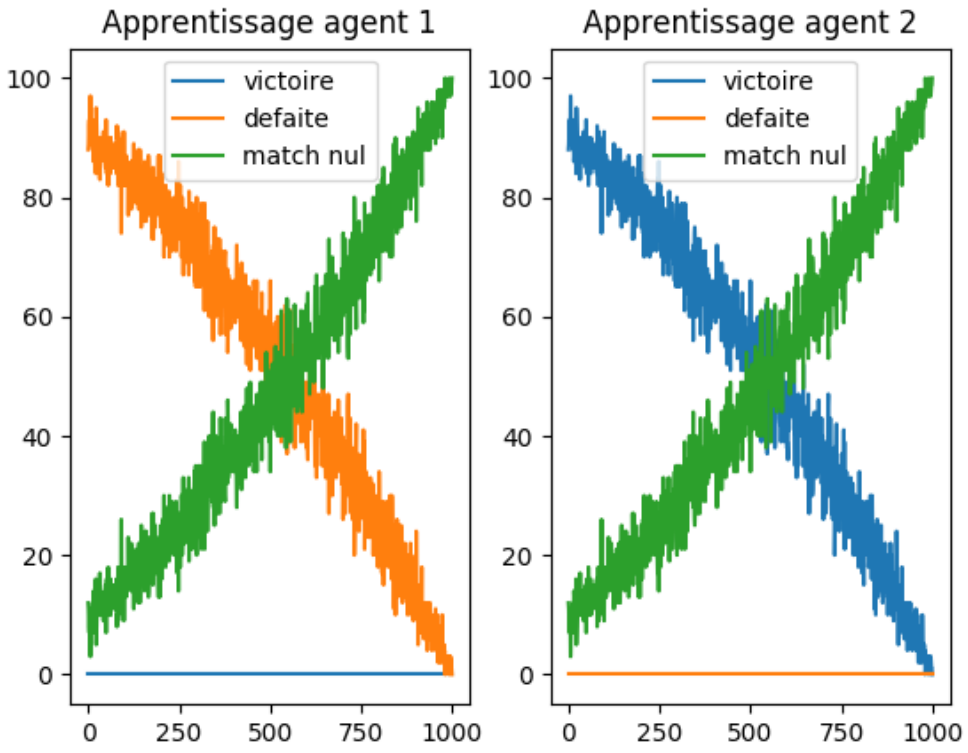


FIGURE 2.22 – Apprentissage du morpion par un agent face à un agent utilisant l'algorithme *MinMax*

Nous pouvons observer dans les deux cas de figure que nous avons exposé la progression de l'apprentissage et la capacité que gagne un agent à prendre la meilleure décision au cours de l'apprentissage. Ainsi nous donnons un premier cas d'application qui nous a permis d'illustrer de façons concrète une partie des mécanismes que nous proposons. Nous avons choisi d'exposer celui-ci sur un exemple didactique pour améliorer la compréhension et la clarté des notions génériques que nous avons abordés dans ce chapitre.

2.4 Conclusion

Le concept d'environnement et le concept d'agent cognitif ont été intégrés dans un modèle computationnel au sein du framework *VLE*. Nous pouvons maintenant simuler le système complexe d'une pêcherie

selon une démarche de modélisation à base d'agents.

Nous avons dans ce chapitre intégré une structure de données dans un modèle informatique générique implémentant des mécanismes de prise de décision. Il permet de décrire des agents cognitifs respectant DPDEMAS [Franceschini et al., 2017].

Cette structure permet à un agent d'utiliser des méthodes d'apprentissage par renforcement afin de construire une base de connaissances dans le but de prendre des décisions. Elle permet également à un agent de résoudre un problème en utilisant des méthodes d'optimisation issues d'une bibliothèque développée par notre équipe.

Nous en avons donné une description conceptuelle ainsi qu'une description de son implémentation. Cette dernière n'a pu être que sommaire dans la mesure où dans la plupart des cas elle est intrinsèquement liée au modèle développé. Notre structure facilite également l'intégration de méthodes différentes. Nous n'avons traité dans nos travaux qu'une infime partie des méthodes existantes, il sera intéressant par la suite d'en développer d'autres et de les implémenter.

Dans le chapitre suivant, nous allons proposer un modèle informatique appliqué aux pêcheries et présenter nos modèles exécutables ainsi que nos résultats.

Algorithm 4 Classe Q-learning

Variable $nb_episodes, episode, alpha$
$q\_table = \{grille, \{coup, valeur\}\}$ // une map associant une string(grille) avec un couple d'entier coup (le coup a jouer) et une valeur
function $apprentissage(temps, listeParametres)$
$fin \leftarrow (nbepisodes > episode)$ // fin recoit $false$ si l'episode courant est le dernier
$episode \leftarrow episode + 1$
return fin
end function

function $q(action, grille)$
return $q_table[grille][action]$
end function

function qmax(grille)
$val \leftarrow -\infty$
for all $k \in q\_table[grille]$ **do**
  if $k.valeur > val$ **then**
    $val \leftarrow k.valeur$
    $result \leftarrow k.clef$
  end if
end for
return $result$
end function

function $reward(historique, finPartiefinPartie)$
for all $k \in historique$ **do**
  $str \leftarrow k.first$
  if $(finPartie = 1)$ **then**
    $result = 1$
  end if
  if $(finPartie = 0)$ **then**
    $result = -1$
  end if
  if $(finPartie = 2)$ **then**
    $result = 0$
  end if
  $q_table[str][historique[k].second] = q(str, historique[k].second) + 0.1 * (result + 0.9 * qmax(str) - q(str, historique[k].second))$
end for
end function

function $choice(grille, episode)$
$list < int > jouables$
$jouables \leftarrow coups_jouables(grille)$ //la liste jouable reçoit tous les coups que l'agent peu techniquement jouer
$epsilon \leftarrow 100 * (nb_episodes - episode)/nb_episodes$
$test \leftarrow random(100)$
if $test > epsilon$ **then**
  $result \leftarrow jouables[0]$
  for all $i \in jouables$ **do**
    if $q_table[grille][result] <= q_table[grille][i]$ **then**
      $result \leftarrow i$
    end if
  end for
else
  $result \leftarrow jouables[random(jouables.size()]$
end if
return $result$
end function

Chapitre 3

EXPÉRIMENTATIONS

Dans ce chapitre, nous abordons la problématique de l'industrie de la pêche qui exploite les ressources halieutiques.

L'application proposée est liée au programme MoonFish débuté en juin 2017 et coordonné par l'Université de Corse et l'UMR SPE. Il associe un ensemble d'acteurs scientifiques et institutionnels, à savoir l'Office de l'Environnement de la Corse (OEC), la STARESO : la station de recherches océanographiques et sous-marines dédiée à la recherche marine en méditerranée, ainsi que le Comité Régional des pêches Maritimes et Élevages Marins de Corse (CRPMEM). Il s'agit d'un projet de recherche pluridisciplinaire traitant de la gestion des ressources halieutiques et s'appuyant en grande partie sur la *science des pêcheries*.

Dans ce chapitre, nous proposons des modèles de pêcherie à base d'agents pour évaluer des scénarios de gestion de biomasse (poissons). Les modèles que nous allons présenter tendent à identifier et à quantifier par la simulation informatique les facteurs environnementaux et économiques d'intérêt dans une pêcherie, et plus particulièrement dans le contexte corse. Notre objectif principal, au travers de ces simulations, est de montrer la capacité de l'approche que nous avons présentée dans le chapitre précédent, à permettre aux agents de prendre des décisions cohérentes et adaptées au contexte.

Nous décrivons dans un premier temps le modèle de pêcherie tel que nous l'avons utilisé ainsi que les paramètres que nous avons fixés. Nous présentons dans un second temps deux exemples de modélisation à base d'agents en exposant pour chacun d'entre eux, le modèle informatique et le modèle exécutable. Il s'ensuit une discussion des résultats de simulations obtenus. Enfin, nous concluons ce chapitre d'exemples.

3.1 Modélisation d'une pêcherie

Historiquement, la *science des pêcheries*[1] s'est développée dans le domaine de la biologie, puis elle a rejoint le domaine de l'économie et enfin la simulation informatique. Cela fait de la science des pêcheries une science transdisciplinaire en charge de la gestion raisonnée des ressources halieutiques ([Anderson, 2015], [Angelini and Moloney, 2007]).

Ainsi les modèles de pêcherie sont construits comme des abstractions que l'on observe à travers le prisme de la composition d'éléments issus de la *dynamique des populations* en *biologie*, et d'éléments définissant un contexte de *ressource limitée* en *science économique* ([Garcia et al., 2017, Gaertner and Dreyfus-Leon, 2004]

1. *Fisheries science* en anglais.

Grâce à ce type de modèle, il est possible de simuler la dynamique de sous-systèmes biologiques et économiques en interaction.

Les *modèles de pêcherie* que l'on rencontre dans la littérature sont pour la plupart composés :

— d'un *modèle de croissance de population*[2] F décrivant l'évolution de la biomasse dans le temps ;

— et d'un *modèle économique* H décrivant l'impact de l'activité économique sur la ressource halieutique.

En général, le *modèle de croissance de population* F décrit la dynamique biologique globale d'une population d'individus d'un, ou de plusieurs stocks de poissons. La plupart du temps, dans la littérature, les auteurs utilisent des *équations différentielles* pour décrire cette évolution dans le temps continu [Sharp et al., 1983, Stewart, 2013, Pavé, 2012]. Le modèle économique H est quant à lui habituellement issu de la *théorie économique de la pêche*[3] qui a vu le jour en Amérique du Nord en 1950 [Boncœur et al., 2000].

La plupart des travaux traitent d'une espèce spécifique (modèle mono-espèce) dans le but d'une exploitation industrielle, comme par exemple dans [Maravelias et al., 2014, Silvestri and Maynou, 2009] ou encore [Fournier et al., 1998]. Ces travaux, bien que courants dans le domaine, nécessitent une grande quantité de données quantitatives et des connaissances sur le recrutement annuel des espèces biologiques d'intérêt pour pouvoir être exploités numériquement. Cela n'est pas possible dans le contexte de notre étude, la pêche sur le littoral corse étant une pêche artisanale (beaucoup d'incertitudes sur les prises) et multi-espèces (une soixantaine exploitée sur presque deux cents recensées d'après le Comité Régional des pêches Maritimes et Élevages Marins de Corse[4]). La collecte de données qualitatives et quantitatives est un problème bien connu dans le domaine de la science des pêcheries et de nombreux travaux ont été effectués afin d'apporter des solutions permettant de s'en affranchir, ils sont énumérés dans l'état de l'art proposé par [Yanıkoğlu et al., 2019] et cette problématique fait actuellement l'objet d'une autre thèse en optimisation robuste et apprentissage par renforcement.

Dans un premier temps, nous nous sommes orientés vers le modèle canonique de *Graham-Schaefer* présenté comme référence dans la littérature, quand il s'agit de modéliser des dynamiques de populations d'espèces marines exploitées [Britten et al., 2017, Dudley, 2008, Quinn, 2003].

2. Dans la littérature on trouve également le terme de *modèle de dynamique des populations*.
3. *Fishery economic theory* en anglais.
4. https ://www.crpmem.corsica/Les-especes-pechees_r40.html

Bien que ce type de modèle soit moins précis qu'un modèle structuré de population tel que le modèle de [Leslie, 1945], comme indiqué dans [Gulland et al., 1970], il présente l'avantage de ne nécessiter que peu de données quantitatives pour être exploitable en simulation informatique. Dans ce type de modèle, on formalise la dynamique de la population d'une ou plusieurs espèces marines sous la forme d'un stock correspondant à l'expression d'une biomasse globale, sans faire de distinction entre les différents individus. Dans ce type de modèle, utilisé également par la FAO pour fixer les quotas de pêche, une hypothèse forte de modélisation considère que sans exploitation, le stock tend à s'équilibrer. Nous y reviendrons dans la section qui décrit le modèle.

Nos expérimentations vont se baser sur le modèle de *Graham-Schaefer*, présenté dans la section suivante, complété d'une analyse préliminaire du modèle et de ses paramètres afin d'affiner notre état des connaissances.

Notre modèle sera alimenté par les données de la littérature présentées dans [Le Manacha et al., 2011, Le Manach and Pauly, 2015] que l'on retrouve sur le site sea around us. Sur la base de ces données qui sont des estimations, nous sélectionnerons certains couples de paramètres qui peuvent solutionner le modèle. Cette étape va nous permettre de caractériser 4 stocks qui modélisent chacun une espèce particulière (Denti, Sar, Rascasse, Langouste rouge). En spécialisant le modèle, nous pourrons tester différents cas d'applications.

3.1.1 Modèle de Graham-Schaefer

Il semblerait que ce soit *M. Graham* et *M.B. Schaefer* dans [Schaefer, 1954, Schaefer, 1957] qui introduisirent dans leur modèle la notion de pêcherie. Á travers la notion de captures, ils proposèrent un modèle économique simple d'exploitation d'une zone de pêche [Quinn, 2003].

Le modèle de population que nous utilisons, appelé modèle de *Graham Schaefer*, est une variante du modèle original d'exploitation d'une zone de pêche. Il est détaillé dans [Pella and Tomlinson, 1969] et s'écrit dans sa forme différentielle :

$$\frac{dB(t)}{dt} = r(1 - \frac{B(t)}{k})B(t) - C(t) \tag{3.1}$$

Il utilise trois paramètres biologiques :

— k : la biomasse à l'équilibre (en masse) ;

— r : le taux de croissance par unité de temps ;

— $B(t)$: la biomasse à l'instant t (en masse) ;

La résolution de cette équation nécessite la connaissance d'un paramètre n'apparaissant pas explicitement dans l'équation, à savoir la valeur de la biomasse à l'instant initial, notée B_0. De plus, dans un contexte de pêcherie, on introduit deux autres paramètres : $E(t)$ représentant l'effort de pêche appliqué sur le stock du temps t au temps $t+1$; et q la capturabilité par unité d'effort (la probabilité pour une unité de biomasse d'être capturée lorsqu'une unité d'effort est déployée). Les captures $C(t)$ sont alors calculées grâce à la formule suivante :

$$C(t) = q \times E(t) \times B(t) \tag{3.2}$$

avec :

— q : la capturabilité par unité d'effort ;

— $E(t)$: l'effort de pêche, que l'on va chercher à définir par apprentissage ;

Ceci nous donne dans le cadre d'une simulation discrète la formule d'évolution suivante :

$$B(t+1) = B(t) + r(1 - \frac{B(t)}{k})B(t) - C(t) \tag{3.3}$$

La notion d'équilibre dans les modèles de type *Graham-Schaefer* est fondamentale. L'équilibre est dit dynamique quand il y a toujours des naissances (entrées) et des décès (sorties), et que ces deux phénomènes se compensent.

À titre d'exemple, on peut citer l'étude de *Verhulst* [Schtickzelle, 1981] caractérisant les stocks à partir du nombre d'individus et supposant pour la première fois la notion d'équilibre.

Sur la base de ce modèle discrétisé et des données à notre disposition, nous avons dans un premier temps cherché à définir la population initiale B_0.

3.1.2 Calibration du modèle

Dans [Martelloni et al., 2019] nous avions proposé une approche pour sélectionner des couples qui donnaient des résultats corrects et correspondants à des sorties observées dans la littérature [Le Manacha et al., 2011].

Dans un premier temps, la population initiale est créée selon le tableau 3.1 montrant les bornes inférieures et supérieures de chaque paramètre. On y retrouve chacun des 4 paramètres du modèle, ainsi qu'un ensemble de 58 valeurs, représentant l'effort de pêche à appliquer pour chacune des années de données que nous avons.

Paramètre	Borne inférieure	Borne supérieure
q	0	1
r	0	1
Biomasse initiale	10	5000
k	10	5000
Effort[58]	0	5

TABLE 3.1 – Bornes des paramètres

À chaque itération et pour chaque solution potentielle, une simulation de la pêcherie va être réalisée. Cela va déterminer des captures associées à la dynamique de population proposée. La solution est alors évaluée en comparant captures "réelles", C, et captures simulées, Cs, suivant la fonction suivante :

$$f(x) = \sum_{i=0}^{nbAnnee} [C_i - Cs_i]^2$$

Les solutions proposées seront alors modifiées de différentes façon en fonction de l'algorithme utilisé afin de caler le plus possible aux captures réelles au cours des itérations. Après avoir testé plusieurs méthodes, l'essaim particulaire standard de 2007 (SPSO 2007, [Clerc, 2012]) semble être particulièrement efficace sur ce problème.

La figure 3.1 montre les courbes d'évolution des captures réelles de denti, en pointillées bleus, et celles simulées avec la dynamique de population que nous déterminons, en trait plein rouge. Les deux courbes étant presque parfaitement superposées, le calage par méthode d'optimisation est particulièrement efficace.

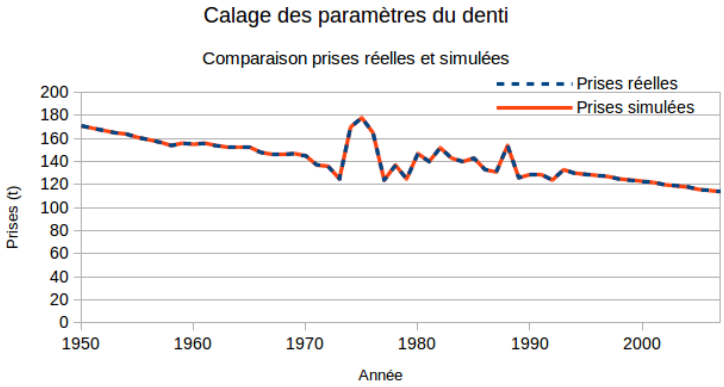


FIGURE 3.1 – Comparaison de l'évolution des prises réelles et simulées de denti

Dans la suite de ce document, nous utiliserons les dynamiques de populations de chaque espèce, déterminées à partir de notre méthode, afin de tester des stratégies de pêche que nous comparerons aux résultats des données de [Le Manacha et al., 2011].

Cette même méthode a été développée dans [Poiron-Guidoni et al., 2020] pour calibrer le modèle dans le cas où il y a de l'incertitude dans les données.

C'est à partir de ces travaux préliminaires que nous proposons d'utiliser les valeurs présentées dans le tableau 3.2. Nous avons décidé d'utiliser quatre stocks différents car cela nous permettait d'illustrer notre approche sur plusieurs espèces différentes sans que cela influe trop sur le temps de simulation. Ces stocks sont issus des données de [Le Manacha et al., 2011], les stocks 0,1,2,3 correspondent respectivement aux données du Denti, de la Langouste rouge, du Sar et de la Rascasse.

TABLE 3.2 – Paramètres utilisés.

	q	r	k	B0	Espèce
Stock 0	0.09	0.54	1060.94	991.88	Denti
Stock 1	0.18	0.48	2841.98	1154.39	Langouste Rouge
Stock 2	0.08	0.48	1454.56	1126.17	Sar
Stock 3	0.11	0.4	1595.37	611.94	Rascasse

Sur cette base, nous allons proposer deux cas d'application. Dans la première, nous souhaitons mettre en avant les capacités des agents intelligents de notre approche à apprendre et adapter leur décision en fonction de leur intérêt. Dans la seconde, nous souhaitons complexifier le plan d'expérience, toujours avec les mêmes agents mais en spatialisant l'environnement.

3.2 Simulation par agents non spatialisée

Dans cette section, nous présentons une première expérience de simulation. Celle-ci se fonde sur un modèle de pêcherie à 5 composants. Nous simulons des scénarios avec quotas, sans prise en compte de l'espace dans l'environnement. Nous considérons quatre stocks de poissons sur lesquels des agents pêcheurs appliquent un effort de pêche annuelle. Un agent modélisant une autorité de régulation de la pêche fournit chaque année la valeur d'un quota aux agents pêcheurs. Ce modèle à base d'agents permet d'expérimenter par la simulation plusieurs scénarios permettant d'étudier les effets des quotas sur les populations de poissons soumises à une activité de pêche. Les expériences de simulation nous permettent de tester nos algorithmes d'apprentissage et d'optimisation. Pour cela, nous mettons en œuvre des agents intelligents qui utilisent notamment les mécanismes que nous avons décrits dans la section 2.3.2.

Nous présentons dans les sous-sections qui suivent le modèle informatique, puis le modèle exécutable, avant de discuter des résultats de simulations dans une dernière sous-section.

3.2.1 Modèle informatique

La figure 3.2 est une illustration des composants utilisés pour notre modèle. Nous retrouvons donc quatre composants pêcherie eux-même contenant un stock et un agent pêcheur et un composant représentant une autorité de régulation.

Les stocks sont modélisés à partir du modèle de *Graham-Schaefer* et caractérisés par les paramètres présentés dans le tableau 3.2.

Sur la figure 3.2, le composant numéro 5 correspond à l'agent intelligent *decideur* modélisant la dynamique décisionnelle des autorités chargées d'édicter les règles et les quotas de pêche durant l'expérience de simulation. Les valeurs des quotas sont calculées avec le module d'optimisation présenté dans la section 2.3.2. Ce composant a pour objectif de sauvegarder la biomasse. Les composants 1, 2, 3 et 4 du modèle possèdent des stocks ayant une dynamique de population similaire, ils ne diffèrent que par les valeurs des paramètres du modèle de population servant à caractériser les différents stocks $(r_i, k_i, q_i, B0_i)$. Le tableau 3.2 recense les paramètres des modèles de population des composants 1 à 4.

Ainsi, comme cela est illustré sur la figure 3.2, notre modèle pêcherie repose essentiellement sur deux types de composants :

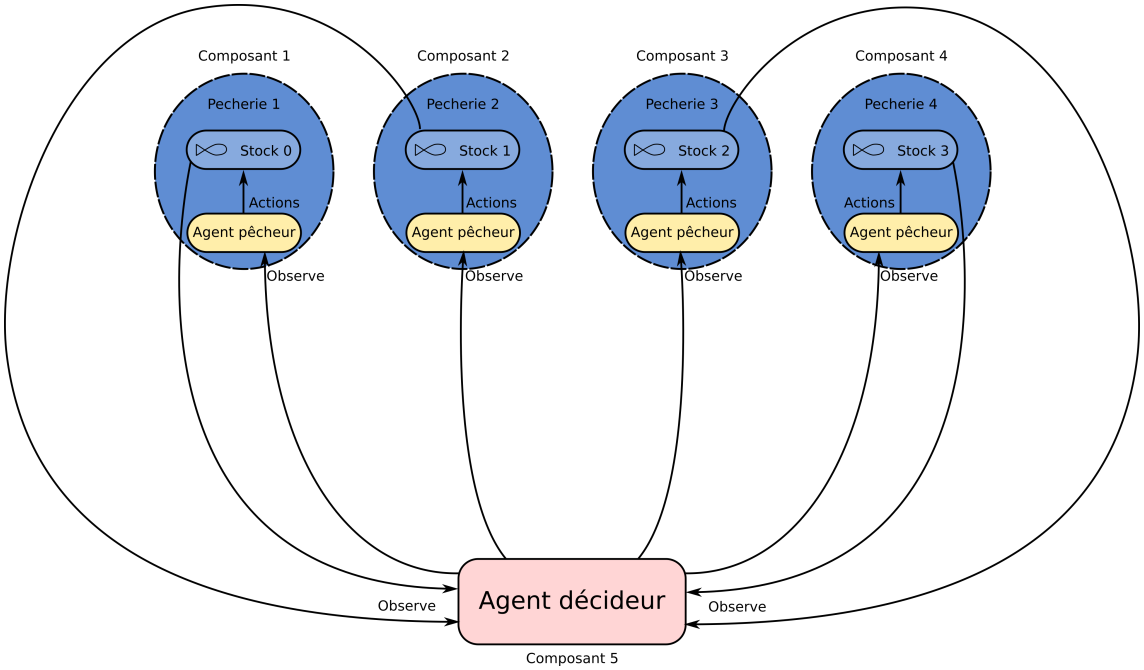


FIGURE 3.2 – Modèle non spatialisé

— un premier type de composant sert à décrire la croissance de la population d'une espèce dans l'environnement de la pêcherie ;

— un second type de composant modélise l'action de pêche d'agents pêcheurs, dont le rôle est de prélever une quantité de biomasse q_i dans les stocks de population. Cette dernière dynamique utilise le module d'apprentissage que nous avons développé afin de calculer l'effort de pêche que nous appliquons à chaque itération. Les pêcheurs ayant pour objectif de maximiser leurs prises.

Agent décideur

Dans le modèle que nous présentons, le décideur a une connaissance de la dynamique des populations. En utilisant ces connaissances, il définit des quotas de pêche à respecter et fait des prévisions sur l'évolution des populations. Son objectif est de sauvegarder la biomasse et de faire en sorte qu'elle reste proche de celle de départ. Les entrées du modèle sont la biomasse courante de chaque stock.

$$X = \{B\_stock0, B\_stock1, B\_stock2, B\_stock3\}$$

de la même manière, les sorties du modèle sont les quotas fixés de chaque stock.

$$Y = \{quota\_stock0, \_stock1, \_stock2, \_stock3\}$$

Enfin, l'état du modèle peut être représenté par l'ensemble des paramètres de chaque stock en considérant la population courante comme $B0$.

$$S = \{\{r_0, q_0, k_0, B\_stock0\}, \{r_1, q_1, k_1, B\_stock1\}, \{r_2, q_2, k_2, B\_stock2\}, \{r_3, q_3, k_3, B\_stock3\}\}$$

Pour chaque stock, nous devons fournir à l'algorithme d'optimisation la dynamique correspondante, c'est-à-dire comment elle évolue dans le temps, ainsi que les paramètres nécessaires. Lors de l'optimisation, l'évolution de la biomasse sera simulée en suivant l'équation 3.3 et les paramètres lui correspondant (c.f. tableau 3.2).

Le paramètre que nous cherchons à optimiser avec cet exemple est $C(t)$ qui décrit les captures. L'algorithme d'optimisation va fixer ce paramètre à chaque pas de temps et lancer la simulation. Une fois la simulation achevée, les résultats sont comparés aux critères que nous souhaitons remplir. Les critères que nous avons utilisés ici tendent à considérer une solution comme meilleure si la population se maintient au niveau de la biomasse initiale ($B_{final} \simeq B_0$). Si les résultats sont conformes, le jeu de paramètres est sauvegardé, sinon l'algorithme d'optimisation met à jour les paramètres du modèle et relance une simulation. Le fonctionnement du modèle est décrit dans la section modèle exécutable 3.2.2.

Agents pêcheries

Le comportement des stocks des séparés en deux parties : (1) l'environnement décrit par le modèle de croissance de population ne fait appel à aucun processus de décision. Il exécute la fonction d'évolution 3.3 ; (2) l'agent pêcheur, lui comprend la partie décisionnelle dont l'objectif est de maximiser ses prises. Il fixe l'effort de pêche à suivre chaque année. Nous considérons que pour prendre cette décision les données connues sont uniquement le quota fixé pour cette année ainsi que la quantité de poissons pris l'année précédente.

L'ensemble des entrées du modèle contient le quota fixé par le décideur.

$$X = \{quota\}$$

de la même manière, l'ensemble des sorties du modèle contient la biomasse courante du stock.

$$Y = \{B_courante\}$$

Enfin, l'état du modèle peut être représenté par l'ensemble des paramètres du stock ainsi que du quota fixé et la biomasse capturée l'année précédente.

$$S = \{r, q, k, B_courante, quota, precedent\_capture\}$$

Pour cela, notre module d'apprentissage est utilisé grâce à un objet de type *Q-learning*. Nous utilisons ici le mécanisme décrit à la section 2.3.2.

Pour prendre cette décision, nous utilisons une base de connaissance. Le modèle a un état courant qui représente ses connaissances. Il s'agit du quota fixé et de la biomasse capturée l'année précédente soit : $quota, precedent\_capture$. Ces connaissances doivent permettre de choisir quel effort de pêche appliquer. Cet effort permet de calculer les prises qui seront faites pour l'année courante selon la formule 3.2.

La base de connaissance a la forme d'une table associant un état $(quota, precedent\_capture)$ à un $effort$ de pêche et à une *valeur* qui représente à quel point ce choix d'effort est bon ou pas en fonction de l'état.

Si cette base existe, un simple processus de décision permet de retrouver en fonction de l'état courant le meilleur effort à effectuer.

Si elle n'existe pas, il faut alors la créer. Pour cela nous utilisons un processus d'apprentissage par renforcement. Ce processus suit le principe décrit dans la section 2.3.2.

Dans ce cas notre processus va initialiser l'apprentissage. Pour cela nous avons fixé un nombre d'épisodes de simulation ainsi que le temps d'une simulation. Puis, les paramètres d'apprentissage tels que α et ϵ sont mis à jour. Ensuite, le modèle est initialisé : les stocks sont créés et la population initiale fixée à B_0 selon le tableau 3.2 ainsi qu'avec les quotas reçus du modèle décideur.

Enfin la simulation est exécutée. À chaque pas de simulation, ce qui correspond dans le modèle à une

année d'exploitation, un effort de pêche est choisi. D'ailleurs, dans la partie implémentation (cf. section sec :exemple1 :computationnel)nous parlerons d'année de simulation plutôt que de pas de temps. L'effort est appliqué et la biomasse du stock évolue. La fonction de valeur est ensuite mise à jour. Nous calculons une récompense en fonction : des prises de l'année courante, des prises de l'année précédente et du quota imposé et met à jour les connaissances de l'agent. Cette fonction de valeur permet de fixer une politique. Cette politique est affinée tout au long de l'apprentissage. C'est cette fonction que nous allons faire varier avec nos 3 scénarios.

Si une simulation se termine et que l'apprentissage n'est pas fini, on remet à jour les paramètres d'apprentissage et on recommence. Si tous les épisodes ont été exécutés, l'apprentissage prend fin.

Les expériences débutent par le paramétrage de quotas et des sanctions en cas de dépassement. Nous avons défini différents scénarios que nous avons déjà évoqués lors de la description de la fonction de récompense dans la section précédente.

Nos scénarios sont construits sur une base identique comme suit :

1. cas où les quotas ne sont donnés qu'à titre informatif, dans ce cas la récompense obtenue ne dépend que des prises effectuées. Nous parlerons de scénario *naïf* (NA) ;
2. cas où un dépassement du quota sans aucune tolérance entraîne une amende forfaitaire qui vient contrebalancer les gains temporaires d'une pêche excessive. Nous parlerons de scénario *amende forfait* (AF) ;
3. cas où un dépassement du quota entraîne une amende proportionelle au dépassement. Nous parlerons de scénario *amende proportionelle* (AP).

Nous allons maintenant aborder en détail l'implémentation du modèle dans la plateforme VLE.

3.2.2 Modèle exécutable

Après avoir détaillé le fonctionnement de notre modèle informatique, nous présentons ici son implémentation en langage C++. L'algorithme 5 expose le code du modèle atomique décideur, il est associé à l'algorithme 6 qui correspond à l'interface du module d'optimisation. Nous retrouvons dans ce modèle les fonctions usuelles d'un modèle atomique.

La définition des quotas, ligne 8 de l'algorithme 5, fait appel à l'interface d'optimisation. Cette interface est décrite dans l'algorithme 6 et est composée des fonctions suivantes :

Algorithm 5 Modèle décideur

1: $population = \{pop, r, q, k, p_{msy}, \{Quota\}\}$ //définition d'une population contenant les paramètres nécessaires
2: **Variable** $temps\_simulation, etat, temps\_quota$
3: **function** $init()$
4: $temps\_simulation \leftarrow 100$
5: $etat \leftarrow$ "$init$"
6: $popu \leftarrow initPopulation()$ // initialisation de l'ensemble des populations a partir du fichier de configuration
7: **for all** $p \in popu$ **do**
8: $p.quota \leftarrow optimisation.main(temps\_quota, p)$ // calcul des quotas en faisant appel a l'interface optimisation
9: **end for**
10: **return** 0
11: **end function**
12:
13: **function** $time\_advance()$
14: **if** $etat =$ "$simule$" **then**
15: $resultat \leftarrow 0$
16: **else**
17: **if** $etat =$ "$init$" **then**
18: $resultat \leftarrow 0$
19: **else**
20: $resultat \leftarrow \infty$
21: **end if**
22: **end if**
23: **return** $resultat$
24: **end function**
25:
26: **function** $\lambda()$ // la fonction de sortie qui permet d'initialiser les modèles espèces.
27: // pour chaque espèce on envoie sur le port de sortie correspondant les caractéristiques de l'espèce et les quotas fixés
28: **for all** $p \in popu$ **do**
29: $output.send(p)$
30: **if** $state =$ "$sendQuota$" **then**
31: $output.send(p.quota)$
32: **end if**
33: **end for**
34: **end function**
35:
36: **function** $\delta_{int}()$ // la fonction de transition interne ne fait que passer de l'état "init" a l'état "wait"
37: **if** $etat =$ "$init$" **then**
38: $etat = wait$
39: **end if**
40: **end function**
41:
42: **function** $\delta_{ext}(events)$
43: **if** $events.askquota()$ **then**
44: $p.quota =\leftarrow optimisation.main(temps\_quota, p)$
45: $state \leftarrow$ "$sendQuota$"
46: **end if**
47: **end function**

Algorithm 6 Optimisation

1: **Variables** $(dim, b, q, c_{msy}, r, k)$
2: **function** $evaluationTest(Solution)$
3: **for** $i \leftarrow 0$ to $dimension$ **do**
4: $somme \leftarrow somme + (B_0 - B)^2$
5: **end for**
6: $Solution.note \leftarrow somme$
7: **return** $somme$
8: **end function**
9:
10: **function** $initProblemTest(problem, dimension)$
11: **for** $i \in dimension$ **do**
12: $listParametre.add(nouveauParametre())$
13: **end for**
14: $problem.initParametre(listParametre)$
15: $probleme.initEval(evaluationTest)$
16: **end function**
17:
18: **function** $simu(solution)$
19: $b \leftarrow B_0$
20: **for** $i \leftarrow 0$ to $dimension$ **do**
21: $c[i] \leftarrow solution.param[i].valeur * q * b$
22: $b \leftarrow b + r * (1 - (b/k)) * b - c[i]$
23: **end for**
24: **end function**
25:
26: **function** $main(dim, b, q, cmsy, r, k)$
27: $initProblemeTest(problem, 100)$
28: $problem.simulationExterne \leftarrow true$
29: $problem.simulation \leftarrow simu$
30: $problem.checkParametre \leftarrow true$
31: $ColonieAbeilleArtificielle.optimise(problem)$
32: $simu(problem.solution)$
33: **return** c
34: **end function**

— *evaluationTest* est la fonction d'évaluation qui permet de fixer les critères d'optimisation (lignes 1 à 6). Nous devons préciser que plus une note est proche de zéro plus elle est bonne. Nous avons choisi d'effectuer une optimisation en privilégiant une population courante proche de la population initiale. Il en va de même à la ligne 6, plus la population courante b est proche de la population initiale B_0 plus la note sera proche de zéro. Ceci est fait pour chaque dimension, une dimension correspond au nombre de quotas successifs que nous voulons. Enfin une note est attribuée à la solution.

— *initProblemeTest* permet d'initialiser le problème. Il faut optimiser un paramètre pour chaque dimension du problème. Dans notre cas avoir un quota optimisé pour chaque temps de simulation. Pour cela, nous créons la liste de paramètres nécessaires (ligne 13), enfin nous lions ces paramètres (ligne 15) et la fonction d'évaluation au problème (ligne 16) ;

— *simu* permet de faire évoluer la simulation au sein de l'optimisation (lignes 19 à 25). Elle permet à l'optimisation de réaliser une simulation dans le but de recueillir les informations nécessaires à l'évaluation de la solution passée en paramètres. Comme nous l'avons dit dans le chapitre précédent, cette fonction est vouée à être directement liée au modèle atomique dans une prochaine version. Dans le cas présent cette fonction intègre les équations logistiques du modèle utilisé (l'équation 3.3) en fonction des paramètres de l'optimisation lignes 21 et 22. $solution.param[i]$ étant la liste des efforts en fonction de la dimension et donc $c[i]$, les captures correspondantes, qui seront à la fin de l'optimisation les quotas que notre agent *decideur* souhaite fixer.

— *main* : permet de coordonner l'optimisation, d'initialiser le problème, de fixer certains paramètres ainsi que l'algorithme d'optimisation utilisé. Ici nous utilisons un algorithme dit de *colonie d'abeille artificielle*. Enfin, cette fonction a la charge de déclencher l'exécution de retourner les résultats.

De la même manière les algorithmes 7 et 8 exposent l'implémentation du modèle atomique pêcherie. Les parties les plus importantes qui définissent le comportement du modèle sont :

— $\delta_{ext}(events)$: la fonction de transition externe. Cette fonction est utilisée pour initialiser les stocks avec les données provenant du décideur. Au démarrage de la simulation, le modèle pêcherie doit recevoir un évènement du décideur contenant les caractéristiques de la population décrite par le modèle. Nous mettons donc à jour les variables de population avec les données reçues. Le temps courant est alors initialisé à 0 et le temps complet de simulation est initialisé en fonction du nombre d'années prévu par le décideur. Cette fonction sert également à recevoir les quotas fixés par le décideur.

— $\delta_{int}()$: la fonction de transition interne est entièrement décrite dans l'algorithme 8. Nous avons choisi de séparer cette fonction car c'est la plus importante de ce modèle et parce qu'elle comprend plusieurs cas de figure. Nous vérifions tout d'abord que le système est bien en phase de simulation. Ensuite si nous effectuons un apprentissage (ligne 6 à 10), nous utilisons le module d'apprentissage pour définir un effort de pêche (ligne 8). L'effort nous permet de calculer les prises associées. Avec

ces prises, nous pouvons calculer la récompense obtenues à partir de ce choix. Cette fonction de récompense est définie dans le module d'apprentissage. Si nous n'effectuons pas d'apprentissage (ligne 11 à 13), les prises effectuées sont directement calculées par la *fonction de choix* du modèle. Comme nous connaissons les prises effectuées durant le pas de simulation courant, nous pouvons mettre à jour la population (ligne 15) Nous traitons ensuite le cas où la simulation est terminée. (ligne 16) Si l'apprentissage est terminé, ou si nous n'effectuons pas d'apprentissage, le système passe en attente. Si nous sommes en cours d'apprentissage (lignes 18 à 22), la fonction *run* du module d'apprentissage est exécutée. Celle-ci permet de faire avancer l'apprentissage d'un épisode. Pour que le prochain épisode puisse démarrer, le temps courant de la simulation est réinitialisé à 0 ainsi que la population à la valeur de départ B_0.

Le module d'apprentissage est associé à l'algorithme 9. Ce module utilise les variables définies aux lignes 1 et 2. Enfin on retrouve la table qui contiendra le résultat de l'apprentissage (ligne 2) cette table, nous l'avons définie comme un dictionnaire (donc une liste $clef/valeur$) imbriqué.C'est-à-dire que la première clef *str* qui est une chaîne de caractères contenant le quota et les prises de l'année précédente et fixe donc la situation actuelle, nous donne accès à un deuxième dictionnaire dont la clef est l'effort à appliquer et la valeur associée, celle de l'action selon le principe de la fonction de valeur. Ce qui comprend les fonctions suivantes que nous avons détaillées précédemment :

— $reward(quota, lastPrise, prise, effort, next\_quota)$: la fonction de récompense, cette fonction permet dans notre cas de définir la politique que nous voulons tester. Cette fonction permet de définir ce que va gagner ou perdre un agent en effectuant une action. Dans le cas présent, un pêcheur gagnera des prises, la biomasse capturée. Ce gain dépendra donc de la quantité de prises effectuées. En plus de cela nous avons défini plusieurs scénarios : Un premier cas où les quotas ne sont donnés qu'à titre informatif, dans ce cas la récompense obtenue ne dépend que des prises effectuées. Un deuxième où un dépassement du quota entraîne une amende forfaitaire qui vient contrebalancer les gains temporaires d'une pêche excessive. Un troisième où un dépassement du quota entraîne une amende proportionelle au dépassement.

— $choice(quota, lastPrise)$: la fonction qui permet de choisir une action en fonction du quota et des prises de l'année précédente (lignes 15 à 35). elle suit le principe dit $\epsilon - greedy$. Nous avons donc une variable ϵ qui décroît au cours de l'apprentissage (ligne 16). À chaque choix, nous tirons une

Algorithm 7 Modèle Pêcherie (partie 1)

```
1: population = {pop, r, q, k, p_msy, {Quota}} //Les caractéristiques de la population du stock.
2: Variable state, file, apprend, table, reward, MSY, prisesG, lastprise
3: function init()
4: state ← "init"
5: apprend ← "true" // Variable qui permet d'effectuer une boucle d'apprentissage.
6: apprentissage(30000) // Initialise l'apprentissage avec 30000 épisodes. RETURN ∞
7: end function
8:
9: function time_advance()
10: if state = "simule" then
11:    resultat ← 1
12: else
13:    resultat ← ∞
14: end if
15: return resultat
16: end function
17:
18: function λ()
19: if state = "new then
20:    output.send("new")
21: end if
22: end function
23:
24: function δ_ext(events)
25: if state = "init" then
26:    //Teste si le message reçu sur le port d'entrée est de type population.
27:    if events.isPopulation() then
28:       population ← events
29:       pop_Initiale ← population.pop
30:       temps_simulation ← population.quotas.size()
31:       temps ← 0
32:       state ← "simule"
33:    end if
34: end if
35: end function
```

valeur entière aléatoire entre 0 et 100, si cette valeur est supérieure à ϵ, ou si le cas actuel n'est pas encore connu, c'est-à-dire si la combinaison du quota et des prises de l'année précédente n'est pas du tout présente dans la table, alors nous faisons un choix aléatoire de l'effort de pêche (ligne 30 et 33) qui sera une valeur aléatoire sur l'intervalle $[0.00, 4.99]$ ce qui est équivalent à l'intervalle des efforts possibles utilisés dans la calibration du modèle (cf. tableau 3.1). Sinon nous récupérons dans la table le meilleur choix connu (lignes 23 à 28), c'est-à-dire, pour le quota actuel et les prises précédentes, l'effort de pêche existant dans la table *q_table* ayant la valeur la plus élevée.

Algorithm 8 Modèle Pecherie (partie 2)

```
1: function δ_int()
2: lastPrise ← 0
3: if state = "simule" then
4:    quota ← population.quotas[temps]
5:    next_quota ← population.quotas[temps + 1]
6:    if apprend = "true" then
7:       lastprise ← prises
8:       effort ← Apprentissage.choix(quota, lastprise)
9:       prises ← arrondi(effort ∗ population.q ∗ population.pop)
10:      Apprentissage.reward(quotat, lastprise, prises, effort, next_quota)
11:   else
12:      lastprise ← prises
13:      prises ← choix(quota, lastprise)
14:   end if
15:   population.pop ← population.pop + (population.r ∗ (1 − (population.pop/population.k)) ∗
      population.pop) − prises
16:   if temps > temps_simulation then
17:      state ← "wait
18:      if Apprentissage.run() then
19:         state ← "simule
20:         temps ← 0
21:         population.pop ← pop_Initiale
22:      else
23:         apprend ← "false"
24:      end if
25:   end if
26: end if
27: end function
```

— *run*() : cette fonction (lignes 37 à 41) fait évoluer l'apprentissage en vérifiant s'il est terminé ou pas et en incrémentant les épisodes d'apprentissage. Celle-ci retourne un booléen qui est vrai tant que l'apprentissage continu et qui passe à faux dès que celui-ci est terminé.

Algorithm 9 Apprentissage

Variables $test, nbepisodes, episode, \alpha$
$q\_table = \{str, \{effort, valeur\}\}$
function $reward(quota, lastPrise, prise, effort, next\_quota)$
$\alpha \leftarrow 1 - (episode/nbepisodes$
$result \leftarrow prise$
if $prise > quota$ **then**
  $result \leftarrow result - (prise - quota) * 15)$ // Dans le cas du scénario 3
  $result \leftarrow result - 1000$ //Dans le cas du scénario 2
end if
$str \leftarrow prise + ";" + lastPrise$
$str2 \leftarrow next\_quota + ";" + prises$
$q\_table[str][effort] = q(str, effort) + alpha * (result + 1 * qmax(str2) - q(str, effort));$
end function

function $choice(quota, lastPrise)$
$\epsilon \leftarrow 100 * (nbepisodes - episode)/nbepisodes$
$test \leftarrow random(100)$ //fonction qui retourne un nombre entier aléatoire sur l'intervalle [0,99]
$str \leftarrow prise + ";" + lastPrise$
// si la clef str existe dans la table
if $q\_table.isIn(str)$ **then**
  if $test > \epsilon$ **then**
    $val \leftarrow -\infty$
    for all $k \in q\_table[str]$ **do**
      if $k.valeur > val$ **then**
        $val \leftarrow k.valeur$
        $result \leftarrow k.clef$
      end if
    end for
  else
    $result = random(500))/100$
  end if
else
  $result = random(500))/100$
end if
end function

function $run()$
$fin \leftarrow (nbepisodes > episode)$ // fin recoit $false$ si l'episode courant est le dernier
$episode \leftarrow episode + 1$
return fin
end function

function $q(str, effort)$
return $q\_table[str][effort]$
end function

function $qmax(str)$
$val \leftarrow -\infty$
for all $k \in q\_table[str]$ **do**
  if $k.valeur > val$ **then**
    $val \leftarrow k.valeur$
    $result \leftarrow k.clef$
  end if
end for
return $result$
end function

3.2.3 Résultats et discussions

Objectifs

Pour rappel, nous souhaitons utiliser la simulation comme outil d'aide à la décision. Notre outil se veut vecteur d'intégration de multiples disciplines. Dans ce premier exemple nous proposons de mettre à l'épreuve notre approche avec un modèle de pêcherie.

Le modèle que nous présentons dans cet exemple est un modèle théorique qui s'applique très bien à démontrer la cohérence de notre approche mais qui ne représente pas la réalité de façon suffisamment précise pour être utilisé dans un contexte de gestion des ressources halieutiques.

Le principal objectif de cet exemple se limite donc à montrer la capacité de l'approche que nous avons présentée dans le chapitre précédent, à prendre des décisions cohérentes.

Résultats des simulations

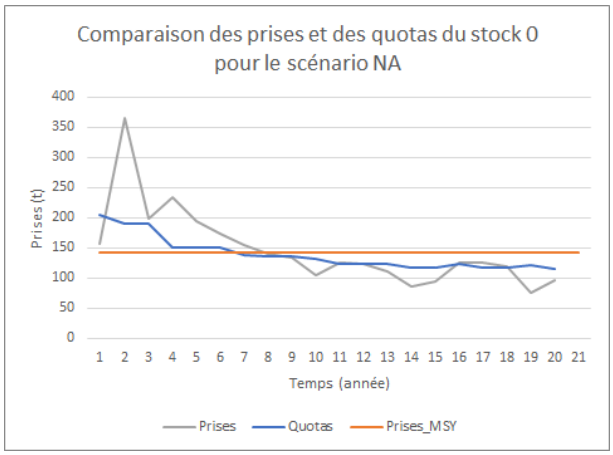
Avant de décrire nos résultats, nous allons rappeler les différents scénarios que nous avons simulés :

1. le premier scénario *naïf* (NA) où les quotas ne sont donnés qu'à titre informatif, dans ce cas la récompense obtenue ne dépend que des prises effectuées ;
2. le deuxième scénario *amende forfait* (AF) où un dépassement du quota sans aucune tolérance entraîne une amende forfaitaire qui vient contrebalancer les gains temporaires d'une pêche excessive ;
3. le troisième scénario applique une *amende proportionelle* (AP) proportionelle au dépassement du quota.

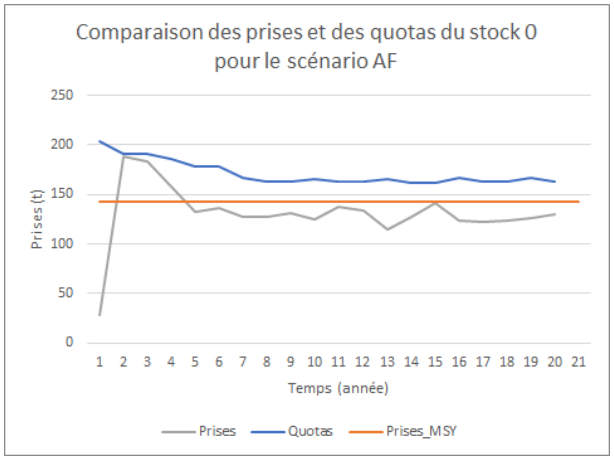
Nous allons maintenant regarder de plus près les prises effectuées en comparaison des quotas fixés par l'autorité de régulation pour le stock 0 sur la figure 3.3.

Le scénario NA, qui pour rappel ne tient compte des quotas que comme une information sans qu'il n'y ait d'amende si celui-ci n'est pas respecté, est le scénario qui dépasse le plus souvent ces quotas. On remarque que sur les 8 premières années les prises sont toujours supérieures aux quotas. ce qui comme nous l'avons vu précédemment a eu des conséquences importantes sur la biomasse.

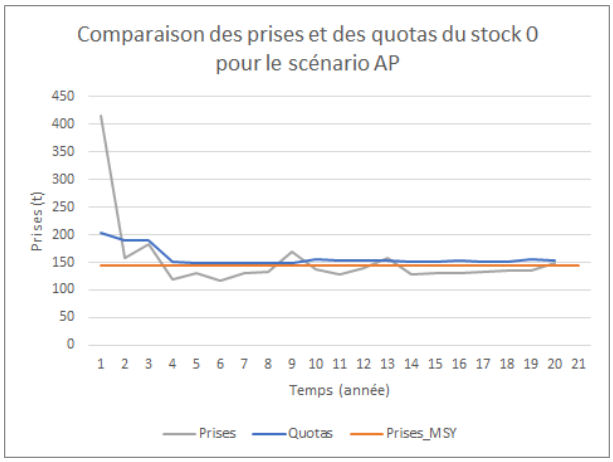
Le scénario AF qui pour rappel applique une amende en cas de dépassement des quotas. Nous observons bien sur la figure 3.3b que les quotas sont toujours respectés. Nous remarquons également que sur la majorité de la simulation, il y a un écart plus important entre les prises et les quotas.



(a) Comparaison entre les prises et les quotas pour le stock 0 pour le scénario NA.



(b) Comparaison entre les prises et les quotas pour le stock 0 pour le scénario AF.



(c) Comparaison entre les prises et les quotas pour le stock 0 pour le scénario AP.

FIGURE 3.3 – Comparaison entre les prises et les quotas pour le stock 0.

Le scénario AP, quant à lui, applique une sanction proportionelle au dépassement. On y observe un fort dépassement en tout début de simulation puis des prises qui restent légèrement en dessous des quotas. En nous concentrant sur le graphiques 3.3c il ressort que ce scénario entraîne un bon respect des quotas fixés. C'est le cas où les prises sont au plus proche des quotas.

On remarque également l'apparition d'une valeur remarquable, le Most sustainable Yeild (MSY). Elle représente le maximum de prises que l'on peut prendre par unité de temps sans impacter le stock et est un indicateur biologique bien connu [Tsikliras and Froese, 2019]. Bien que controversé [Walters et al., 2005], il semble être un bon indicateur dans notre cas. Les valeurs optimisées se rapprochent des valeurs du MSY

sans avoir utilisé cette valeur particulière à ce niveau de l'algorithme. C'est l'émergence d'une conclusion bien connue mais non recherchée et inattendue de nos simulations.

Bilan et discussion

Les résultats que nous venons de présenter sont cohérents avec ce que nous attendions avec un agent décideur cherchant à préserver la biomasse et des agents pêcheurs cherchant à maximiser leurs prises. Nous retrouvons des comportements logiques de la part d'agents intelligents artificiels. Néanmoins dans le cas que nous venons de présenter, le comportement des agents pêcheurs est difficilement comparable à un comportement humain réel. En effet nous avons affaire à des agents pêcheurs qui respectent toutes les règles et à un agent décideur omniscient.

Nous pouvons donc dire que dans le cadre de l'exemple, l'application de sanctions est un gage de maintien de la biomasse et de l'activité de pêche. L'application d'une sanction proportionnée est garante d'une biomasse maintenue ainsi que de prises proches des quotas.

Dans un cadre plus général, cet exemple nous permet de voir que nos agents intelligents réagissent comme nous l'attendions. Le comportement logique et artificiel de nos agents pêcheur et décideur, qui sont une limite pour le réalisme du modèle, sont un avantage ici. En effet, nous pouvons observer que le comportement de nos agents intelligents est bien logique et cohérent. Cet exemple nous a permis de penser que la capacité de nos agents à prendre une décision est bien cohérente, en revanche, nous n'avons pas testé ici l'approche multicomposant et la représentation spatiale que nous proposons. Nous allons donc dans la section suivante nous atteler à un exemple complet qui met en œuvre tous les aspects de l'approche que nous avons proposé dans le chapitre précédent.

3.3 Simulation par agents spatialisée

Nous exposerons dans ce dernier exemple la mise en œuvre des mécanismes liés à la décision et les particularités de l'approche multicomposant qui a été présentée dans le chapitre précédent (c.f. section 2.2).

Nous détaillons sur la figure 3.2 les composants du modèle de pêcherie. Nous utilisons quatre zones de pêche qui représentent le lieu de pêche où vivent différents stocks de poissons.

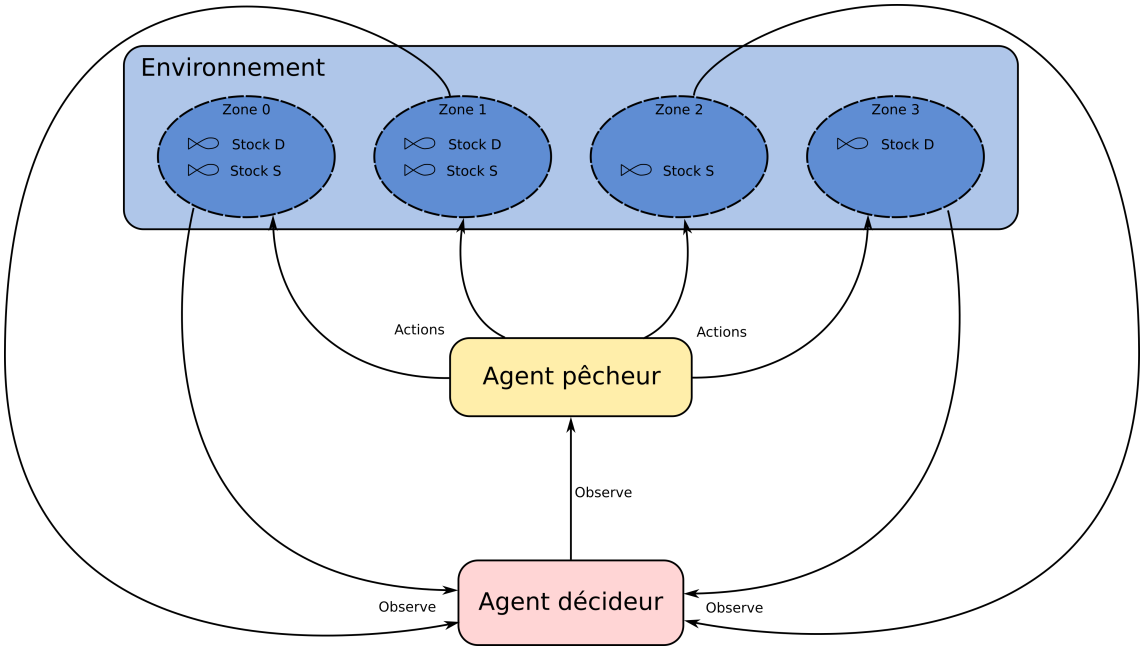


FIGURE 3.4 – Modèle spatialisé

3.3.1 Modèle informatique

Nous retrouvons dans le modèle de cet exemple trois types de composants différents : (1) un agent intelligent décideur qui aura le rôle d'une autorité de régulation. Il connaît la biomasse globale mais n'a pas connaissance des spécificités des zones. Cet agent a la responsabilité de fixer des quotas et des règles de pêche à l'échelle globale. L'objectif de cet agent est de sauvegarder la biomasse globale de chaque stock indépendemment des zones.

(2) Un agent intelligent pêcheur qui devra décider de l'effort de pêche à appliquer dans les différentes zones de pêche. L'objectif de cet agent est de maximiser ses prises dans chaque zone, comme dans l'exemple 1. La classe *Pêcheur* représente cet agent, il évolue dans les différentes zones de pêche. Il est responsable des décisions de pêche dans chaque zone. Le pêcheur applique le même prélèvement (effort de pêche) sur les deux stocks de la zone, c'est-à-dire qu'il prélève dans tous les stocks de la zone.

Et (3) un agent zone qui représentera une zone de pêche c'est-à-dire l'environnement de notre système. Le pas de temps est toujours fixé à l'année.

Nous retrouvons également 4 zones de pêche (0,1,2,3) dans lesquelles se répartissent les stocks de Denti (stock D) et de Sar (stock S).

De la même manière que dans l'exemple 1, nous allons expérimenter 3 scénarios :

1. le premier cas, que nous appellerons *amende avec tolérance* (AT), où un dépassement du quota avec une tolérance de 20% entraîne une amende forfaitaire qui vient contrebalancer les gains temporaires d'une pêche excessive.
2. Le deuxième cas, que nous appellerons *amende proportionnelle* (AP), où un dépassement du quota entraîne une amende proportionnelle au dépassement.
3. Le troisième cas, que nous appellerons *espèce protégée* (EP), où il n'y a aucun quota, en revanche si la biomasse d'un stock passe en dessous d'un seuil fixe (20% de la biomasse au MSY) que nous avons arbitrairement fixé, cette espèce devient protégée. Tant qu'un stock est protégé, la pêche de celui-ci entraîne une amande forfaitaire.

Agent zone Chaque zone de pêche contient des stocks de poissons. Exactement de la même manière que dans l'exemple précédent, les stocks ont une dynamique identique et ne diffèrent que par la valeur de leurs paramètres.

Dans les quatre zones de pêches, nous avons réparti deux stocks (Denti (D) et Sar (S)) du tableau 3.2 uniformément dans trois zones sur 4 avec la répartition suivante :

— zone 0 : stocks D et S présents ;

— zone 1 : stocks D et S présents ;

— zone 2 : stock D absent et S présent ;

— zone 3 : stock D présent et S absent.

Agent pêcheur L'agent pêcheur utilise notre approche cognitive décrite dans la section 2.3 pour fixer son effort de pêche. Cette description est illustrée sur la figure 3.5.

À chaque pas de simulation, l'agent devra décider pour chaque zone de pêche d'un effort à appliquer. Pour prendre cette décision, l'agent connaît les directives (quota par stock, ou zone protégée) données par l'agent décideur ainsi que les résultats précédents par zone.

Nous appliquons un effort de pêche à une zone qui contient elle-même plusieurs stocks de poissons, l'agent doit prendre en compte les directives qui concernent tous les stocks présents dans la zone. La

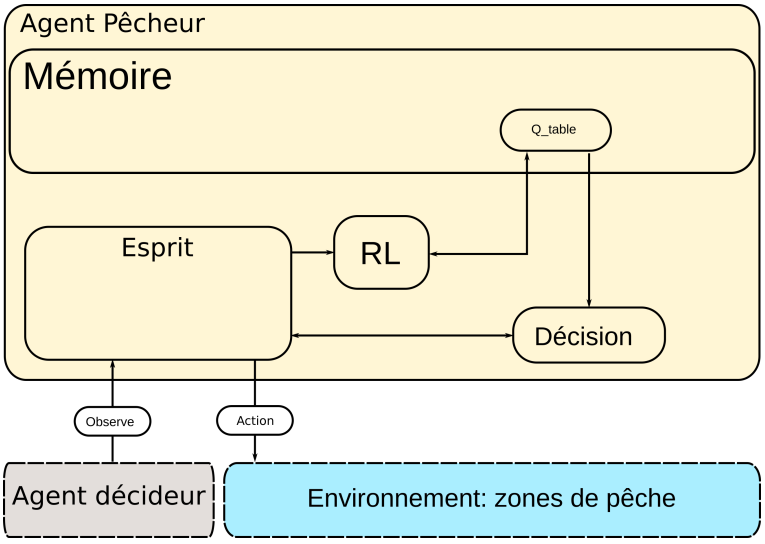


FIGURE 3.5 – Agent pecheur intelligent

fonction de prise de décision est décrite par :

$$\forall Z_i, e_i = f(liste\_quota, precedent\_capture_i)$$

Pour pouvoir prendre cette décision nous utilisons aussi une base de connaissances. Si elle n'existe pas, nous utilisons notre processus d'apprentissage pour la créer.

Dans cet exemple, le choix de l'effort de pêche pour chaque zone se fait en fonction du couple $(liste\_quota, precedent\_capture)$.

Pour implémenter cet agent, nous avons utilisé notre approche multicomposant. Cela nous a permis d'utiliser le mécanisme d'influence pour gérer les interactions *agent/agent* et *agent/environnement*. Un agent pêcheur aura une liste d'influencés qui seront les zones de pêche. En effet, il agira sur les populations des zones de pêche en effectuant des prélèvements en fonction des décisions prises. Il aura de la même façon une liste d'influenceurs composée uniquement de l'agent décideur. En effet les quotas fixés par l'agent décideur vont influencer la décision prise par l'agent pêcheur.

Agent décideur Le modèle décideur quant à lui représente les autorités chargées de décider des règles et des limites à appliquer à la pêche. Des quotas sont fixés en utilisant le module d'optimisation.

Le comportement et la structure de cet agent sont proches du décideur de l'exemple 1 (cf section

3.2.1). La différence est uniquement qu'il doit effectuer un pré-traitement qui va agréger les données de chaque zone pour connaître l'état courant des stocks à l'échelle globale ; pour ensuite calculer des quotas également à l'échelle globale.

Un agent décideur devra observer l'état des biomasses de poissons de chaque zone pour décider des quotas à mettre en place chaque année. De la même manière que dans l'exemple 1, il transmet ensuite les quotas à l'agent pêcheur.

Nous verrons dans la partie modèle exécutable ci-dessous, l'implémentation des parties les plus importantes de ces composants.

3.3.2 Modèle exécutable

L'algorithme 10 présente l'implémentation des agents *zone*.

Nous retrouvons à la ligne 1 une liste contenant un objet *population* qui est une structure contenant les différents paramètres d'une population. À la ligne 2 nous retrouvons la liste des influencés, cette liste contient les autres agents *zone* et permet la gestion des migrations de poissons entre les zones.

Des lignes 4 à 10, nous retrouvons l'initialisation du modèle, dans notre cas l'initialisation des populations.

Enfin aux lignes 23 à 30, nous avons la fonction de transition interne. C'est la principale fonction d'évolution du modèle. Chaque population présente dans la zone augmente. Puis nous effectuons les migrations. La migration fait transiter une petite partie de la biomasse d'un stock d'une zone à une autre.

L'algorithme 13 présente l'implémentation de l'agent décideur.

L'interface d'optimisation est exactement identique à celle utilisée dans le premier exemple (cf. 3.2.2) l'algorithme 6 est donc également utilisé ici.

L'agent pêcheur quant à lui est décrit par les algorithmes 11 et 12. Nous pouvons noter que contrairement au premier exemple dans lequel les pêcheurs apprenaient pour une espèce en particulier qui lui était associée, ici l'agent pêcheur doit apprendre à pécher dans les 4 zones que nous avons définies. Ainsi nous retrouvons au début de l'algorithme 11 la définition des listes nécessaires. Nous aurons dans notre cas précis 4 objets apprentissages qui vont permettre d'appliquer un processus d'apprentissage par zone.

Algorithm 10 Agent zone (Deuxième exemple)

1: $populations = \{\{pop, r, q, k, pop\_init\}\}$ //définition de la liste des stocks présents dans la zone contenant les paramètres nécessaires
2: $liste\_zones$ //liste d'influencés, la liste des zones de pêche pour gérer les migrations
3: **Variable** $temps\_simulation, etat$
4: **function** $init()$
5: $temps\_simulation \leftarrow 100$
6: $etat \leftarrow$ "$init$"
7: $popu \leftarrow initPopulation()$ // initialisation de l'ensemble des populations a des zones de pêche
8: **return** 0
9: **end function**
10:
11: **function** $time\_advance()$
12: **if** $etat =$ "$simule$" **then**
13: $resultat \leftarrow 1$
14: **else**
15: $resultat \leftarrow \infty$
16: **end if**
17: **return** $resultat$
18: **end function**
19:
20: **function** $\lambda()$
21: **end function**
22:
23: **function** $\delta_{int}()$
24: **for** $i = 0$ *a* $population.size()$ **do**
25: $population[i].pop = population[i].pop + population[i].pop * population[i].r * (1 - (population[i].pop - population[i].k))$
26: **for** $j = 0$ *a* $liste\_zones.size()$ **do**
27: $migration(liste\_zones[j].population[i])$
28: **end for**
29: **end for**
30: **end function**
31:
32: **function** $\delta_{ext}(events)$ //dans le contexte utilisé il n'y a pas de transition externe.
33: **end function**
34:
35: **function** $peche(effort)$
36: $prises \leftarrow 0$
37: **for** p in $populations$ **do**
38: $prises \leftarrow prises + populations[i].pop * effort * populations[i].q$
39: $populations[i].pop = populations[i].pop - populations[i].pop * effort * populations[i].q$
40: **end for**
41: **return** $prises$
42: **end function**
43:
44: **function** $reinit()$
45: **for** p in $populations$ **do**
46: $p.pop = p.pop\_init$
47: **end for**
48: **end function**

Algorithm 11 Agent pêcheur Partie 1

Variable *decideur* l'influenceur, le modèle décideur
liste_zones //liste d'influencés, la liste des zones de pêche que les pécheurs impactent
apprentissages // liste des objets de type Apprentissage
$last\_prises, prises, efforts$ //respectivement les prises précédentes, les prises courantes et les efforts appliqué sur chaque zone de pêche
temps_simulation, *etat*
function *init*()
temps_simulation $\leftarrow 100$
etat $\leftarrow$ "*init*"
popu $\leftarrow$ *initPopulation*() // initialisation de l'ensemble des populations a des zones de pêche
for all $p \in popu$ **do**
    p.quota $\leftarrow$ *optimisation.main*(*temps_simulation*, *p*) // calcul des quotas en faisant appel a l'interface optimisation
end for
return 0
end function

function *time_advance*()
if *etat* = "*simule*" **then**
    resultat $\leftarrow 0$
else
    if *etat* = "*init*" **then**
        resultat $\leftarrow 0$
    else
        resultat $\leftarrow \infty$
    end if
end if
return *resultat*
end function

function λ()
end function

function $\delta_{ext}(events)$ //dans le contexte utilisé il n'y a pas de transition externe.
end function

Algorithm 12 Agent pêcheur Partie 2 (Deuxième exemple)

```
1: function δ_int()
2: if state = "simule" then
3:    quotas ← decideur.quotas
4:    next_quotas ← decideur.next_quotas // Nous récupérons les quotas et quotas de l'année n+1 grâce à l'influenceur décideur
5:    if apprend = "true" then
6:       for i = 0 à liste_zones.size() do
7:          last_prises[i] ← prises[i]
8:          efforts[i] ← apprentissages[i].choix(quota, lastprise)
9:          prises[i] ← arrondi(liste_zones[i].peche(efforts[i]))
10:         apprentissages[i].reward(quotas, last_prises[i], prises[i], efforts[i], next_quotas)
11:      end for
12:   else
13:      for i = 0 à liste_zones.size() do
14:         last_prises[i] ← prises[i]
15:         efforts[i] ← choix(quota, lastprise)
16:         prises[i] ← arrondi(liste_zones[i].peche(efforts[i]))
17:      end for
18:   end if
19:   population.pop ← population.pop + (population.r * (1 − (population.pop/population.k)) * population.pop) − prises
20:   if temps > temps_simulation then
21:      state ← "wait
22:      if Apprentissage.run() then
23:         state ← "simule
24:         temps ← 0
25:         for i = 0 à liste_zones.size() do
26:            liste_zones[i].reinit() //on réinitialise la population de toutes les zones.
27:         end for
28:      else
29:         apprend ← "false"
30:      end if
31:   end if
32: end if
33: end function
```

Algorithm 13 Agent décideur (Deuxième exemple)

Variable $populations = \{\{pop, r, q, k, p_{msy}, \{Quota\}\}\}$ //définition de la liste des stocks contenant les paramètres nécessaires
$liste\_zones$ //liste d'influenceurs, la liste des zones de pêche que le décideur peut observer
$quotas, next\_quotas$ // liste des quotas pour l'année courante et pour l'année suivante
$temps\_simulation, etat$
function $init()$
$temps\_simulation \leftarrow 100$
$temps\_quota \leftarrow 2$
$etat \leftarrow$ "$init$"
$initPopulation()$ // initialisation de l'ensemble des populations a des zones de pêche
return 0
end function

function $time\_advance()$
if $etat =$ "$simule$" **then**
  $resultat \leftarrow 1$
else
  $resultat \leftarrow \infty$
end if
return $resultat$
end function

function $\lambda()$
end function

function $\delta_{int}()$
$agregStock()$ //fonction permettant d'agréger les données de chaque stock et de mettre à jour la variable $populations$
for all $p \in populations$ **do**
  $p.quota \leftarrow optimisation.main(temps\_quota, p)$ // calcul des quotas en faisant appel a l'interface optimisation
end for
end function

function $\delta_{ext}(events)$ //dans le contexte utilisé il n'y a pas de transition externe.
end function

3.3.3 Résultats et discussions

Objectif

La problématique est toujours la même, à savoir de proposer une approche complète de modélisation et de simulation de systèmes complexes adaptée à l'étude des pêcheries.

Le modèle que nous venons de présenter, bien que prenant en compte plus d'interactions que celui de

l'exemple 1 reste malgré tout un modèle théorique qui ne représente que trop partiellement la réalité de la pêcherie pour nous permettre de tirer des conclusions sur l'activité de pêche.

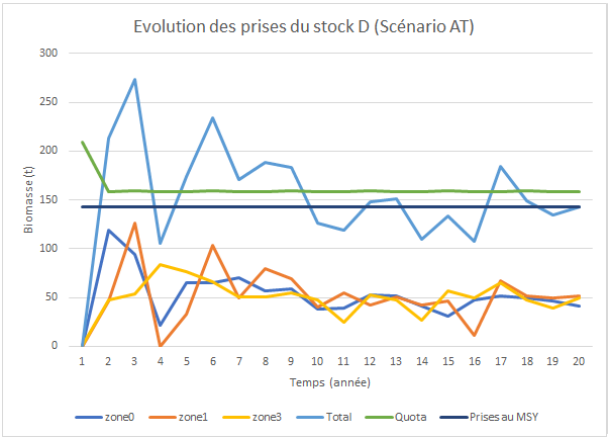
En revanche cet exemple nous a permis d'intégrer dans un seul et même modèle tous les principaux éléments que nous proposons dans le chapitre précédent. Son objectif principal est de montrer la capacité de notre approche à permettre aux agents de prendre des décisions cohérentes dans une situation plus complexe que dans le premier exemple.

Résultats des simulations

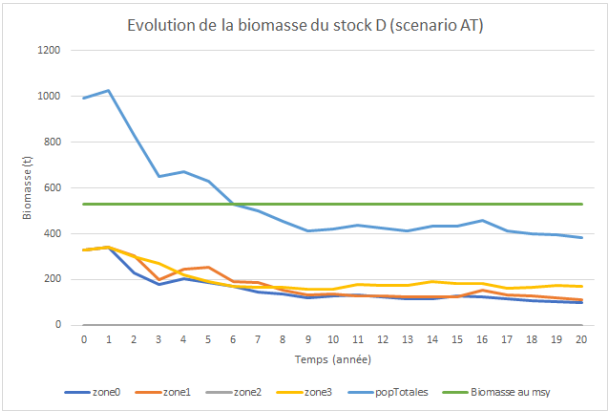
Comme dans l'exemple 1, nous allons tout d'abord rappeler les différents scénarios que nous avons simulés : *amende avec tolérance* (AT), *amende proportionnelle* (AP) et *espèces protégées* (EP).

Nous allons maintenant regarder plus en détail l'évolution des biomasses et des prises sur chaque zone de pêche au cours du temps. Regardons dans un premier temps le stock 1, nous rappelons que ce stock n'est pas présent dans la zone 2.

Les figures 3.6, 3.7 et 3.8, présentent l'évolution des prises et de la biomasse du stock D pour chaque scénario. Nous y avons également fait apparaître les prises ainsi que la biomasse au MSY à titre de comparaison.



(a) Évolution des prises.

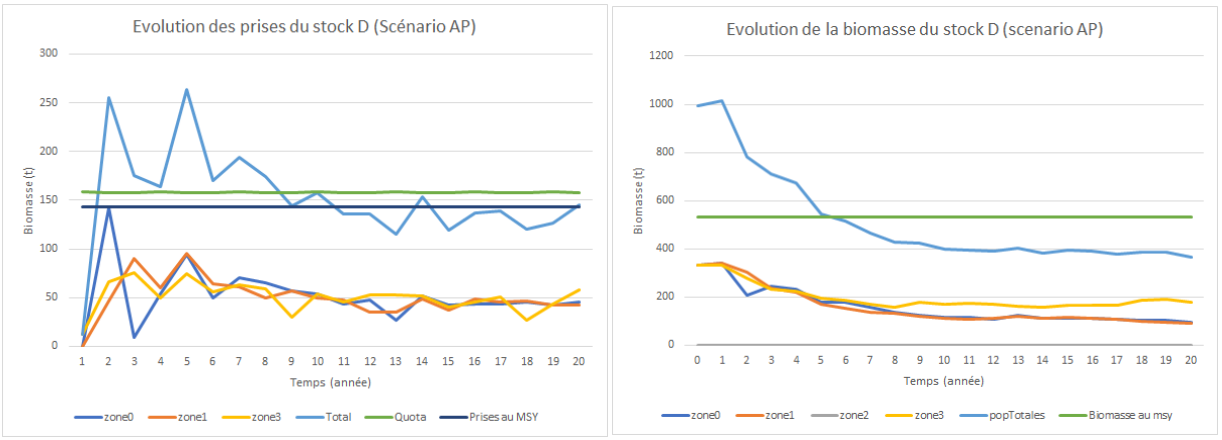


(b) Évolution de la population.

FIGURE 3.6 – Prises et populations du stock D pour le scénario AT (amende avec tolérance).

Sur la figure 3.6, nous observons le cas du scénario AT. Nous pouvons voir plusieurs pics des prises totales au dessus des quotas au début puis à partir du milieu de la simulation, les prises oscillent en restant

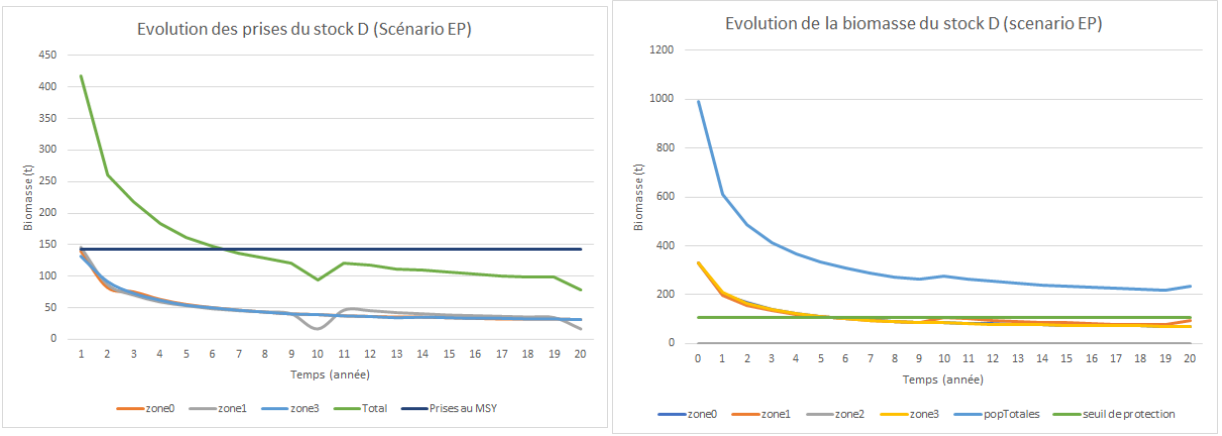
relativement proche du quota fixé. Du côté de la biomasse, nous observons une forte baisse globale en début de simulation, cohérente avec les pics de prises effectuées puis un équilibre semble se trouver un peu en dessous de la biomasse au MSY.



(a) Évolution des prises. (b) Évolution de la population.

FIGURE 3.7 – Prises et population du stock D pour le scénario AP (amande proportionelle).

Sur la figure 3.7, nous observons le cas du scénario AP les résultats sont très similaires.



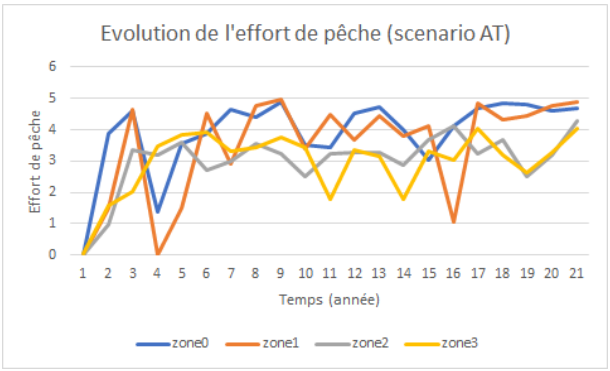
(a) Évolution des prises. (b) Évolution de la population.

FIGURE 3.8 – Prises et population du stock D pour le scénario EP (espèce protégée).

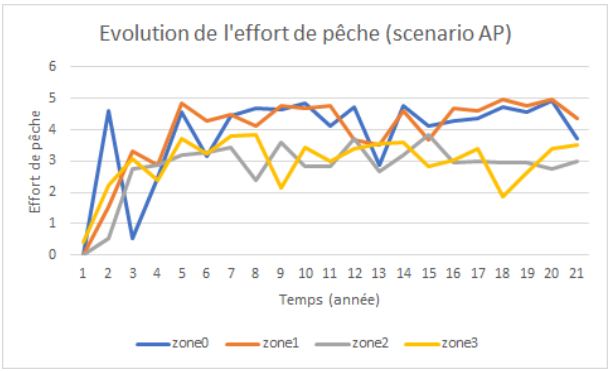
Nous observons sur la figure 3.8 le cas du scénario EP. Sur cette courbe, nous observons de fortes prises en début qui vont diminuer tout au long de la simulation. Ceci peut s'expliquer par le fait que n'étant pas contrains par un quota les pêcheurs appliquent un effort maximal. En effet si nous regardons

l'évolution de la biomasse, nous voyons que celle-ci diminue de la même manière sans atteindre néanmoins le seuil de protection. Nous pouvons donc penser que nos agents pêcheurs vont pêcher au maximum en restant au delà de ce seuil.

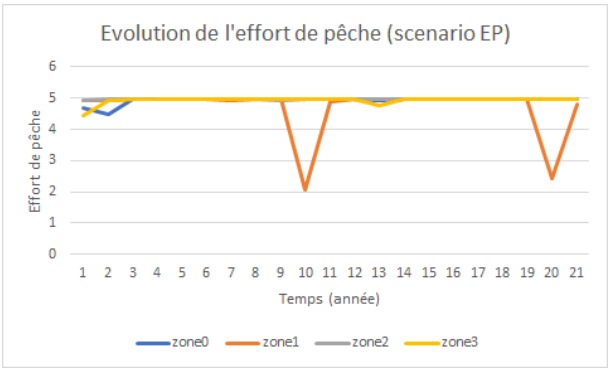
La figure 3.9, nous montre l'évolution de l'effort de pêche en fonction du scénario.



(a) Évolution de l'effort de pêche par zone pour le scénario AT.



(b) Évolution de l'effort de pêche par zone pour le scénario AP.



(c) Évolution de l'effort de pêche par zone pour le scénario EP.

FIGURE 3.9 – Évolution de l'effort de pêche par zone pour chaque scénario.

La figure 3.9a, illustre l'effort de pêche appliqué pour le scénario AT. Nous pouvons voir que les variations d'une année à l'autre peuvent être importantes. Néanmoins deux groupes se dégagent. En effet on y voit des similarités entre les zones 0 et 1 et entre les zones 2 et 3 ceci peut s'expliquer par le fait que les zones 0 et 1 contiennent les deux stocks et que les zones 2 et 3 contiennent chacune un seul stock.

La figure 3.9b dégage les mêmes tendances que la précédente, mais de façon plus claire. En effet nous observons ici une grande similitude entre les efforts de la zone 0 et 1. Ainsi qu'une stabilisation sur les dernières années de simulation.

La figure 3.9c présente l'évolution de l'effort pour le scénario EP. Elle nous confirme ce que nous avons précédemment supposé, c'est-à-dire que l'effort appliqué est maximum tant que le seuil n'est pas atteint.

Nos résultats nous permettent de penser que nos agents ont pris des décisions cohérentes au vu des contraintes que nous leur avons données. En effet, nos agents ont appris à respecter des quotas plus ou moins strictement en fonctions des scénarios et à pêcher au maximum lorsqu'ils n'étaient pas contraints.

Nous pouvons donc penser que notre approche fournit des résultats concluants. Néanmoins, il semble peu réaliste d'utiliser ces résultats pour influer une réelle gestion des pêcheries. Nous pouvons tout de même espérer utiliser notre approche pour développer un modèle plus représentatif de la réalité et capable de donner des indications pour améliorer la gestion des pêcheries en Corse.

3.4 Conclusion

Dans ce chapitre, nous avons proposé deux exemples d'application de notre approche dans un cas théorique de gestion des ressources halieutiques.

Pour cela nous avons commencé par décrire le modèle de croissance de population utilisé pour modéliser notre pêcherie. Au vu de notre situation et des données dont nous disposons, nous avons dû faire un choix de modèle à utiliser. Ce choix a été fait principalement dû aux données dont nous disposons actuellement. Le modèle que nous avons présenté nous semble cohérent avec les données que nous avons et nous a permis de tester différentes politiques de pêche basées sur des quotas. Mais il est aussi important de noter que le modèle utilisé ne peut pas toujours représenter la complexité de la réalité. Il sera donc intéressant par la suite de pouvoir complexifier le modèle ainsi que de le calibrer pour correspondre le plus possible à la réalité. Ainsi que de se pencher sur d'autres scénarios.

Le premier exemple est un modèle à 5 composants pour simuler des scénarios. Cela nous a permis de mettre en œuvre notre approche décisionnelle dans le cas d'une pêcherie. Nous avons choisi dans ce premier exemple de ne pas prendre en compte de dimensions spatiales pour nous concentrer sur l'aspect décision. Ceci nous a permis de faire interagir des agents qui ont pour objectif de prendre des décisions et de voir leurs effets. Les résultat nous permettent principalement de penser que les processus de décision mis en œuvre semblent cohérents avec ce que nous attendions. Ils nous permettent aussi d'avoir une vision des impacts que peuvent avoir les différents scénarios.

Le second exemple est un modèle à 6 composants pour simuler lui aussi des scénarios avec quotas,

avec prise en compte de l'espace et des migrations entre zones. Dans cet exemple, nous avons ajouté la prise en compte de la dimension spatiale, cela nous a permis de tester notre approche de décision mais également notre approche multicomposant. Ainsi nous avons pu nous assurer que la gestion de l'espace en utilisant cette approche reste cohérente. De plus nous avons, avec cet exemple, pu tester sur un seul et même modèle tous les principaux éléments que nous proposons dans le chapitre précédent. Les résultats obtenus permettent là aussi d'avoir une vison des effets possibles des scénarios que nous avons testés.

CONCLUSION ET PERSPECTIVES DE RECHERCHE

La problématique de ce travail est d'utiliser la simulation informatique comme outil permettant de faire des expériences virtuelles dans un objectif d'aide à la décision. Nous avons cherché, dans nos travaux, à replacer la simulation au centre d'une approche interdisciplinaire s'intéressant au fonctionnement des petits métiers de la pêche en Corse. Cette problématique principale, nous a amené à nous poser plusieurs questions.

Premièrement, comment représenter un système complexe bio-économique ? Nous avons choisi d'utiliser le paradigme agent pour modéliser ce système. En effet l'aspect interdisciplinaire du paradigme agent ainsi que sa capacité à pouvoir facilement exprimer dans un modèle des sous-composants hiérarchiquement organisés et capables d'interactions font qu'il se prête très bien à la modélisation des systèmes complexes bio-économiques.

Deuxièmement nous nous sommes penchés sur la robustesse de l'approche. Pour cela nous nous sommes rapprochés de la théorie de la modélisation et de la simulation proposée par *B.P. Zeigler* dans [Zeigler, 1976]. Ceci nous a amené à aborder d'autres problématiques telles que la représentation spatiale d'un modèle, l'articulation et la composition de modèles hétérogènes, ainsi que l'intégration dans des

modèles conventionnels de processus de décision.

Nous avons choisi pour socle unificateur le formalisme PDEVS. En effet, celui-ci, défini comme multi-formalisme par [Vangheluwe et al., 2002], permet d'inclure d'autres concepts. Ainsi, nous pouvons le compléter avec le paradigme agent, afin de bénéficier de ses avantages.

Enfin, pour respecter un cadre formel, indispensable au travail préalable de conceptualisation, nous utilisons la formalisation DPDEMAS proposée dans [Franceschini et al., 2017]. En effet, cette formalisation offre dans sa forme originale une analogie intéressante entre un Système Multi-Agents (SMA) et les modèles PDEVS et DSDE (version dynamique de PDEVS [Barros, 1998]). Nous y retrouvons une description de l'agent à partir d'un corps qui modélise sa représentation physique dans un environnement, ainsi qu'à partir d'un esprit qui est le lieu des prises de décision. Néanmoins DPDEMAS ne donne pas de description. Ceci nous a amené à nous pencher sur la problématique de la prise de décision. L'enjeu central était donc de proposer un modèle exécutable suffisamment générique pour permettre l'aide à la décision et de l'utiliser dans le cas de la modélisation d'une pêcherie.

Ainsi, nous y avons contribué en proposant une approche complète de modélisation et de simulation de systèmes complexes adaptée à l'étude des pêcheries.

Dans un chapitre d'état de l'art, nous avons présenté les principales définitions issues de la théorie de la modélisation et de la simulation (TM&S) informatique et des Systèmes Multi-Agents (SMA). À partir de ces définitions, nous avons proposé un *modèle conceptuel* permettant de décrire à partir du formalisme PDEVS et de l'approche Soar des agents cognitifs.

Dans le chapitre suivant, nous avons sélectionné VLE comme cadre d'implémentation et avons détaillé notre *modèle informatique*. Ce chapitre présente le coeur de nos travaux car il décrit notre approche multicomposant puis les moyens à disposition des agents pour utiliser des briques inspirées de Soar. La brique d'optimisation permet d'affiner les décisions des agents. La brique d'apprentissage par renforcement élargit les possibilités d'exploration de décisions des agents en leur permettant de compléter leurs connaissances sur une situation donnée.

Dans le dernier chapitre, nous proposons deux exemples d'applications. Le premier exemple est un modèle à 5 composants pour simuler des scénarios avec quotas. Le second exemple est un modèle à 6 composants pour simuler lui aussi des scénarios avec quotas, avec en plus la prise en compte de l'espace et des migrations entre zones.

Nous pouvons voir dans l'approche que nous proposons certains avantages. Notamment la capacité à

facilement exprimer dans un modèle des sous-composants hiérarchiquement organisés et capables d'interaction grâce au paradigme agent. De plus nous proposons une approche basée sur le formalisme PDEVS et la formalisation DPDEMAS. Ceci nous permet d'avoir une approche formelle qui limite les ambiguïtés de modélisation. Nous proposons un modèle informatique générique implémentant des mécanismes de prise de décision. Cette approche peut être appliquée dans un grand nombre de cas. Notre apport principal permet de décrire des agents cognitifs et permet l'utilisation de méthodes d'apprentissage liées à la fois à l'environnement, via les interactions, ainsi qu'aux expériences de l'agent. Ceci permettant à l'agent de construire une base de connaissance dans le but de prendre des décisions adaptées. Notre approche permet également à un agent de résoudre un problème en utilisant des méthodes d'optimisation issues d'une bibliothèque développée par notre équipe. Ceci dans un but d'autonomie et d'adaptabilité de nos agents.

Nous avons aussi, dans un souci d'évolutivité, donné une structure informatique qui facilite l'intégration de nouvelles méthodes de décision, et d'apprentissage. L'intégration de méthodes d'optimisation étant liée à l'évolution de la bibliothèque que nous utilisons.

Enfin, Le modèle que nous avons présenté nous semble cohérent avec les données que nous avons et nous a permis de tester différentes politiques de pêche basées sur des quotas.

Il reste néanmoins quelques inconvénients. L'approche de modélisation que nous proposons nécessite, pour être mise en place, de bonnes connaissances en programmation. Cela peut donc poser des problèmes d'appropriation aux non-informaticien. En effet le modèle exécutable final est dans la plupart des cas intrinsèquement lié au modèle développé. De plus nous n'avons implémenté que peu de méthodes ce qui obligera l'utilisateur à ajouter les méthodes les plus adaptées à sa situation pour tirer profit de notre approche.

Enfin, les deux exemples que nous avons présentés dans le chapitre précédent, n'ont pour objectif que de valider notre approche. Ces modèles sont purement théoriques et ne représentent pas suffisamment bien la réalité du système pour pouvoir être utilisés en l'état pour aider à la gestion des ressources halieutiques.

Les applications proposées ne tirent pas partie du caractère événementiel du formalisme DEVS car un pas de temps est fixé et nos simulations sont donc discrètes en temps et non dépendantes d'évènements. L'utilisation de DTSS aurait peut-être été suffisante. Nous pouvons également nous interroger sur la pertinence de mêler le formalisme DEVS et les approches d'optimisation ou d'apprentissage. Cette expérience ne sera pas suffisante pour trancher le débat mais est-ce pertinent d'adapter le formalise DEVS pour tous

les cas de figure ? C'est une question qui reste en suspens. Nos travaux ne portaient pas sur ce point nous ne l'avons donc pas abordé dans ce document. Mais si le formalisme DEVS a un intérêt indiscutable dans le processus de modélisation, dans un contexte d'optimisation ou d'apprentissage tel que nous les avons utilisés qui demande l'exécution d'un grand nombre de simulations, les lourdeurs du formaliste peuvent se faire ressentir.

Les perspectives que nous envisageons par la suite sont : d'une part l'implémentation de nouvelles méthodes de décision dans notre approche. De la même manière il sera intéressant d'intégrer d'autres types d'apprentissages pour permettre une plus grande flexibilité de nos agents cognitifs. En effet nous nous sommes concentrés sur des méthodes d'apprentissage par renforcement basées valeur pour nos besoins immédiats mais beaucoup d'autres méthodes existent et s'adaptent à différentes problématiques. Nous pouvons notamment parler de méthodes telles que *programmation dynamique*[5] (DP) [Munos, 2000], celles de Monte Carlo [Kalos and Whitlock, 2009] ou encore celle de *gradient de politique*[6] [Sutton et al., 1999] qui font parties des méthodes *basées politiques* Il pourra également être intéressant, de comparer des résultats obtenus et les performances de chacune de ces différentes méthodes.

D'autre part maintenant que nous savons que notre approche est cohérente, il sera également intéressant de l'utiliser pour développer un modèle de pêcherie plus représentatif de la réalité et ainsi permettre qu'il soit utilisable par les professionnels de la pêche. Ce qui impliquera également une phase de validation des résultats du modèle par observations. Pour finir, nous souhaiterions pouvoir appliquer notre approche à d'autres cas d'étude.

5. dynamic programming
6. Policy gradient

BIBLIOGRAPHIE

[Anderson, 2015] Anderson, E. D. (2015). Lessons from a career in fisheries science. *ICES Journal of Marine Science*, 72(8) :2169–2179.

[Andrighetto et al., 2007] Andrighetto, G., Conte, R., Turrini, P., and Paolucci, M. (2007). Emergence in the loop : Simulating the two way dynamics of norm innovation. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Angelini and Moloney, 2007] Angelini, R. and Moloney, C. L. (2007). Fisheries, ecology and modelling : An historical perspective. *Pan-American Journal of Aquatic Sciences*, 2(2) :75–85.

[Balbi et al., 2009] Balbi, J. H., Morandini, F., Silvani, X., Filippi, J. B., and Rinieri, F. (2009). A physical model for wildland fires. *Combustion and Flame*, 156(12) :2217–2230.

[Balci, 2011] Balci, O. (2011). How to successfully conduct large-scale modeling and simulation projects. In *Proceedings of the Winter Simulation Conference*, pages 176–182. Winter Simulation Conference.

[Balke and Gilbert, 2014] Balke, T. and Gilbert, N. (2014). How do agents make decisions ? a survey. *Journal of Artificial Societies and Social Simulation*, 17(4) :13.

[Barros, 1995] Barros, F. J. (1995). Dynamic structure discrete event system specification : A new formalism for dynamic structure modeling and simulation. In *Proceedings of the 27th Conference on Winter Simulation*, WSC '95, pages 781–785, Washington, DC, USA. IEEE Computer Society.

[Barros, 1997] Barros, F. J. (1997). Modeling formalisms for dynamic structure systems. *ACM Trans. Model. Comput. Simul.*, 7(4) :501–515.

[Barros, 1998] Barros, F. J. (1998). Abstract simulators for the dsde formalism. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 407–412. IEEE.

[Bellman, 1966] Bellman, R. (1966). Dynamic programming. *Science*, 153(3731) :34–37.

[Bertalanffy, 1969] Bertalanffy, L. V. (1969). General system theory : Foundations, development, applications.

[Bisgambiglia et al., 2017] Bisgambiglia, P.-A., Rossi, J.-L., Franceschini, R., Chatelon, F.-J., Rossi, L., Bisgambiglia, P., and Marcelli, T. (2017). Dimzal : A software tool to compute acceptable safety distance. *Open Journal of Forestry*, 7 :11–33.

[Boncœur et al., 2000] Boncœur, J., Fifas, S., and Le Gallic, B. (2000). Un modèle bioéconomique d'évalution du coût social des rejets au sein d'une pêcherie complexe. *Économie & prévision*, 143(2) :185–199.

[Boulding, 1956] Boulding, K. E. (1956). General systems theory—the skeleton of science. *Management science*, 2(3) :197–208.

[Bousquet et al., 1994] Bousquet, F., Cambier, C., and Morand, P. (1994). Distributed artificial intelligence and object-oriented modelling of a fishery. *Mathematical and computer modelling*, 20(8) :97–107.

[Bousquet and Le Page, 2004] Bousquet, F. and Le Page, C. (2004). Multi-agent simulations and ecosystem management : a review. *Ecological modelling*, 176(3-4) :313–332.

[Britten et al., 2017] Britten, G. L., Dowd, M., Kanary, L., and Worm, B. (2017). Extended fisheries recovery timelines in a changing environment. *Nature Communications*, 8(1) :1–7.

[Broersen et al., 2002] Broersen, J., Dastani, M., Hulstijn, J., and van der Torre, L. (2002). Goal generation in the boid architecture. *Cognitive Science Quarterly*, 2(3-4) :428–447.

[Bruskotter and Fulton, 2013] Bruskotter, J. T. and Fulton, D. C. (2013). The future of fishing : an introduction to the special issue. *Human dimensions of wildlife*, 18(5) :319–321.

[Béné et al., 2001] Béné, C., Doyen, L., and Gabay, D. (2001). A viability analysis for a bio-economic model. *Ecological economics*, 36(3) :385–396.

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systemes Complexes*. PhD thesis, xx.

[Cervenka and Trencansky, 2007] Cervenka, R. and Trencansky, I. (2007). *The Agent Modeling Language-AML : A Comprehensive Approach to Modeling Multi-Agent Systems.* Springer Science & Business Media.

[Chow and Zeigler, 1994] Chow, A. C. H. and Zeigler, B. P. (1994). Parallel devs : A parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th Conference on Winter Simulation*, WSC '94, pages 716–722, San Diego, CA, USA. Society for Computer Simulation International.

[Chwif et al., 2013] Chwif, L., Banks, J., de Moura Filho, J. P., and Santini, B. (2013). A framework for specifying a discrete-event simulation conceptual model. *Journal of Simulation*, 7(1) :50–60.

[Clark, 2010] Clark, C. W. (2010). *Mathematical bioeconomics : the mathematics of conservation*, volume 91. John Wiley & Sons.

[Clerc, 2012] Clerc, M. (2012). Beyond standard particle swarm optimisation. In *Innovations and Developments of Swarm Intelligence Applications*, pages 1–19. IGI Global.

[Coquillard and Hill, 1997] Coquillard, P. and Hill, D. R. (1997). Modélisation et simulation d'écosystèmesdes modèles déterministes aux simulations à événements discrets. Technical report, Masson.

[Corbara et al., 1993] Corbara, B., Drogoul, A., Fresneau, D., and Lalande, S. (1993). Simulating the sociogenesis process in ant colonies with manta. In *Toward A Practice of Autonomous Systems : Proceedings of the First European Conference on Artificial Life*, pages 224–235.

[Corum, 2014] Corum, J. P. (2014). *Using Pattern Oriented Modeling to Design and Validate Spatial Models : a Case Study in Agent-based Modeling.* PhD thesis, University of Southern California.

[David et al., 2002] David, N., Sichman, J. S., and Coelho, H. (2002). Towards an emergence-driven software process for agent-based simulation. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 89–104. Springer.

[Deffuant et al., 2002] Deffuant, G., Amblard, F., Weisbuch, G., and Faure, T. (2002). How can extremism prevail ? a study based on the relative agreement model. *Journal of Artificial Societies and Social Simulation*, 5(4) :27.

[Dignum et al., 2008] Dignum, F., Dignum, V., and Jonker, C. M. (2008). Towards agents for policy making. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 141–153. Springer.

[Dinu et al., 2012] Dinu, R., Stratulat, T., and Ferber, J. (2012). A Formal Model of Agent Interaction Based on MASQ. In *AMPLE'2012 : 2nd International Workshop on Agent-based Modeling for PoLicy Engineering*, Montpellier, France.

[Duboz et al., 2002] Duboz, R., Ramat, E., and Giambiasi, N. (2002). Utilisation du formalisme DEVS pour la spécification de systèmes d'agents réactifs. In *Dixièmes journées francophones sur les systèmes multi-agents*, pages 99–102, Lille, France.

[Duboz et al., 2006] Duboz, R., Versmisse, D., Quesnel, G., Muzy, A., and Ramat, E. (2006). Specification of dynamic structure discrete event multiagent systems. *Simulation Series*, 38(2) :103.

[Dudley, 2008] Dudley, R. G. (2008). A basis for understanding fishery management dynamics. *System Dynamics Review : The Journal of the System Dynamics Society*, 24(1) :1–29.

[Edmonds and Moss, 2004] Edmonds, B. and Moss, S. (2004). From kiss to kids–an 'anti-simplistic'modelling approach. In *International workshop on multi-agent systems and agent-based simulation*, pages 130–144. Springer.

[Erard and Déguénon, 1996] Erard, P.-J. and Déguénon, P. (1996). *Simulation par événements discrets*. PPUR presses polytechniques.

[Ferber, 1995a] Ferber, J. (1995a). *Les systèmes multi-agents : vers une intelligence collective*, volume 322. InterEditions, Paris.

[Ferber, 1995b] Ferber, J. (1995b). *Multi-agent systems : an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

[Ferber et al., 2003] Ferber, J., Gutknecht, O., and Michel, F. (2003). From agents to organizations : an organizational view of multi-agent systems. In *International workshop on agent-oriented software engineering*, pages 214–230. Springer.

[Ferber and Müller, 1996] Ferber, J. and Müller, J.-P. (1996). Influences and reaction : a model of situated multiagent systems. In *Proceedings of second international conference on multi-agent systems (ICMAS-96)*, pages 72–79.

[Ferber and Perrot, 1995] Ferber, J. and Perrot, J.-F. (1995). *Les systèmes multi-agents : vers une intelligence collective*. InterEditions.

[Filippi and Bisgambiglia, 2004] Filippi, J.-B. and Bisgambiglia, P. (2004). Jdevs : an implementation of a devs based formal framework for environmental modelling. *Environmental Modelling & Software*, 19(3) :261–274.

[Foures et al., 2018] Foures, D., Franceschini, R., Bisgambiglia, P.-A., and Zeigler, B. P. (2018). multip-devs : A parallel multicomponent system specification formalism. *Complexity*, 2018.

[Fournier et al., 1998] Fournier, D. A., Hampton, J., and Sibert, J. R. (1998). Multifan-cl : a length-based, age-structured model for fisheries stock assessment, with application to south pacific albacore, thunnus alalunga. *Canadian Journal of Fisheries and Aquatic Sciences*, 55(9) :2105–2116.

[Franceschini, 2017] Franceschini, R. (2017). *Approche formelle pour la modélisation et la simulation à évènements discrets de systèmes multi-agents.* PhD thesis, xx.

[Franceschini et al., 2017] Franceschini, R., Bisgambiglia, P.-A., and Bisgambiglia, P. (2017). Approche formelle pour la modélisation et la simulation de systèmes multi-agents. In *Vingt-cinquièmes Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2017), Caen, France*, page 22.

[Franceschini et al., 2014] Franceschini, R., Bisgambiglia, P.-A., Touraille, L., Bisgambiglia, P., and Hill, D. (2014). A survey of modelling and simulation software frameworks using discrete event system specification. In *2014 Imperial College Computing Student Workshop*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Franceschini and Duboz, 2020] Franceschini, R. and Duboz, R. (2020). Système multi-agents et devs, une petite histoire commune. In *Convergences entre la théorie de la modélisation et de la simulation et les systèmes multi-agents.*

[Gaertner and Dreyfus-Leon, 2004] Gaertner, D. and Dreyfus-Leon, M. (2004). Analysis of non-linear relationships between catch per unit effort and abundance in a tuna purse-seine fishery simulated with artificial neural networks. *ICES Journal of Marine Science*, 61(5) :812–820.

[Garcia et al., 2017] Garcia, D., Sánchez, S., Prellezo, R., Urtizberea, A., and Andrés, M. (2017). Flbeia : A simulation model to conduct bio-economic evaluation of fisheries management strategies. *SoftwareX*, 6 :141–147.

[Gelcich and Donlan, 2015] Gelcich, S. and Donlan, C. J. (2015). Incentivizing biodiversity conservation in artisanal fishing communities through territorial user rights and business model innovation. *Conservation biology*, 29(4) :1076–1085.

[Grimm, 1999] Grimm, V. (1999). Ten years of individual-based modelling in ecology : what have we learned and what could we learn in the future ? *Ecological modelling*, 115(2-3) :129–148.

[Gulland et al., 1970] Gulland, J. A. et al. (1970). The fish resources of the oceans.

[Haugeland, 1989] Haugeland, J. (1989). *Artificial Intelligence : The Very Idea*. MIT Press. A Bradford Book.

[Idda et al., 2020] Idda, C., Innocenti, E., Prunetti, D., and Gonsolin, P.-R. (2020). Agent-based multi-component spatial simulation of a fishery. In *Proceedings of the 2020 Summer Simulation Conference*, pages 1–12.

[IFREMER, 2019] IFREMER (2019). Les ressources halieutiques françaises : bilan 2018. `https://wwz.ifremer.fr/content/download/124503/file/DP_halieutique_ifremer.pdf`.

[Innocenti et al., 2016] Innocenti, E., Urbani, D., Bisgambiglia, P.-A., and Gonsolin, P.-R. (2016). A multicomponent modeling approach for fishery simulations. In *2016 IEEE International Conference on Agents (ICA)*, pages 108–109. IEEE.

[Jager and Janssen, 2002] Jager, W. and Janssen, M. (2002). The need for and development of behaviourally realistic agents. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 36–49. Springer.

[Jerry, 1984] Jerry, B. (1984). Discrete-event system simulation. 1984.

[Jiang and Vidal, 2006] Jiang, H. and Vidal, J. M. (2006). From rational to emotional agents. In *Proceedings of the AAAI Workshop on Cognitive Modeling and Agent-based Social Simulation*.

[Judson, 1994] Judson, O. P. (1994). The rise of the individual-based model in ecology. *Trends in ecology & evolution*, 9(1) :9–14.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning : A survey. *Journal of artificial intelligence research*, 4 :237–285.

[Kalos and Whitlock, 2009] Kalos, M. H. and Whitlock, P. A. (2009). *Monte carlo methods*. John Wiley & Sons.

[Kim and Kim, 2000] Kim, J.-H. and Kim, T. G. (2000). Framework for modeling/simulation of mobile agent systems. In *Proceedings of 2000 Conference on AI, Simulation and Planning in High Autonomy Systems*, pages 53–59. Citeseer.

[Kim et al., 2009] Kim, S., Sarjoughian, H. S., and Elamvazhuthi, V. (2009). Devs-suite : a simulator supporting visual experimentation design and behavior monitoring. In *Proceedings of the 2009 Spring Simulation Multiconference*, page 161. Society for Computer Simulation International.

[Koeck et al., 2015] Koeck, B., Gérigny, O., Durieux, E. D. H., Coudray, S., Garsi, L.-H., Bisgambiglia, P.-A., Galgani, F., and Agostini, S. (2015). Connectivity patterns of coastal fishes following different dispersal scenarios across a transboundary marine protected area (bonifacio strait, nw mediterranean). *Estuarine, Coastal and Shelf Science*, 154 :234–247.

[Kofman and Junco, 2001] Kofman, E. and Junco, S. (2001). Quantized-state systems : a devs approach for continuous system simulation. *Transactions of The Society for Modeling and Simulation International*, 18(3) :123–132.

[Konda and Tsitsiklis, 2000] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[Laird, 2008] Laird, J. E. (2008). Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, 171 :224.

[Laird, 2012] Laird, J. E. (2012). *The Soar cognitive architecture*. MIT press.

[Laird et al., 1986] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1986). Soar : An architecture for general intelligence. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

[Lawson et al., 2000] Lawson, B. G., Park, S., et al. (2000). Asynchronous time evolution in an artificial society model. *Journal of Artificial Societies and Social Simulation*, 3(1) :1–2.

[Le Manach and Pauly, 2015] Le Manach, F. and Pauly, D. (2015). Update of the fisheries catch reconstruction of corsica (france), 1950–2010. *Fisheries Centre Working Paper*, 33 :5.

[Le Manacha et al., 2011] Le Manacha, F., Durab, D., Perecd, A., Riutorte, J.-J., Lejeunec, P., Santonif, M.-C., Culiolif, J.-M., and Paulyg, D. (2011). Preliminary estimate of total marine fisheries catches in corsica. *Fisheries Centre Research Reports*, 19(3).

[Lebiere et al., 2013] Lebiere, C., Pirolli, P., Thomson, R., Paik, J., Rutledge-Taylor, M., Staszewski, J., and Anderson, J. R. (2013). A functional model of sensemaking in a neurocognitive architecture. *Computational intelligence and neuroscience*, 2013 :5.

[Leslie, 1945] Leslie, P. H. (1945). On the use of matrices in certain population mathematics. *Biometrika*, 33(3) :183–212.

[Maravelias et al., 2014] Maravelias, C. D., Pantazi, M., and Maynou, F. (2014). Fisheries management scenarios : trade-offs between economic and biological objectives. *Fisheries management and ecology*, 21(3) :186–195.

[Martelloni et al., 2019] Martelloni, P.-H., Poiron-Guidoni, N., Bisgambiglia, P.-A., and Bisgambiglia, P. (2019). Poster : Comparaison entre optimisation et q-learning : application pour l'aide à la décision de la pêche en corse. In *JFPDA2019*, Toulouse, France.

[Martelloni et al., 2018] Martelloni, P.-H., Quesnel, G., Innocenti, E., Bisgambiglia, P.-A., Gonsolin, P. R., and Bisgambiglia, P. (2018). Component-based simulation for spatial complex systems in vle environment. In *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences*, pages 1–12.

[Mattei et al., 2006] Mattei, M., Notton, G., Cristofari, C., Muselli, M., and Poggi, P. (2006). Calculation of the polycrystalline pv module temperature using a simple method of energy balance. *Renewable energy*, 31(4) :553–567.

[Mattei et al., 2012] Mattei, S., Bisgambiglia, P.-A., Delhom, M., and Vittori, E. (2012). Towards discrete event multi agent platform specification. In *Proceedings in the Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking*, pages 14–21. Citeseer.

[McAllester, 1985] McAllester, D. A. (1985). A new procedure for growing min-max trees. *Artificial Intelligence*.

[Michel, 2007a] Michel, F. (2007a). The irm4s model : the influence/reaction principle for multiagent based simulation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3.

[Michel, 2007b] Michel, F. (2007b). Le modèle irm4s. de l'utilisation des notions d'influence et de réaction pour la simulation de systèmes multi-agents. *Revue d'intelligence artificielle*, 21(5-6) :757–779.

[Michel, 2015] Michel, F. (2015). Approches environnement-centrées pour la simulation de systèmes multi-agents : Pour un déplacement de la complexité des agents vers l'environnement. *Habilitation à Diriger des Recherches*.

[Michie, 1961] Michie, D. (1961). Trial and error. *Science Survey, Part*, 2 :129–145.

[Mitroff et al., 1974] Mitroff, I. I., Betz, F., Pondy, L. R., and Sagasti, F. (1974). On managing science in the systems age : two schemas for the study of science as a whole systems phenomenon. *Interfaces*, 4(3) :46–58.

[Munos, 2000] Munos, R. (2000). A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40(3) :265–299.

[Nothias et al., 2020] Nothias, L.-F., Petras, D., Schmid, R., Dührkop, K., Rainer, J., Sarvepalli, A., Protsyuk, I., Ernst, M., Tsugawa, H., Fleischauer, M., et al. (2020). Feature-based molecular networking in the gnps analysis environment. *Nature Methods*, 17(9) :905–908.

[Odell et al., 2002] Odell, J. J., Parunak, H. V. D., Fleischer, M., and Brueckner, S. (2002). Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer.

[Oussalah et al., 1989] Oussalah, C., Santucci, J., Giambiasi, N., and Roux, P. (1989). Expert system based on multi-view/multi-level model approach for test pattern generation. In *Knowledge-Based System Diagnosis, Supervision, and Control*, pages 81–101. Springer.

[Parunak et al., 1998] Parunak, H. V. D., Savit, R., and Riolo, R. L. (1998). Agent-based modeling vs. equation-based modeling : A case study and users' guide. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 10–25. Springer.

[Pavé, 2012] Pavé, A. (2012). *Modélisation des systèmes vivants, de la cellule à l'écosystème*. Lavoisier, hermes science.

[PDMIA, 2008] PDMIA, G. (2008). Processus décisionnels de markov en intelligence artificielle. *Edité par Olivier Buffet et Olivier Sigaud*, 1.

[Pella and Tomlinson, 1969] Pella, J. J. and Tomlinson, P. K. (1969). A generalized stock production model. *Inter-American Tropical Tuna Commission Bulletin*, 13(3) :416–497.

[Pereira et al., 2005] Pereira, D., Oliveira, E., Moreira, N., and Sarmento, L. (2005). Towards an architecture for emotional bdi agents. In *2005 portuguese conference on artificial intelligence*, pages 40–46. IEEE.

[Phan, 2014] Phan, D. (2014). *La modélisation à base d'agents et la simulation par systèmes multi-agents de sociétés d'agents intentionnels*. Éditions Matériologiques.

[Pidd, 2010] Pidd, M. (2010). Why modelling and model use matter. *Journal of the operational Research Society*, 61(1) :14–24.

[Poiron-Guidoni, 2018] Poiron-Guidoni, N. (2018). Bibliothèque d'optimisation générique multi-objectif sous contraintes, implémentation et comparaison des méthodes sur benchmarket problème réel. Master's thesis, Université de Corse, Corte, France. (Rapport de stage).

[Poiron-Guidoni et al., 2020] Poiron-Guidoni, N., Bisgambiglia, P.-A., and Bisgambiglia, P. (2020). A probabilistic optimization approach to deal with uncertainties in model calibration. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

[Puterman, 2014] Puterman, M. L. (2014). *Markov decision processes : discrete stochastic dynamic programming*. John Wiley & Sons.

[Quesnel, 2006] Quesnel, G. (2006). Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes. *PHD trésis Laboratoire d'Informatique du Littoral (LIL). Calais-France.*

[Quesnel et al., 2009] Quesnel, G., Duboz, R., and Ramat, É. (2009). The virtual laboratory environment–an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4) :641–653.

[Quinn, 2003] Quinn, T. J. (2003). Ruminations on the development and future of population dynamics models in fisheries. *Natural Resource Modeling*, 16(4) :341–392.

[R. Duboz et al., 2005] R. Duboz, D. Versmisse, G. Quesnel, A. Muzzy, and E. Ramat (2005). Specification of Dynamic Structure Discret event Multiagent Systems. In *Agent-Directed Simulation (ADS 2006)*, Huntsville, AL, USA,. SCS.

[Ramat, 2003] Ramat, E. (2003). Contributions à la modélisation et à la simulation des systèmes complexes. *Habilitation à diriger des recherches.*

[Rao and Georgeff, 1991] Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. *KR*, 91 :473–484.

[Raykar and Agrawal, 2014] Raykar, V. and Agrawal, P. (2014). Sequential crowdsourced labeling as an epsilon-greedy exploration in a Markov Decision Process. In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 832–840, Reykjavik, Iceland. PMLR.

[Rivest, 1987] Rivest, R. L. (1987). Game tree searching by min/max approximation. *Artificial Intelligence*, 34(1) :77–96.

[Robinson, 2013] Robinson, S. (2013). Conceptual modeling for simulation. In *Simulation Conference (WSC), 2013 Winter*, pages 377–388. IEEE.

[Robinson, 2015] Robinson, S. (2015). A tutorial on conceptual modeling for simulation. In *Proceedings of the 2015 Winter Simulation Conference*, pages 1820–1834. IEEE Press.

[Rosenbloom et al., 1993] Rosenbloom, P. S., Laird, J., and Newell, A. (1993). *The SOAR papers : Research on integrated intelligence*. Mit Press Cambridge, MA.

[Rummery and Niranjan, 1994] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England.

[Russell et al., 2003] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial intelligence : a modern approach*, volume 2. Prentice hall Upper Saddle River.

[Saunier, 2015] Saunier, J. (2015). De l'intérêt de la cognition incarnée pour les agents logiciels. In *23èmes Journées Francophones sur les Systèmes Multi-Agents*, pages 101–110, Rennes, France.

[Schaefer, 1954] Schaefer, M. B. (1954). Some aspects of the dynamics of populations important to the management of the commercial marine fisheries. *Inter-American Tropical Tuna Commission Bulletin*, 1(2) :23–56.

[Schaefer, 1957] Schaefer, M. B. (1957). Some considerations of population dynamics and economics in relation to the management of the commercial marine fisheries. *Journal of the Fisheries Board of Canada*, 14(5) :669–681.

[Schtickzelle, 1981] Schtickzelle, M. (1981). Pierre-françois verhulst (1804-1849). la première découverte de la fonction logistique. *Population (french edition)*, pages 541–556.

[Seo et al., 2013] Seo, C., Zeigler, B. P., Coop, R., and Kim, D. (2013). Devs modeling and simulation methodology with ms4 me software tool. In *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium*, page 33. Society for Computer Simulation International.

[Sharp et al., 1983] Sharp, G. D., Csirke, J., Garcia, S., et al. (1983). Modelling fisheries : what was the question. *Proceedings of the Expert Consultations to Examine Changes in Abundance and Species Composition of Neritic Fisheries Resources*, pages 1177–1214.

[Siarry, 2014] Siarry, P. (2014). *Métaheuristiques*. Editions Eyrolles.

[Siegfried, 2014] Siegfried, R. (2014). "agent-based modeling and simulation". In Siegfried, R., editor, *"Modeling and Simulation of Complex Systems"*, chapter 2. "Springer".

[Silvestri and Maynou, 2009] Silvestri, S. and Maynou, F. (2009). Application of a bioeconomic model for supporting the management process of the small pelagic fishery in the veneto region, northern adriatic sea, italy. *Scientia Marina*, 73(3) :563–572.

[Simon, 1990] Simon, H. A. (1990). Prediction and prescription in systems modeling. *Operations Research*, 38(1) :7–14.

[Soulié, 2001] Soulié, J.-C. (2001). *Vers une approche multi-environnements pour les agents*. PhD thesis, Université de la Réunion, 2001., Université de la Rénunion.

[Soulié, 2012] Soulié, J.-C. (2012). Multi-modélisation & Simulation de Systemes Complexes : de la théorie à l'application. *Habilitation à Diriger des Recherches*. 00000.

[Stewart, 2013] Stewart, I. (2013). *Les mathématiques du vivant : ou La clef des mystères de l'existence*. Flammarion.

[Strehl and Littman, 2004] Strehl, A. L. and Littman, M. L. (2004). An empirical evaluation of interval estimation for markov decision processes. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 128–135. IEEE.

[Sun et al., 2003] Sun, R., Zhang, X., et al. (2003). Accessibility versus action-centeredness in the representation of cognitive skills. In *Proceedings of the fifth international conference on cognitive modeling*, pages 195–200.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1) :9–44.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning : An introduction*. MIT press.

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[Sutton et al., 1999] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

[Troccoli and Wainer, 2003] Troccoli, A. and Wainer, G. (2003). Implementing parallel cell-devs. In *Proceedings of the 36th annual symposium on Simulation*, page 273. IEEE Computer Society.

[Tsikliras and Froese, 2019] Tsikliras, A. C. and Froese, R. (2019). Maximum sustainable yield. *Encyclopedia of Ecology,*, 1 :108–115.

[Uhrmacher, 2001] Uhrmacher, A. M. (2001). Dynamic structures in modeling and simulation : a reflective approach. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 11(2) :206–232.

[Uhrmacher and Arnold, 1994] Uhrmacher, A. M. and Arnold, R. (1994). Distributing and maintaining knowledge : agents in variable structure environments. In *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*, pages 178–184. IEEE Comput. Soc. Press.

[Uhrmacher and Schattenberg, 1998] Uhrmacher, A. M. and Schattenberg, B. (1998). Agents in Discrete Event Simulation. In International, S. f. C. S., editor, *Proceedings of ESS98*.

[Urbani, 2006] Urbani, D. (2006). *Elaboration d'une approche hybride SMA-SIG pour la définition d'un système d'aide à la décision ;application à la gestion de l'eau. (Development of a new MAS GIS hybrid approach for Decision Support Systems ; application to water management).* PhD thesis, University of Corsica Pasquale Paoli, Corte, France.

[Van Tendeloo and Vangheluwe, 2017] Van Tendeloo, Y. and Vangheluwe, H. (2017). An evaluation of devs simulation tools. *Simulation*, 93(2) :103–121.

[Vangheluwe et al., 2002] Vangheluwe, H., De Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal*, pages 9–20. 00166.

[von Bertalanffy and Sutherland, 1974] von Bertalanffy, L. and Sutherland, J. W. (1974). General systems theory : Foundations, developments, applications. *IEEE Transactions on Systems, Man, and Cybernetics*, -(6) :592–592.

[Von Hanxleden et al., 2012] Von Hanxleden, R., Lee, E. A., Motika, C., and Fuhrmann, H. (2012). Multi-view modeling and pragmatics in 2020. In *Monterey Workshop*, pages 209–223. Springer.

[Von NEUMAN, 1966] Von NEUMAN, J. (1966). The theory of self-reproducing automata. arthur burks ed.

[Voyant et al., 2017] Voyant, C., Notton, G., Kalogirou, S., Nivet, M.-L., Paoli, C., Motte, F., and Fouilloy, A. (2017). Machine learning methods for solar radiation forecasting : A review. *Renewable Energy*, 105 :569–582.

[Wainer et al., 2011] Wainer, G., Glinsky, E., and Gutierrez-Alcaraz, M. (2011). Studying performance of devs modeling and simulation environments using the devstone benchmark. *Simulation*, 87(7) :555–580.

[Wainer, 2000] Wainer, G. A. (2000). Improved cellular models with parallel cell-devs. *TRANSACTIONS*, 17(2).

[Wainer, 2014] Wainer, G. A. (2014). Cellular modeling with cell-devs : A discrete-event cellular automata formalism. In *International Conference on Cellular Automata*, pages 6–15. Springer.

[Wainer, 2018] Wainer, G. A. (2018). Advanced cell-devs modeling applications : a legacy of norbert giambiasi. *Simulation*, page 0037549718761596.

[Wainer and Giambiasi, 2001] Wainer, G. A. and Giambiasi, N. (2001). Application of the cell-devs paradigm for cell spaces modelling and simulation. *Simulation*, 76(1) :22–39.

[Wainer and Giambiasi, 2005] Wainer, G. A. and Giambiasi, N. (2005). Cell-devs/gdevs for complex continuous systems. *Simulation*, 81(2) :137–151.

[Walters et al., 2005] Walters, C. J., Christensen, V., Martell, S. J., and Kitchell, J. F. (2005). Possible ecosystem impacts of applying msy policies from single-species assessment. *ICES Journal of Marine Science*, 62(3) :558–568.

[Watkins and Dayan, 1992] Watkins, C. and Dayan, P. (1992). gtechnical note : Q-learning,• h machine learning, vol. 8. .

[Watkins, 1989] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[Weyns et al., 2007] Weyns, D., Omicini, A., and Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1) :5–30.

[Wilson, 2002] Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin & Review*, 9(4) :625–636.

[Wooldridge, 2002] Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. Wiley and Sons, Chichester, West Sussex, Angleterre, wiley and sons edition. 07648.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents : Theory and practice. *The knowledge engineering review*, 10(02) :115–152.

[Yanıkoğlu et al., 2019] Yanıkoğlu, İ., Gorissen, B. L., and den Hertog, D. (2019). A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3) :799–813.

[Zeigler, 1976] Zeigler, B. P. (1976). Theory of modelling and simulation. *Interscience. Jhon Wiley& Sons, New York*.

[Zeigler et al., 2018] Zeigler, B. P., Muzy, A., and Kofman, E. (2018). *Theory of modeling and simulation : discrete event & iterative system computational foundations*. Academic press.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems*. Academic press.

TABLE DES FIGURES

1 Représentation du système complexe inspiré de [Idda et al., 2020]. 15

1.1 Le processus de modélisation itératif en quatre étapes que nous proposons de suivre. Ce processus est en partie issus des travaux précurseurs de [Siegfried, 2014, Chwif et al., 2013, Robinson, 2013]. 22

1.2 Le processus de simulation d'un système complexe. 28

1.3 Principe de la simulation à évènements discrets. 29

1.4 Principe de la simulation en temps discret. 30

1.5 Représentation schématique d'un SMA. 31

1.6 Évolution dans le temps des états d'un modèle DEVS. 39

1.7 Simulation d'un modèle PDEVS. 41

1.8 Illustration d'un *modèle conceptuel multicomposant*. 45

1.9 Positionnement dans l'existant [Franceschini and Duboz, 2020] 47

1.10 Concepts généraux de l'approche proposée. 50

1.11 L'architecture cognitive *Soar* d'un agent décisionnel. 52

1.12 Concepts généraux. 54

1.13 Principe d'apprentissage par renforcement. 57

1.14 Principe d'apprentissage par renforcement 59

2.1 Modèle PDDEVS. 66

2.2 Modèle multicomposant. 66
2.3 Diagramme de classe schématisant le cœur de VLE 2.0 67
2.4 Classe Dynamics de VLE 69
2.5 Principaux éléments de modélisation. 70
2.6 Principaux éléments de simulation. 71
2.7 Comparaison des algorithmes de simulation de notre exemple. 72
2.8 schémas UML utilisé par l'approche proposée extrait de [Franceschini, 2017] 76
2.9 Algorithme d'apprentissage 78
2.10 Processus d'optimisation 83
2.11 Diagramme de séquences du processus d'apprentissage. 87
2.12 Représentation générale de la bibliothèque d'optimisation issue de [Poiron-Guidoni, 2018]. 89
2.13 Diagramme de séquence du processus d'optimisation 91
2.14 Jeu de morpion 91
2.15 modélisation DEVS du jeu de morpion 92
2.16 Classe du modèle atomique Jeu 93
2.17 Classe de l'agent Joueur 93
2.18 modélisation DEVS du jeu de morpion 94
2.19 Algorithme d'apprentissage du morpion 95
2.20 Arbre de possibilité de l'algorithme *MinMax* 96
2.21 Apprentissage du morpion par deux agents 97
2.22 Apprentissage du morpion par un agent face à un agent utilisant l'algorithme *MinMax* . . 98

3.1 Comparaison de l'évolution des prises réelles et simulées de denti 107
3.2 Modèle non spatialisé 109
3.3 Comparaison entre les prises et les quotas pour le stock 0. 121
3.4 Modèle spatialisé 123
3.5 Agent pecheur intelligent 125
3.6 Prises et populations du stock D pour le scénario AT (amende avec tolérance). 131
3.7 Prises et population du stock D pour le scénario AP (amande proportionelle). 132
3.8 Prises et population du stock D pour le scénario EP (espèce protégée). 132

3.9 Évolution de l'effort de pêche par zone pour chaque scénario. 133

LISTE DES TABLEAUX

1.1 Classification des formalismes, proposée par [Ramat, 2003] en fonction des caractéristiques de l'état du modèle, de l'espace et du temps. 38
1.2 Comparaison entre les méthodes *basées valeurs* et *basées politique*. 58

2.1 Tableau récapitulatif des différents framework DEVS 64
2.2 Comparaison des temps moyen de calcul en secondes. 74

3.1 Bornes des paramètres 106
3.2 Paramètres utilisés. 107

LISTE DES ALGORITHMES

1 Fonction apprentissage 86
2 Fonction choix 86
3 fonction main d'utilisation de l'optimisation 90
4 Classe Q-learning 100
5 Modèle décideur 113
6 Optimisation 114
7 Modèle Pêcherie (partie 1) 117
8 Modèle Pecherie (partie 2) 118
9 Apprentissage 119
10 Agent zone (Deuxième exemple) 127
11 Agent pêcheur Partie 1 128
12 Agent pêcheur Partie 2 (Deuxième exemple) 129
13 Agent décideur (Deuxième exemple) 130