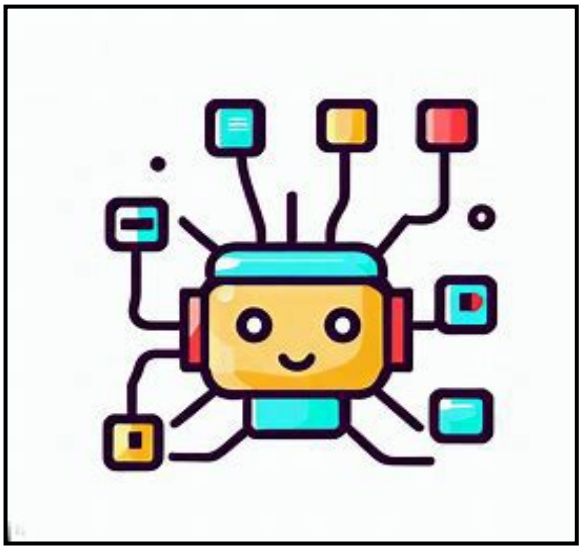
Génération de comportements variables induits par des modèles cognitifs modulaires, appliqué à une équipe virtuelle en situation de crise, dans un contexte de formation

Tristan De Blauwe

Par **Tristan DE BLAUWE**

Génération de comportements variables induits par des modèles cognitifs modulaires, appliqué à une équipe virtuelle en situation de crise, dans un contexte de formation

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 29 mars 2023
Spécialité : Sciences et Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)
D2732

Thèse
pour l'obtention du grade de

Docteur de l'université de technologie de Compiègne

Spécialité : Sciences et Technologies de l'Information et des Systèmes

Génération de comportements variables induits par des modèles cognitifs modulaires, appliqué à une équipe virtuelle en situation de crise, dans un contexte de formation

par

Tristan DE BLAUWE

Soutenue le 29 mars 2023 devant un jury composé de :

M. PICAULT Sébastien *Rapporteur*	Chargé de recherche, **INRAE** *Nantes*
M. VERCOUTER Laurent *Rapporteur*	Professeur, **LITIS** *INSA Rouen Normandie*
Mme. DUGDALE Julie *Examinatrice*	Professeure, **IMAG** *Université de Grenoble Alpes*
Mme. GOURANTON Valérie *Examinatrice*	Maître de conférence (HDR), **IRISA & INRIA** *INSA Rennes*
M. LAGRUE Sylvain *Examinateur*	Professeur, **Heudiasyc** *Université technologie de Compiègne*
Mme. LOURDEAUX Domitile *Directrice*	Professeuse, **Heudiasyc** *Université technologie de Compiègne*
M. SABOURET Nicolas *Directeur*	Professeur, **LISN** *Université Paris-Saclay*

Résumé

Les recherches effectuées lors de cette thèse traitent de la **génération de comportements d'agents virtuels**. C'est-à-dire de la construction de modèles informatiques pour déterminer les actions qu'un agent doit effectuer au sein d'un environnement virtuel, de manière à reproduire les actions et les réactions d'un humain dans la situation simulée. Nous appliquons notamment ces travaux dans le contexte de la formation professionnelle [Barot, 2014, Huguet et al., 2016, Silverman et al., 2012] pour entraîner des opérateurs lors de situations de crise.

La littérature en informatique propose de nombreux modèles d'agent qui intègrent des dimensions humaines dans le fonctionnement de l'agent, comme les émotions, en combinant certains modèles cognitifs issus de la littérature des Sciences Humaines et Sociales (SHS).

Notre objectif est de faciliter la conception de tels modèles informatiques. Nous souhaitons une approche **générique** facilitant l'intégration **modulaire** de modèles cognitifs dans des modèles d'agents quelconques et ceux pour générer des comportements **représentatifs**, **sensibles** et **intelligibles**.

Nous proposons un méta-modèle qui abstrait un modèle d'agent sous forme de cinq concepts : **O**peration, **P**erception, **A**ction, **C**aractéristiques et **K**nowledge (connaissances). Les modèles cognitifs peuvent affecter modulairement ces cinq concepts, grâce à leur composabilité. Nous posons aussi le moins d'hypothèses possibles sur le modèle d'agent et les modèles cognitifs.

Ensuite, pour compléter le méta-modèle et nos objectifs de génération de comportements sensibles et intelligibles, nous proposons une seconde contribution : un mécanisme de sélection d'action par graphes d'influences et de préférences. Ce moyen a l'avantage d'être intelligible par sa nature visuelle et de faciliter la prise en compte de manière modulaire de nouveaux comportements sur la sélection.

Enfin, nous proposons une implémentation de notre méta-modèle, au travers d'une bibliothèque pour la simulation d'agents virtuels multi-paradigmes [Picault and Sicard, 2020]. Nous n'imposons pas le paradigme associé aux agents, qui peut aussi changer dynamiquement lors de la simulation. Nous employons une approche de design orientée-données (DOD), avec l'utilisation d'une méthode appelée ECS, plutôt qu'une approche orientée-objet (POO), dans le but d'améliorer les performances.

Nous avons montré lors d'une évaluation de la sensibilité que les modèles cognitifs affectent le comportement de l'agent de manière cohérente par rapport aux propriétés des modèles cognitifs introduits. Nous montrons ensuite, lors d'une évaluation de

l'intelligibilité et de la représentativité, que les utilisateurs ont perçus la différence de comportements et que les explications construites à partir des modèles cognitifs ont été perçues comme pertinentes. Nous avons aussi appliqué nos travaux dans un contexte de formation professionnelle dans le cadre du projet ORCHESTRAA. Enfin, les résultats de notre évaluation de performances de notre bibliothèque vis-à-vis d'autres plate-formes de la communauté des systèmes multi-agents, montrent que notre implémentation est constamment plus rapide, souvent de plusieurs ordres de grandeur dans certains cas.

Abstract

The research carried out during this thesis deals with the **generation of virtual agent behaviors**. That is to say, the construction of computer models to determine the actions that an agent must perform in a virtual environment, in order to reproduce the actions and reactions of a human in the simulated situation. We apply this work in the context of professional training to train operators in crisis situations.

The computer science literature proposes many agent models that integrate human dimensions in the agent's functioning, such as emotions, by combining some cognitive models from the Human and Social Sciences (SHS) literature.

Our goal is to facilitate the design of such computer models. We wish to develop a generic approach that facilitates the integration of cognitive models in any agent model and that generates representative, sensitive and intelligible behaviors.

We propose a meta-model that abstracts an agent model into five concepts : **O**peration, **P**erception, **A**action, **C**features, and **K**nowledge. Cognitive models can modularly affect these five concepts, thanks to their composability. We also make as few assumptions as possible about the agent model and the cognitive models.

Then, to complete the meta-model and our goals of generating sensible and intelligible behaviors, we propose a second contribution : an action selection mechanism by influence and preference graphs. This means has the advantage of being intelligible by its visual nature and of facilitating the modular consideration of new behaviors on the selection.

Finally, we propose an implementation of our meta-model, through a library for the simulation of multi-paradigm virtual agents. We do not impose the paradigm associated with the agents, which can also change dynamically during the simulation. We employ a data-oriented design approach (DOD), using a method called ECS, rather than an object-oriented approach (OOP), in order to improve performance.

We show in a sensitivity evaluation that the cognitive models affect the agent's behavior in a consistent way with the properties of the introduced cognitive models. We then show, in an evaluation of intelligibility and representativeness, that users perceived the difference in behavior and that the explanations built from the cognitive models were perceived as relevant. We also applied our work in a professional training context in the context of the ORCHESTRAA project. Finally, the results of our performance evaluation of our library against other platforms in the multi-agent systems community show that our implementation is consistently faster, often by several orders of magnitude in some cases.

Remerciements

Tout d'abord, je voudrais remercier Domitile LOURDEAUX qui m'a introduit dans le monde de la recherche. Voici la petite histoire : en dernier année de master, nous devions trouver un stage. Aucune offre ne me convenait. J'avais donc décidé de créer mon propre sujet de stage. Il fut envoyé à une dizaine de chercheurs et entreprises en lien avec le domaine. Je n'ai eu aucun retour, sauf de Domitile qui m'a répondu positivement, et ceux dans les minutes suivant l'envoi ! Mon stage s'est ainsi déroulée sous sa tutelle. Puis, elle m'a proposé de continuer en thèse. Merci de m'avoir donné l'opportunité de travailler sur ce sujet passionnant. Merci aussi pour ta confiance, ton humanité et ton aiguillage durant toute la thèse.

Ensuite, je voudrais étendre ces remerciements à mon co-directeur Nicolas SABOURET. Merci pour le temps que tu m'as accordé. Surtout, merci pour les nombreuses réunions où tu m'as aidé à bien formaliser ma thèse ! Cette thèse n'aurait pas été aussi loin et de la même qualité, sans tes retours et contributions.

Merci à vous deux pour votre disponibilité et votre soutien. Vous formez un super duo de directeurs complémentaires, je n'aurais pas pu espérer mieux. Merci !

Merci au Lieutenant-colonel Bruno CEPPARO, à Dominique Palabost et aux membres du CASPOA pour leur disponibilité et leur aide tout au long du projet ! C'était un plaisir de se rendre au CASPOA et de découvrir ce domaine passionant. Nous avons toujours été merveilleusement bien reçu.

Merci à Alex, Louis, Pierre et Tarek pour leur précieuses amitiés !

Mais surtout, je voudrais remercier mon incroyable famille à qui je dois tout.

Sommaire

Chapitre 1

Introduction

Contents

Les recherches effectuées au cours de cette thèse traitent de la génération de comportements d'agents virtuels, c'est-à-dire de la construction de modèles informatiques pour déterminer les actions qu'un agent doit effectuer au sein d'un Environnement Virtuel (EV) pour reproduire les actions et les réactions d'un humain dans la situation simulée.

Le cadre applicatif dans lequel nous développons ces travaux est celui de la formation professionnelle [Barot, 2014, Huguet et al., 2016, Silverman et al., 2012] dans lequel la simulation en environnement virtuel est utilisée pour entraîner des opérateurs lors de situations de crise. Les agents doivent alors adopter des comportements complexes afin de reproduire les comportements observés chez les humains. La complexité provient des nombreuses dimensions à intégrer dans le modèle informatique : la prise de décision par rapport à une tâche, la coordination avec les autres acteurs, mais également des aspects sociaux vis-à-vis d'autres acteurs, comme l'adaptation aux membres de son équipe, des phénomènes psychologiques, comme les émotions, ou encore physiologiques, comme le stress. L'intégration de toutes ces dimensions au sein d'un même modèle est un problème difficile [Bourgais, 2018] et la littérature en informatique propose de nombreux modèles qui intègrent certaines de ces dimensions. De plus, ils doivent s'adapter à l'utilisateur, car nous considérons des environnements relativement ouverts pour l'apprentissage (et la possibilité de se tromper). Il devient alors compliqué de tout scripter.

Ces modèles informatiques s'appuient sur deux aspects : un modèle d'agent d'un côté, un ou plusieurs modèles cognitifs de l'autre. Arrêtons-nous un instant sur ces deux notions qui sont au cœur de nos travaux.

FIGURE 1.1 – Capture d'écran de l'EV VICTEAMS où les infirmiers sont des agents sous les ordres de l'apprenant [Huguet et al., 2016].

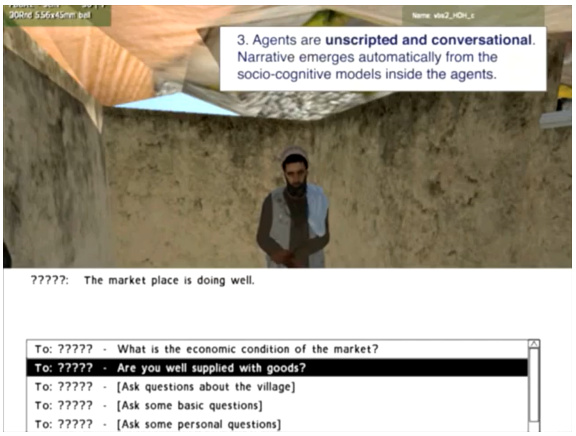


FIGURE 1.2 – Capture d'écran de l'EV NON-KIN VILLAGE pour l'entrainement socio-culturel avec la simulation d'une petite communauté autonome [Silverman et al., 2012].

1.1 Modèle d'agent et modèle cognitif

Modèle d'agent Un *modèle d'agent* est une manière de définir la boucle perception-délibération-action des agents. Elle détermine comment l'agent sélectionne les actions qu'il effectue selon la situation (*cf.* figure 1.3) [Balaji and Srinivasan, 2010]).

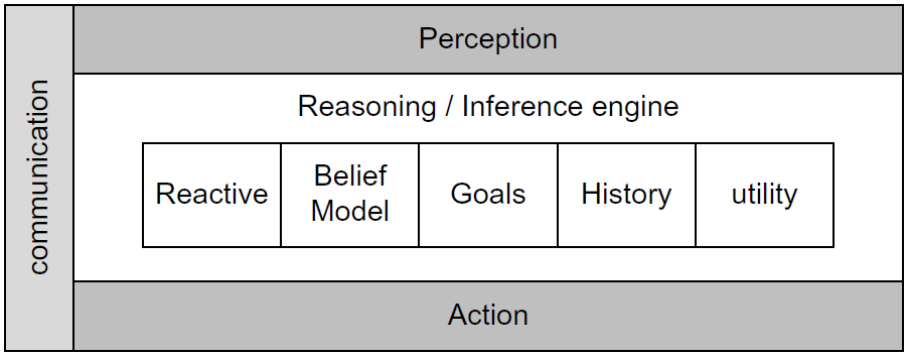


FIGURE 1.3 – Schématisation d'un cycle « Perception-Raisonnement-Sélection » d'un agent selon [Balaji and Srinivasan, 2010].

On distingue ainsi les modèles d'agents réactifs [Russell and Norvig, 2021], dans lesquels les actions sont directement reliées aux perceptions, des modèles d'agents cognitifs, intégrants des dimensions humaines dans leur raisonnement. Par exemple, dans les modèles d'agents fondés sur BDI [Bratman, 1987] la sélection d'action se fait à l'aide d'inférences sur une base de croyances pour satisfaire des plans.

Un exemple de modèles réactifs issu de [Russell and Norvig, 2021] est une voiture qui freine lorsque la voiture de devant freine. Cela suppose que l'information « la voi-

ture freine » soit inférable instantanément, sans avoir besoin de mémoire, ce qui rend impossible de tester si la vitesse de la voiture décroît par exemple.

Un exemple de modèle d'agent fondé sur BDI [Bratman, 1987] est l'architecture BEN [Bourgais et al., 2021] qui a permis de simuler le comportement des individus lors de l'évacuation d'une boîte de nuit où un feu s'est déclenché. Les agents possèdent :

1. des croyances, notamment sur la position de la porte de sortie ou sur la présence d'un feu,
2. des règles d'inférence, comme une règle qui indique que si l'agent croit qu'un feu est présent, alors il a le désir de fuir,
3. des plans pour répondre aux désirs. Par exemple, pour fuir un feu, l'agent peut courir vers la porte s'il sait où elle se trouve, suivre un autre agent envers qui il a confiance, suivre les panneaux de signalisation, *etc.* .

Ces deux modèles, réactif et cognitif, correspondent à deux manières de modéliser et de concevoir le comportement des agents. Nous verrons lors de l'état de l'art qu'il en existe de nombreux autres, avec des propriétés différentes, adaptés aux objectifs et contraintes techniques de chaque simulation. Afin d'intégrer des dimensions humaines dans cette production de comportements, certains intègrent des modèles cognitifs.

Modèle cognitif Un *modèle cognitif* (à ne pas confondre avec les modèles d'agents cognitifs) est un modèle issu de la littérature en Sciences Humaines et Sociales (SHS) qui propose une représentation simplifiée d'un ensemble de processus psychologiques et/ou intellectuels. L'intérêt des modèles cognitifs réside dans la construction de comportements plus proches de ceux des humains grâce à l'intégration de dimensions sociales, psychologiques, *etc.* , dans la production du comportement. De plus, ces modèles procurent un support d'explication au choix d'action de l'agent.

Par exemple, les travaux de Demary [Demary, 2020] proposent un modèle cognitif des différents comportements vis-à-vis d'un leader, en fonction du style « suiveur » et « communicant » de l'agent (*cf.* tableau 1.1).

	Communicant	**Non-communicant**
Proactif	Je raisonne, j'informe le leader, je conteste si besoin, j'obéis au final. (Exemplaire / Courageux)	Je prend des décisions, je conteste. (Loup solitaire)
Passif	Je demande quoi faire et je fais ce qu'on me dit. (Mouton / Faiseur)	Si on ne me dit rien, je suis la procédure. Sinon, je fais ce qu'on me demande (Yes people / Ressource)

TABLE 1.1 – Comportement selon le style « suiveur » et le style « communicant ».

Ensuite, voici un autre exemple issu des travaux de Driskell [Driskell et al., 2018]. Nous le verrons plus en détail lors de l'état de l'art. Succinctement, à la suite de l'étude des comportements des équipes en situations extrêmes, il propose un modèle dans lequel le stress affecte le comportement des individus d'une équipe de plusieurs manières. Un de ces effets est la réduction de la communication et l'augmentation de frictions

interpersonnelles. Nous observons que ces deux modèles : Demary [Demary, 2020] et Driskell [Driskell et al., 2018] ont, du moins ici, des comportements qui se chevauchent (comportement vis-à-vis de ses équipiers). Ainsi, au sein d'un même modèle d'agent et pour un même problème, nous pouvons implémenter différents modèles cognitifs, qui conduiront de fait à des comportements d'agents différents.

Dans cette thèse, nous nous sommes intéressés à la conception d'un modèle informatique capable d'intégrer de manière modulaire et de combiner plusieurs modèles cognitifs dans un modèle d'agent, dans le but de *générer des comportements représentatifs, sensibles et intelligibles*. Nous allons immédiatement définir ces différentes notions.

1.2 Génération de comportements

Selon le « Grand dictionnaire de la psychologie » [Bloch et al., 1992], plusieurs définitions pour le terme « comportement » sont envisageables :

Citation 1 - Comportement selon Gallo A.

« Le comportement est un ensemble de phénomènes observables de façon externe. »

Bloch, H., Chemama, R., Gallo, A., Leconte, P., Le Ny, J.-F., Postel, J., Moscovici, S., Reuchlin, M., and Vurpillot, E. (1992). Grand dictionnaire de la psychologie. Publisher: Larousse

Citation 2 - Comportement selon Piéron H.

« Manière d'être et d'agir des Animaux et des Hommes, manifestement objectives de leur activité globale. »

Bloch, H., Chemama, R., Gallo, A., Leconte, P., Le Ny, J.-F., Postel, J., Moscovici, S., Reuchlin, M., and Vurpillot, E. (1992). Grand dictionnaire de la psychologie. Publisher: Larousse

Citation 3 - Comportement selon Watson J.B

« Le comportement est l'ensemble des réactions objectivement observables qu'un organisme généralement pourvu d'un système nerveux exécute en réponse aux stimulations du milieu, elles-mêmes objectivement observables. »

Bloch, H., Chemama, R., Gallo, A., Leconte, P., Le Ny, J.-F., Postel, J., Moscovici, S., Reuchlin, M., and Vurpillot, E. (1992). Grand dictionnaire de la psychologie. Publisher: Larousse

Citation 4 - Comportement selon Gallo A.

« Le comportement est une réalité appréhendable sous la forme d'unités d'observation, les actes, dont la fréquence et les enchaînements sont susceptibles de se modifier; il traduit en action l'image de la situation telle qu'elle est élaborée, avec ses outils propres, par l'être que l'on étudie : le comportement exprime une forme de représentation et de construction d'un monde particulier (Umwelt) »

Bloch, H., Chemama, R., Gallo, A., Leconte, P., Le Ny, J.-F., Postel, J., Moscovici, S., Reuchlin, M., and Vurpillot, E. (1992). Grand dictionnaire de la psychologie. Publisher: Larousse

Le comportement humain peut être donc être vu comme l'ensemble des actions et réactions (mouvements, modifications physiologiques, expression verbale, *etc.*) d'un individu observable dans une situation donnée.

Transposée à la simulation en environnement virtuel, cette définition nous conduit à la proposition suivante :

Comportement d'un agent

Le comportement d'un agent est l'ensemble des actions (déplacer, interactions avec un objet de l'environnement, expressions verbales/faciales, *etc.*) qu'il effectue au sein de l'environnement virtuel, directement observable par un autre agent ou un utilisateur de la simulation, sans interprétation subjective en termes de buts ou cause d'action, lors d'une situation donnée, c'est-à-dire un état partiel (observable) de la simulation.

Cette définition revêt deux aspects :

1. Un comportement correspond à un ensemble d'actions : les modèles d'agents visent donc à permettre aux entités artificielles de produire des comportements, à travers la sélection d'action.
2. Un comportement traduit la manière de réagir à une situation donnée : les modèles cognitifs visent donc à permettre aux agents d'avoir des « manières de réagir » qui ressemblent à celles des humains.

La génération de comportement consiste donc à 1) sélectionner des actions dans le temps afin de 2) rendre compte de manières de réagir.

Il faut distinguer le comportement de l'agent de son *identification* par l'utilisateur, qui va le qualifier avec des termes issus des modèles cognitifs (par exemple : l'agent semble stressé). Cette subjectivité de l'observateur, appliquée sur le comportement objectif de l'agent, correspond à l'image mentale que l'observateur s'est faite de l'agent et de la situation, qui peut varier d'un utilisateur à l'autre pour une même situation et un même comportement d'agent. Du point de vue système, nous savons quel comportement générer, mais nous ne pouvons pas contrôler la subjectivité d'un observateur externe.

Exemple

Nous allons illustrer la génération de comportement au travers d'un exemple issu du projet VICTEAMS [Huguet et al., 2016] :

Un patient nécessite une injection de morphine. Un médecin diagnostique ce patient et conclut à tort qu'il a besoin d'une injection d'adrénaline. Il ordonne à l'agent infirmier de réaliser l'injection. La figure 1.4 illustre l'exemple.

FIGURE 1.4 – Illustration d'un exemple issu du projet VICTEAMS où un agent infirmier doit réaliser une injection suite à un ordre du médecin.

Supposons que notre modèle d'agent permette d'abord de confirmer le suivi de l'ordre du médecin, de contester son ordre ou de se taire. Ensuite, il peut réaliser l'une des deux injections, puis enfin rapporter ou non ce qu'il a réalisé. La figure 1.5 résume les différentes actions pour chaque étape. Ainsi, la séquence d'action [« confirmer », « injecter adrénaline », « rapporter »] est un des comportements productibles par le modèle d'agent.

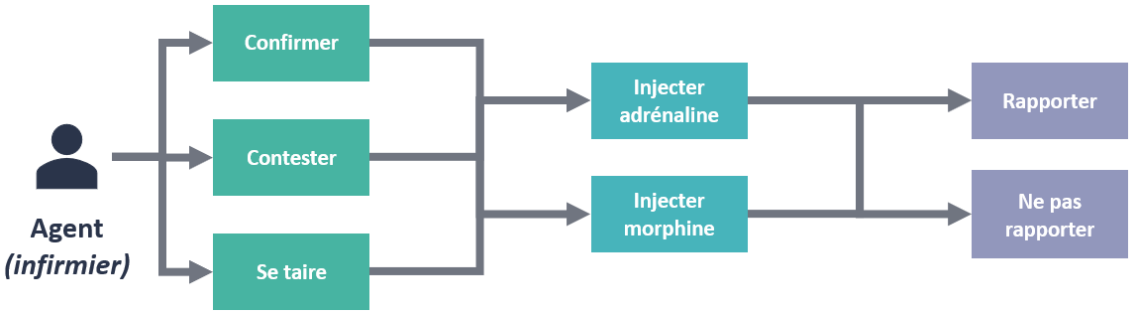


FIGURE 1.5 – Le modèle d'agent permettrait de produire les actions observables suivantes.

La prise de décision et les actions de l'infirmier varient selon son profil et la situation, en accord avec les modèles cognitifs. Supposons aussi qu'au sein de ce modèle, deux modèles cognitifs soient intégrés :

1. Le premier est le modèle de Demary [Demary, 2020], introduit précédemment avec le tableau 1.1. Ce modèle introduit quatre comportements, correspondant à chaque case.
2. Le deuxième est un modèle cognitif de stress ad-hoc, soit inventé pour l'exemple, qui introduit la possibilité de se tromper d'injection lorsque notre agent est stressé. Ce modèle introduit deux comportements : « stressé » ou « non-stressé »

Suite à l'ajout de ces modèles, le profil de l'agent peut correspondre à :

— proactif ou passif,
— communicant ou non-communicant,
— stressé ou non-stressé.

Du point de vue du système, le profil de l'agent est connu. Le système sait donc quel comportement produire à priori pour que l'agent adopte les comportements induits par les modèles cognitifs. Le système produit ainsi le comportement suivant : [« se taire », « injection adrénaline », « ne rapporte pas »].

Avant de révéler pourquoi ce comportement a été produit, plaçons-nous du point de vue d'un observateur externe, afin d'illustrer la difficulté d'identifier un comportement. Précisons qu'un observateur externe, observe uniquement le comportement de l'agent et non son état interne et la situation. Dans l'exemple, il observe ainsi l'ordre du médecin. Nous supposons aussi pour l'exemple que l'observateur externe est au courant de l'injection correcte (morphine). Essayons d'identifier quels comportements notre agent a adoptés en inférant le profil de l'agent. À première vue, nous pouvons inférer que notre agent est « non-communicant » n'ayant pas confirmé ni contesté l'ordre et n'ayant pas rapporté son injection. Toutefois, le fait que l'infirmier ait réalisé l'injection inverse, mais correcte, pourrait rendre compte de deux profils :

1. « stressé » et « passif », ce qui correspondrait aux comportements « stressé » et « Yes-people » (*cf.* table 1.1).
2. « non-stressé » et « pro-actif », ce qui correspondrait aux comportements « non-stressé » et « Loup solitaire » (*cf.* table 1.1).

Cet exemple illustre que la production de comportements et l'identification de comportements sont deux problèmes différents, mais très liés. Une première difficulté est d'arriver à produire un comportement induit par les modèles cognitifs. Une autre difficulté est d'arriver après coup à ce qu'un observateur externe puisse correctement identifier les comportements.

1.3 Objectifs

Ainsi, pour la génération de comportements, nous aimerions disposer d'un modèle informatique qui vérifie plusieurs propriétés :

1. Premièrement, la *représentativité* des comportements aux comportements observés. Chaque métier définit une/des activités qu'il est nécessaire de reproduire, comme l'exemple de la section précédente.
2. Deuxièmement, la *sensibilité* des comportements aux modèles cognitifs, c'est-à-dire le fait que l'intégration des modèles affectent le comportement d'un agent de manière cohérente vis-à-vis des propriétés de ces modèles. Par exemple, en situation de crise, les médecins secouristes peuvent être amenés à utiliser un chiffon pour gagner de précieuses secondes au lieu de mettre des gants pour faire un point

de compression. Nous voulons que l'ajout d'un modèle cognitif, affectant le respect de la réglementation, permette effectivement l'apparition d'un tel comportement.

3. Troisièmement, l'*intelligibilité* des comportements, c'est-à-dire qu'un utilisateur, externe de la simulation, doit pouvoir qualifier les séquences d'actions qui traduisent l'expression d'un comportement (par exemple : « l'agent semble stressé » ou « l'agent respecte la réglementation »).

Nous pouvons dire que les modèles d'agents permettent de générer des comportements représentatifs et les modèles cognitifs de générer des comportements sensibles et intelligibles.

1.4 Problème(s) et questions de recherche

Concevoir une architecture d'agents générique est difficile : les comportements et les modèles cognitifs des agents dépendent fortement du domaine d'application et des objectifs de la simulation (qu'il s'agisse d'un scénario de formation, d'un jeu, etc). Dès lors, nous ne connaissons pas à première vue les modèles cognitifs à intégrer et à combiner, ainsi que les comportements à générer. Qui plus est, les différentes combinaisons de modèles affectent le comportement à différents niveaux. Il ne s'agit pas juste de généraliser l'intégration dans une étape du modèle d'agent, mais dans sa globalité (*cf.* la figure 1.6 schématise cette difficulté). Enfin, la sensibilité et l'intelligibilité, de modèles cognitifs dans un modèle d'agent donné, est un problème compliqué, même avec une combinaison de modèles cognitifs connus. Il faut s'assurer que chaque modèle affecte correctement la production tout en restant intelligible. Plus le nombre de modèles cognitifs est important, plus cela est compliqué par le nombre d'interactions et d'impacts à considérer sur les diverses étapes du modèle d'agent. Sans ces connaissances à priori (modèle d'agent + modèles cognitifs), ce problème est encore plus complexifié, notamment lors de la sélection d'action. En effet, nous ne pouvons pas proposer une formule spécifique à un ensemble de modèles cognitifs. Il faudrait la revoir à chaque ajout/retrait de modèles cognitifs.

Dans cette thèse, nous avons essayé de proposer un modèle qui permet d'intégrer, sans connaissance à priori, des modèles cognitifs au sein d'un modèle d'agent.

Plus précisément, nous nous posons les questions de recherches suivantes :

1. Comment généraliser un modèle d'agent et les modèles cognitifs pour les combiner ?
2. Comment à partir de cette généralisation, pouvons-nous générer des comportements représentatifs ?
3. Serait-il possible, à partir de cette base, de générer des comportements :
   (a) sensibles à l'ajout de modèles cognitifs et
   (b) qui restent intelligibles malgré tout ?

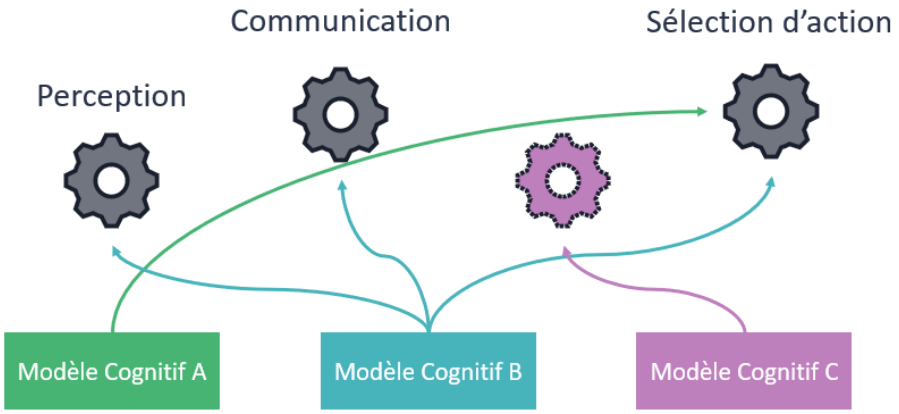


FIGURE 1.6 – Une des difficultés est de généraliser l'intégration de modèles cognitifs, en raison de la nature variée de leurs impacts.

Pour répondre à ces questions, nous souhaiterons disposer d'un modèle, dont l'originalité serait de répondre à cette double difficulté : **permettre l'intégration de modèles cognitifs dans un modèle d'agent, avec un minimum d'hypothèses sur le modèle d'agent et sur les modèles cognitifs, tout en s'assurant que les comportements produits restent sensibles aux modèles et intelligibles (*cf.* 1.7).**

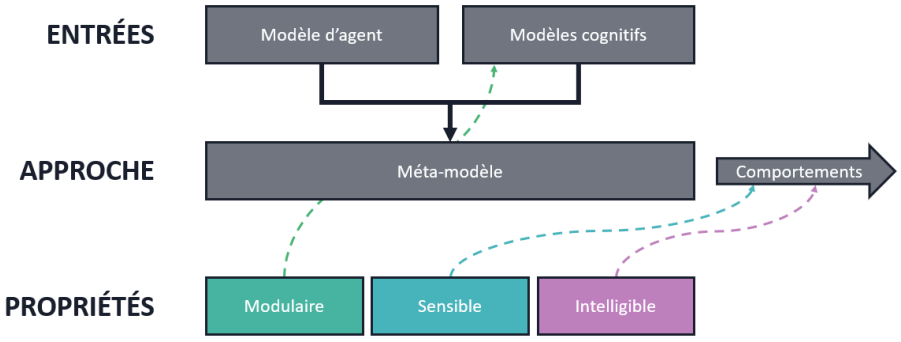


FIGURE 1.7 – Illustration de notre approche.

1.5 Contributions

Nos travaux portent deux contributions :

— OPACK, un **méta-modèle** computationnel pour **généraliser la réification de comportements** au sein d'opérations grâce à des stratégies. L'objectif de ce modèle est d'**intégrer, sans connaissance à priori, des modèles cognitifs au sein d'un modèle d'agent**, tout en donnant les moyens d'obtenir **les propriétés de représentativité, de sensibilité et d'intelligibilité**.

— Nous nous appuyons sur ce modèle pour proposer un **mécanisme de sélection d'action** qui permet d'**incorporer modulairement et intelligiblement l'impact de modèles cognitifs** lors de cette étape.

1.6 Organisation du mémoire

Nous commençons par un état de l'art organisé en trois parties. La première concerne la littérature SHS. Nous y étudions ce qui caractérise les modèles cognitifs dans les buts de les généraliser. Cette généralisation aspire à faciliter leur intégration dans un modèle d'agent. Dès lors, nous étudions, dans la seconde partie, la littérature Système Multi-Agent (SMA) afin d'observer les manières d'intégrer de tels modèles, au sein d'un modèle d'agent. Enfin, nous étudierons des méthodes de sélection d'action, dans l'objectif d'intégrer de manière modulaire et intelligible l'impact de modèles cognitifs lors de cette étape.

À la suite de cet état de l'art, nous présenterons les contributions apportées, en commençant par le méta-modèle OPACK, puis le mécanisme de sélection d'action par graphes d'influences et de préférences.

Le chapitre suivant sera dédié à l'implémentation du méta-modèle sous forme d'une bibliothèque pour la simulation d'agents virtuels. Nous détaillerons ses fonctionnalités, au travers de quelques exemples.

Ensuite, nous détaillerons l'application de nos travaux au projet Orchestration of stessful situations for training based on virtual reality and adaptative agents in air operations context (ORCHESTRAA). Nous commençons par le présenter, avant de suivre avec les différentes expérimentations menées.

Nous présentons les résultats de nos diverses évaluations préliminaires sur nos propriétés recherchées, ainsi que les retours de l'application de nos travaux au projet ORCHESTRAA. Nous présenterons aussi plusieurs évaluations de performances de notre outil comparé à des plate-formes de la communauté SMA, comme Gama ou Netlogo.

Enfin, nous concluons sur nos travaux et les perspectives envisageables.

Chapitre 2

État de l'art

Contents

Notre objectif est de comprendre comment généraliser un modèle d'agent et des modèles cognitifs afin de les combiner, tout en générant des comportements représentatifs, sensibles et intelligibles. Nous présentons d'abord ce qui caractérise ces modèles cognitifs. Ensuite, nous présentons quelques modèles d'agent classiques en simulation multi-agents, afin d'observer leur manière d'intégrer des modèles cognitifs. Puis, nous présentons des modèles de sélection d'action, afin d'étudier les manières d'intégrer de manière intelligible et modulaire l'impact des modèles cognitifs lors de cette étape. Enfin, nous positionnerons l'ensemble de ces travaux par rapport à notre problématique. Nous montrons notamment en quoi l'opérationnalisation de modèles cognitifs et leur généralisation sont difficiles.

2.1 Modèles cognitifs

Selon le dictionnaire de psychologie de l'Association Américaine de Psychologie (APA) [1] , un modèle cognitif et un processus cognitif sont définis comme suit :

Citation 5 - Modèle cognitif

« Une représentation simplifiée et théorique d'un ensemble de processus cognitifs/-mentaux amenant à des phénomènes observables et qui fournit des explications sur la cause. »

APA Dictionary of Psychology

Citation 6 - Processus cognitif

« N'importe quelle fonction impliquée dans des activités telles que l'attention, la perception, l'apprentissage et la résolution de problèmes. »

APA Dictionary of Psychology

Nous retrouvons aussi ce principe dans les travaux de Seel [see, 2012].

La littérature en SHS propose un grand nombre de modèles cognitifs selon les processus à modéliser. Dans le domaine des processus socio-affectifs, par exemple, on peut citer le modèle transactionnel de stress proposé par Folkman et Lazarus [Lazarus and Folkman, 1984], le modèle de personnalité OCEAN [Costa Jr. and McCrae, 2008] ou en encore le modèle OCC [Ortony et al., 1988] très souvent utilisé en informatique pour opérationnaliser l'évaluation cognitive.

Les modèles que nous présentons ci-dessous ont été choisis pour leur complémentarité : ils couvrent les différentes difficultés qui apparaissent lors de la conception d'un modèle générique tel que nous essayons de le concevoir. Ensuite, la plupart ont été intégrés dans des modèles d'agent présentés dans la partie 2.1 de l'état de l'art. En revanche, nous présentons un modèle cognitif moins connu : celui de Driskell [Driskell et al., 2018]. Il nous a semblé intéressant pour notre domaine d'application, à savoir le travail en équipe et les situations critiques / de crises.

2.1.1 Modèle transactionnel du stress par Folkman & Lazarus

En 1984 [Lazarus and Folkman, 1984], Folkman propose un modèle transactionnel du stress, fondé sur la définition suivante du stress par Lazarus [Lambert and Lazarus, 1970] :

1. https://dictionary.apa.org

Citation 7 - Stress

« Relation entre la personne et l'environnement, considérée comme importante sur le plan personnel et qui met à l'épreuve ou dépasse les ressources disponibles pour faire face à la situation. »

Lambert, W. W. and Lazarus, R. S. (1970). Psychological Stress and the Coping Process. *The American Journal of Psychology*, 83(4):634

Nous appuyons la présentation du modèle sur le chapitre 3 de [Algava et al., 2011], en réutilisant notamment la figure 2.1. Le modèle se découpe en deux processus. Un premier processus, dit d'*appraisal*, où l'individu évalue la situation. Un deuxième processus, dit de *coping*, où l'individu évalue des stratégies d'adaptation. Lors du processus d'*appraisal*, deux évaluations sont menées :

1. Une première évaluation, dite *primaire*, où l'individu évalue l'enjeu de la situation : perte, menace ou défi. Il en ressort le stress perçu.
2. Une deuxième évaluation, dite *secondaire*, où l'individu évalue les ressources disponibles pour faire face à la situation. Il en ressort le contrôle perçu.

À la suite de ce processus, le processus de *coping* se déclenche. L'individu évalue, selon le stress et le contrôle perçu, les stratégies d'adaptations envisageables. Trois types de stratégies ont été identifiées :

1. une stratégie active centrée sur la résolution de problèmes, soit s'attaquer à la source du problème, *e.g.* vérifier que le gaz a bien été fermé,
2. une stratégie passive centrée sur les émotions, *e.g.* se calmer,
3. et une stratégie d'évitement, *e.g.* se distraire.

L'adaptation s'exprime autant par des actions, *e.g.* lors de la stratégie 1. et 3. que par des efforts cognitifs, *e.g.* lors de la stratégie 2.

Le fonctionnement du modèle est résumé par la figure 2.1, issue de [Algava et al., 2011].

L'intégration d'un modèle cognitif comme celui-ci pose la question de l'intégration modulaire des deux processus et notamment la prise en compte des stratégies sur la prise de décision.

2.1.2 Modèle de personnalité : OCEAN

Le modèle OCEAN, aussi appelé *Big Five*, est un modèle descriptif de la personnalité issu de travaux du 20e siècle [Plaisant et al., 2010]. Il s'agit d'une taxonomie des traits de personnalité, découpée en cinq sections (*cf.* table 2.1.)

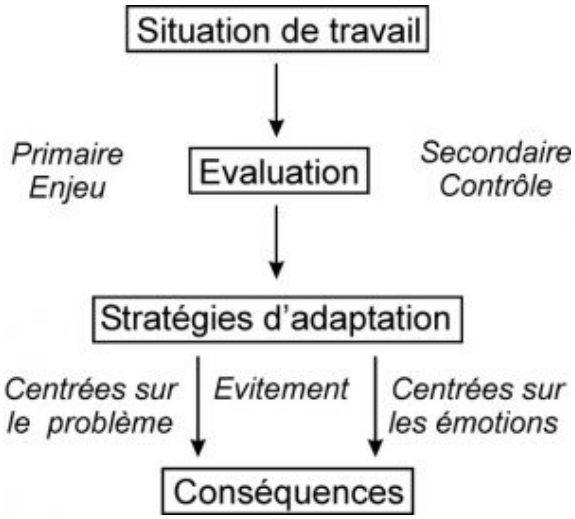


FIGURE 2.1 – Schématisation du modèle transactionnel du stress par Folkman & Lazarus [Lazarus and Folkman, 1984]. La processus d'*appraisal* évalue le stress, ainsi que le contrôle perçu, tandis que le processus de *coping* évalue la stratégie d'adaptation à adopter.

O	C	E	A	N
Ouverture	Conscience	Extraversion	Agréabilité	Névrosisme

TABLE 2.1 – Cinq dimensions de la personnalité.

Selon le dictionnaire de psychologie de l'APA [2] , un trait de personnalité est défini comme suit :

Citation 8 - Trait de personnalité

« Une caractéristique interne relativement stable, cohérente et durable qui est déduite d'un modèle de comportements, d'attitudes, de sentiments et d'habitudes chez l'individu. L'étude des traits de personnalité peut être utile pour résumer, prédire et expliquer le comportement d'un individu [...]". »

APA Dictionary of Psychology

Afin de déduire ces traits chez un individu, Costa et McCrae proposent un questionnaire pour évaluer initialement les dimensions N, E et O, puis les deux autres, intitulé NEO/PI-R *(NEO Personality Inventory Revised)* pour la dernière version en date [Costa Jr. and McCrae, 2008]. Chaque dimension est découpée par 6 facettes, recensées dans la figure 2.2. Le questionnaire permet aussi d'évaluer ces différentes facettes.

Ainsi, le modèle OCEAN a pour objectif de couvrir le spectre de la personnalité aux travers de ces cinq dimensions. Une limitation soulevée par Caroline Faur [Faur,

2. https://dictionary.apa.org/personality-trait

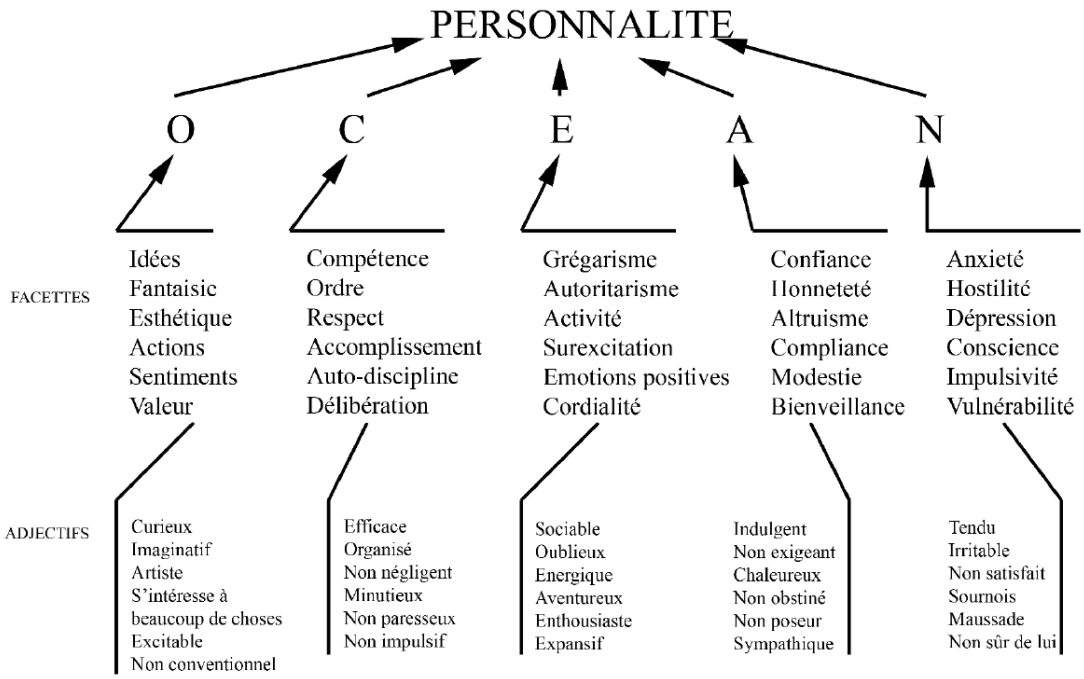


FIGURE 2.2 – Illustration issue de [Plaisant et al., 2010] des 5 dimensions et de leurs 6 facettes de NEO/PI-R [Costa Jr. and McCrae, 2008].

2016] est que ce genre d'approche dimensionnelle ne prend pas en compte l'importance de la situation dans le comportement. Elle emploie l'exemple suivant : « *une personne introvertie peut se comporter de manière extravertie en présence d'amis, mais de manière introvertie dans un milieu professionnel.* »

Comme nous le verrons dans la partie 2.2, ce modèle est très repris dans l'informatique affective [Bourgais et al., 2021, Sansonnet and Bouchet, 2014, Sánchez et al., 2019]. Vis-à-vis de nos travaux, nous nous posons la question d'intégrer les différentes dimensions modulairement.

2.1.3 Modèle de personnalité : Mayer

Les travaux de Mayer proposent une autre manière de voir la personnalité [Mayer, 2005, Mayer, 2015] définit comme suit :

Citation 9 - Personalité

« Personality is the organized, developing system within the individual that represents the collective action of that individual's major psychological subsystems. »

Mayer, J. D. (2005). A Tale of Two Visions: Can a New View of Personality Help Integrate Psychology? *American Psychologist*, 60(4):294–307

La personnalité est un système dynamique, composé de sous-systèmes. Elle est le résultat du fonctionnement collectif de ses sous-systèmes. Mayer les organise par la figure 2.3 [Mayer, 2015].

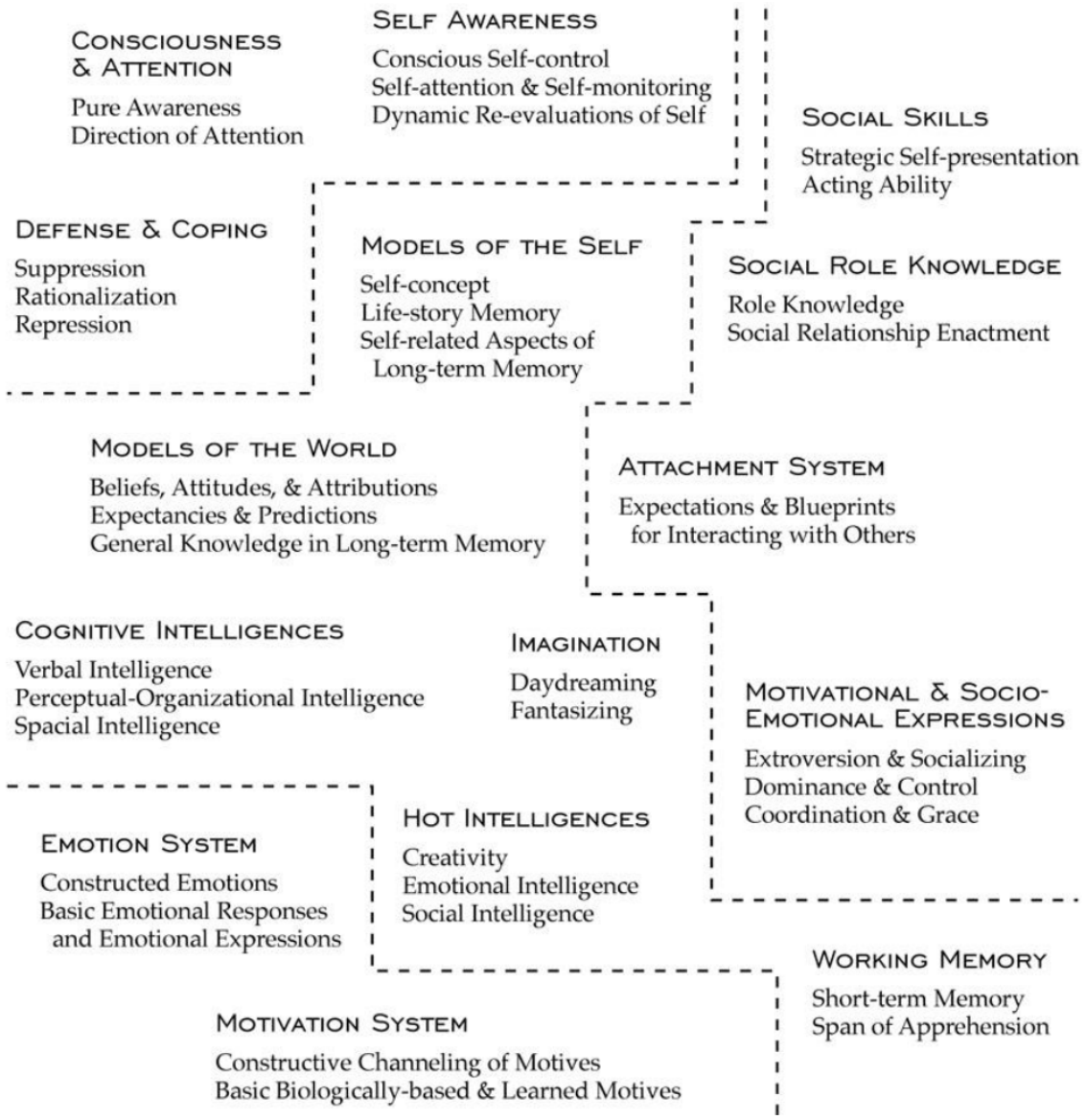


Figure 2.3 – Schématisation des sous-systèmes majeurs psychologiques contenus dans la personnalité selon Mayer [Mayer, 2005]. Voici la description : *"Each psychological subsystem in the diagram is depicted with a few examples of its possible parts. Each subsystem performs a unique set of psychological functions. Systems related to the external aspects of personality are to the right; those related to internal processing are to the left. More complex, learned systems are toward the top; smaller more specific systems are lower in the diagram. As much as was possible in two dimensions, related systems were placed close to each other. These systems blend into one another and often operate in parallel with one another"* [Mayer, 2005].

Afin de compléter sa vision de la personnalité, Mayer désigne aussi la personnalité comme un système qui cohabite avec d'autres systèmes, comme des systèmes biologiques et sociaux [Mayer, 2005]. Il illustre dans la figure 2.4, la position de la personnalité vis-à-vis de ces autres systèmes.

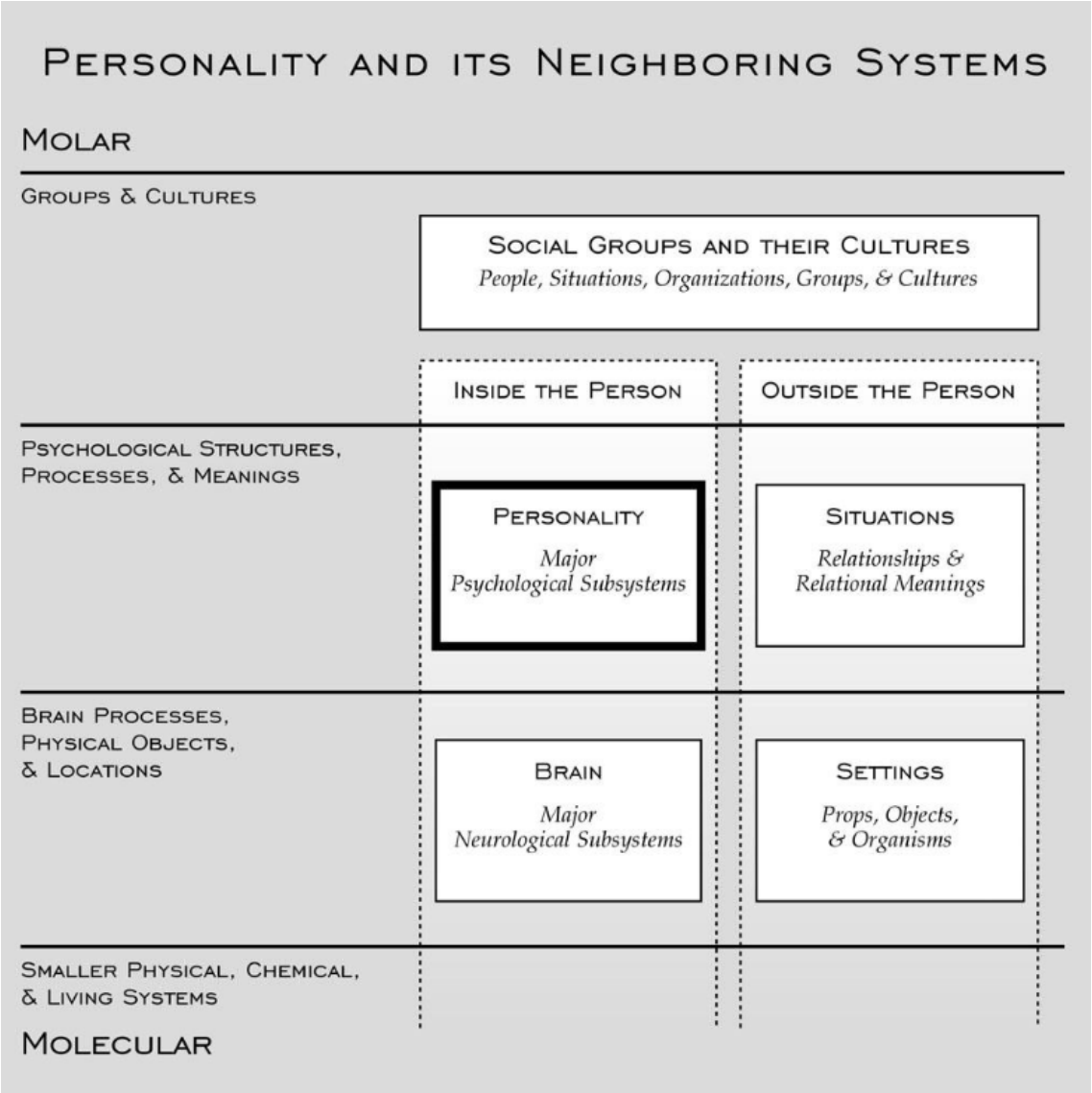


FIGURE 2.4 – Localisation de la personnalité, vis-à-vis des autres systèmes.

Ce modèle cognitif soulève deux difficultés majeures : celle d'arriver à généraliser l'intégration de processus très variés, qui plus est de manière modulaire, ainsi que de gérer les interactions entre les processus, notamment sur la prise de décision.

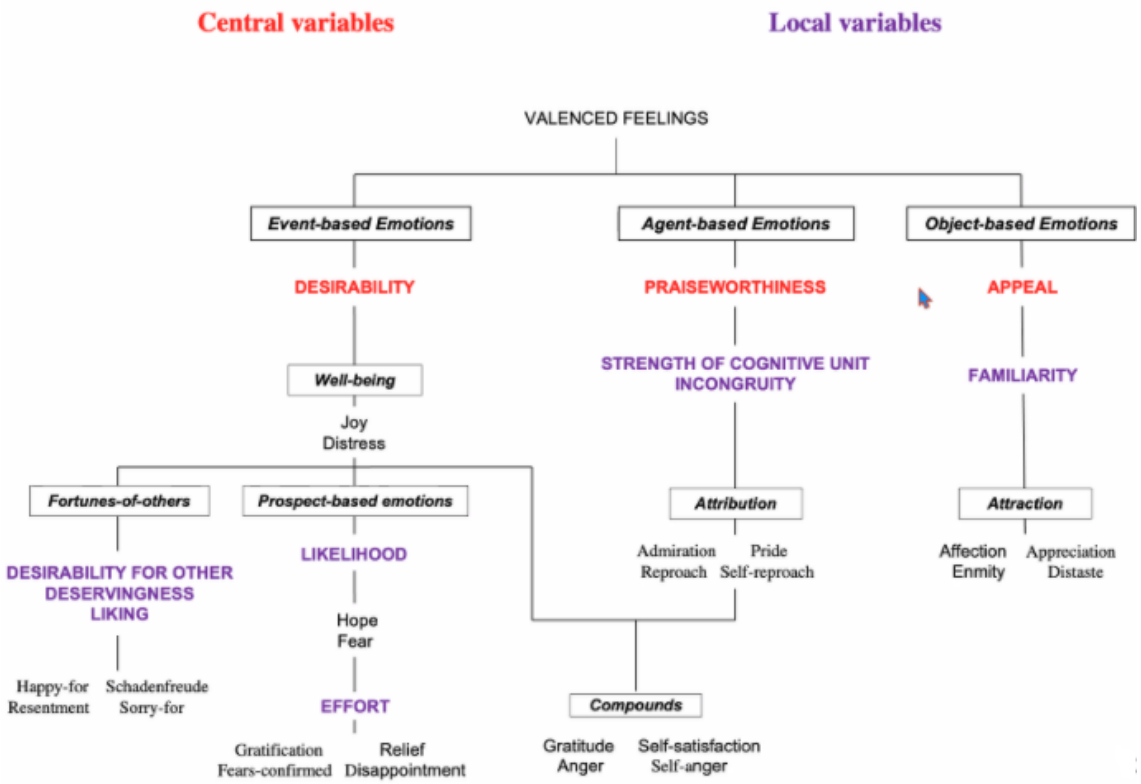


FIGURE 2.5 – Structure des émotions du modèle OCC [Ortony et al., 2022]. *Schadenfreude* est une expression allemande pour signifier une « joie maligne/malsaine » provoquée par le malheur d'autrui. « Enmity » est un sentiment d'hostilité/de haine.

2.1.4 Modèle d'évaluation cognitive des émotions : OCC

En 1988, Ortony, Clore et Collin proposent le modèle d'émotions OCC (acronyme des auteurs du modèle) [Ortony et al., 1988]. Une nouvelle édition est sortie en 2022 [Ortony et al., 2022]. L'état émotionnel d'un individu est représenté aux travers de 24 émotions, soit 12 couples d'émotions antagonistes, e.g *joie / détresse*. La figure 2.5 est un schéma de la dernière version en date (de 2022 donc).

Trois grandes classes d'émotions sont définies : émotions fondées sur les évènements, sur les agents (notamment leurs actions) et sur les objets. Ces classes sont présentées comme étant neutre culturellement. Plus précisément, voici comment les auteurs [Ortony et al., 2022] caractérisent ces trois classes :

— « Les évènements sont évalués en terme de désirabilité vis-à-vis de buts. »
— « Les actions des agents sont évaluées selon leur louabilité vis-à-vis de standards. »
— « Les objets sont évalués en terme d'attrait vis-à-vis de goûts. »

L'intensité des émotions est évaluée autour de trois variables :

— Global : caractère inattendu, proximité psychologique, *etc.*

— Central : désirabilité de l'évènement, louabilité de l'agent, attrait de l'objet.
— Local : mérite, attente, familiarité *etc.* .

Un attrait de ce modèle est sa formalisation adaptée à une approche computationnelle. En revanche, une limite ce modèle est de ne pas relier l'état émotionnel aux comportements. D'une certaine manière, nous pouvons dire que le modèle propose un processus d'*appraisal* des émotions, mais pas de *coping*.

2.1.5 Modèle de comportements de followership par Demary

Les travaux de Demary [Demary, 2020] analysent les interactions sociales des membres d'une équipe, notamment entre le chef et ses subordonnées : comment les subordonnées agissent selon le leader et comment le leader perçoit ses subordonnées. Il prend aussi en compte le rôle de la communication lors de ces interactions. Parmi les comportements proposés, Demary propose des comportements liés aux profils « suiveur » et « communicant » des subordonnées [Demary, 2020]. Ils sont résumés dans le tableau 2.2. En fonction du profil du subordonné, il réagit différemment lorsqu'il reçoit un ordre de son supérieur.

	Communicant	**Non-communicant**
Proactif	Je raisonne, j'informe le leader, je conteste si besoin, j'obéis au final. (Exemplaire / Courageux)	Je prend des décisions, je conteste. (Loup solitaire)
Passif	Je demande quoi faire et je fais ce qu'on me dit. (Mouton / Faiseur)	Si on ne me dit rien, je suis la procédure. Sinon, je fais ce qu'on me demande (Yes people / Ressource)

TABLE 2.2 – Comportement selon le style « suiveur » et le style « communicant ». Chaque case du tableau correspond à un comportement à exprimer, lorsque le profil de l'agent correspond aux différents styles.

Comparé aux autres modèles, les comportements du tableau 2.2 sont plus spécifiques / situationnels (comportements vis-à-vis de son dirigeant). Nous nous posons ainsi la question du rapport de la combinaison d'un modèle plus spécifique à des modèles plus généraux. Aussi, les comportements décrits comprennent plusieurs étapes. Par exemple, le comportement de la case « Proactif / Communicant », décrit d'abord un raisonnement, ensuite une communication avec le leader, puis une prise de décision.

2.1.6 Modèle d'activités limites tolérées par l'usage

Les travaux en psychologie cognitive de Fadier et al. [Fadier et al., 2003] introduisent le concept d'activités limites tolérées par l'usage (ALU). Il s'agit d'un compromis informel / d'une déviation dans la prescription d'une activité, mais tolérée dans les faits sous certaines conditions. Par exemple, en situation de crise, une infirmière décide de

ne pas remplacer ses gants déchirés, afin de continuer le traitement urgent d'un patient, pour gérer l'afflux massif de patients.

Ainsi, la représentativité du comportement métier n'est pas juste le suivi des normes du métier. Il s'agit aussi de rendre compte de la variabilité, des déviations réalisées par les opérateurs sur le terrain, pour s'adapter aux contraintes. Nous verrons plus tard une manière de représenter l'activité, nommée ACTIVITY-DL [Barot et al., 2013, Huguet, 2018], qui intègre cet aspect au travers de tags.

2.1.7 Étude de comportements d'équipe en situations extrêmes

Driskell et Salas [Driskell et al., 2015, Driskell et al., 2018] ont réalisé des études conséquentes sur les comportements d'équipes en environnements extrêmes. Ils définissent un environnement extrême, comme « un environnement dans lequel il existe des contraintes importantes en matière de tâches, de société ou d'environnement, qui entraînent des niveaux élevés de risque et des conséquences accrues en cas de mauvaise performance. » [Driskell et al., 2018].

Ils proposent ainsi « un nombre limité de mécanismes cognitifs, émotionnels et sociaux à travers duquel le stress affecte les performances » des membres d'une équipe, dénommés les « *Big Five* » des mécanismes du stress :

1. Le stress réduit l'attention et augmente la distraction :
   (a) réduction du champ de vision,
   (b) réduction de la mémoire de travail,
   (c) augmentation de la rigidité des performances.
2. Le stress augmente la charge cognitive et la demande de capacité :
   (a) L'augmentation de la charge de travail augmente la charge cognitive/mentale. Plusieurs mécanismes affecte la charge cognitive comme l'*appraisal* de Lazarus.
3. Le stress augmente les émotions négatives et la frustration :
   (a) nécessite un self-contrôle tout en se concentrant sur la tâche,
   (b) la stabilité émotionnelle affecte les comportements collaboratifs.
4. Le stress augmente la peur et l'anxiété :
   (a) impact physiologique : battements de cœur plus élevés, transpiration, tremblement *etc.* .
   (b) augmentation des anxiétés cognitives et somatiques.
5. Le stress accroît les déficiences sociales :
   (a) réduction de la tendance à aider les personnes,
   (b) augmentation des agressions interpersonnelles,
   (c) négligence des indices sociales et interpersonnelles,
   (d) plus de difficulté à se coordonner.

Ils discutent aussi l'impact des processus et de la structure d'une équipe sur les performances :

1. Une équipe avec un modèle mental partagé accroît les performances.
2. La flexibilité de la structure et des rôles d'une équipe est nécessaire pour faire face aux contraintes, mais l'ambiguïté peut aussi amener à de la confusion.
3. Les relations et la cohésion de l'équipe ont un impact prononcé sur les performances : conflits, manque de confiance, difficulté à adopter une orientation commune *etc.*

Ces travaux décrivent un grand nombre de processus et de comportements. De par sa taille et la complexité des intrications de tous ces processus sur le comportement, il est très complexe de définir un modèle computationnel de ces travaux. Toutefois, cela semble compatible avec une opérationnalisation au sein d'un modèle d'agent.

2.1.8 Mini-bilan

Lors de cette section, nous avons présenté plusieurs modèles cognitifs, afin d'étudier ce qui les caractérise. Nous rappelons qu'une partie de nos objectifs est de généraliser et de combiner les modèles cognitifs, pour simplifier leur intégration au sein d'un modèle d'agent quelconque.

Suite à cet aperçu de la littérature SHS, nous retenons qu'un modèle cognitif est une modélisation simplifiée d'un ensemble de processus psychologiques et/ou intellectuelles qui induit des comportements intelligibles et explicables. Certains modèles traitent d'un même aspect, *e.g.* personnalité, émotion, *etc.* , tandis que d'autres en couvrent plusieurs.

Intégrer un modèle cognitif dans un modèle d'agent consiste à introduire les mécanismes nécessaires et suffisants pour retrouver les comportements intelligibles du modèle cognitif sur le comportement de l'agent (produit donc par le modèle d'agent).

Nous notons une première difficulté. Comme l'illustre Mayer par la figure 2.3, les processus cognitifs sont nombreux et couvrent un spectre large et riche : émotions, mémoire, imagination, modèle du monde, conscience, social *etc.* . Chacun affecte le comportement à sa manière. Si nous prenons le modèle ALU de Fadier et al. [Fadier et al., 2003], des aspects externes comme la représentation de l'activité sont aussi affectés. Une autre difficulté majeure est que les modèles ne sont pas indépendants entre eux, *e.g.* les mécanismes du stress décrits par Driskell [Driskell et al., 2015]. Sans connaître les modèles cognitifs, comment les généraliser pour les combiner et les intégrer dans un modèle d'agent que nous ne connaissons pas non plus à l'avance ?

Nous nous tournons désormais vers les modèles d'agents pour étudier la manière dont ils intègrent ces modèles cognitifs.

2.2 Modèles d'agent

Lors de l'introduction, nous avions brièvement introduit la notion d'agent et de modèle d'agent. Dans cette section, nous commençons par décrire plus largement le domaine auquel ces deux notions sont rattachées : les systèmes multi-agents. Ensuite, nous présentons plusieurs modèles d'agent utilisés pour la simulation d'agents virtuels. Nous voulons observer leur manière d'intégrer des modèles cognitifs. Les modèles ont été choisis soit selon leur popularité, soit afin de couvrir un spectre large des différentes manières d'intégrer des modèles cognitifs.

2.2.1 Système multi-agents

Un SMA est une organisation d'agents en société / interaction. Un agent est une entité réelle ou virtuelle, évoluant dans un environnement, capable de le percevoir et d'agir dessus, qui peut communiquer avec d'autres agents, qui exhibe un comportement autonome, lequel peut être vu comme la conséquence de ses connaissances, de ses interactions avec d'autres agents et des buts qu'il poursuit. [3]

Ce domaine de l'Intelligence Artificielle (IA) possède un cadre applicatif large et varié, allant de la simulation du vol des oiseaux (*cf.* figure 2.6 qui illustre une implémentation avec NetLogo [4]) à la simulation d'ambulances, de forces de police et de pompiers, afin de secourir des civils et d'éteindre les incendies dans une ville où un tremblement de terre vient de se produire (*cf.* figure 2.7 extraite de la RoboCup RescueTeam [5]).

Plusieurs champs de recherches sont aussi distinguables selon la notion d'agent, où chacun met l'accent sur différentes propriétés : « intelligent agent » [Weiss, 2013], « embodied agent », « artificial social agent » [Fitrianie et al., 2020]. Ainsi, le domaine des *« embodied agent »* étudie l'importance de l'apparence et des effecteurs, *« artificial social agent »* sur les capacités sociales et expressives, et *« intelligent agent »* sur les capacités de raisonnement et décision. Dans le contexte de la thèse, nous travaillons sur les *« agents intelligents virtuels »*, soit sur les modèles d'agent et les comportements produits.

Plateformes d'agents

Plusieurs plateformes d'agents sont disponibles pour la simulation d'agents virtuels, nous présentons succinctement les plus populaires :

Jade « Jade [6] (Java Agent DEvelopment Framework) est une plateforme logicielle [sous Java] permettant de développer des applications basées sur des agents en confor-

3. Définition tirée de mes cours de master (par Aurélie Beynier).
4. http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/Flocking.nlogo
5. http://rescuesim.robocup.org/competitions/agent-simulation-competition/
6. https://jade-project.gitlab.io/

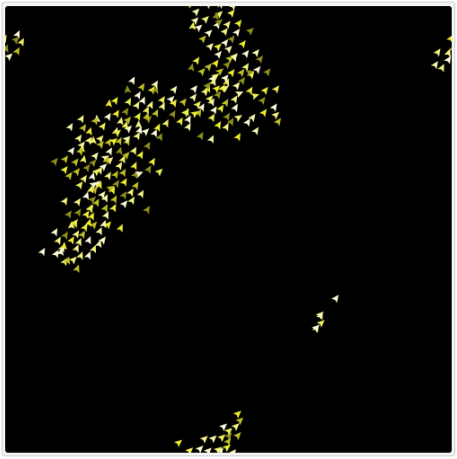


FIGURE 2.6 – Capture d'écran d'une simulation de NETLOGO du flocage des oiseaux.

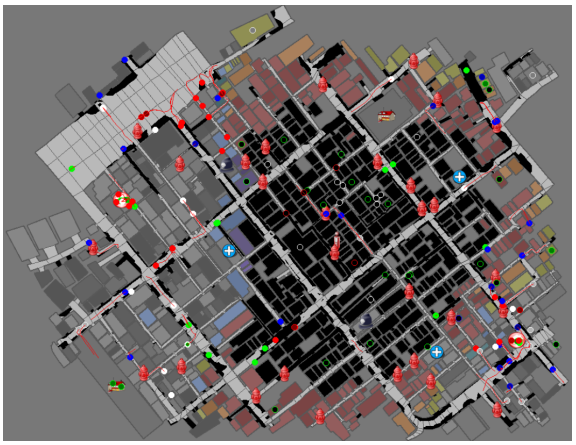


FIGURE 2.7 – Capture d'écran d'une simulation d'une ville ayant subi un tremblement de terre où les agents (pompiers, ambulanciers et policiers) s'organisent pour sauver les civils (*cf.* RoboCup rescue team).

mité avec les spécifications FIPA pour des systèmes multi-agents intelligents interopérables. L'objectif est de simplifier le développement tout en assurant la conformité aux normes grâce à un ensemble complet de services système et d'agents »[7]. Cette plateforme permet des agents mobiles distribués, avec de nombreux mécanismes comme : un système de pages jaunes, le transport des messages asynchrones, l'encodage, l'analyse syntaxique, le cycle de vie des agents, les protocoles de communication, comme des enchères, *etc.* . La figure 2.8 montre un des outils disponibles pour visualiser les agents et écouter leurs conversations. Son développement a débuté dans les années 2000 par le laboratoire *Telecom Italia* puis repris par Cedric Herpson du LIP6 (Laboratoire d'informatique à Paris Sorbonne Université).

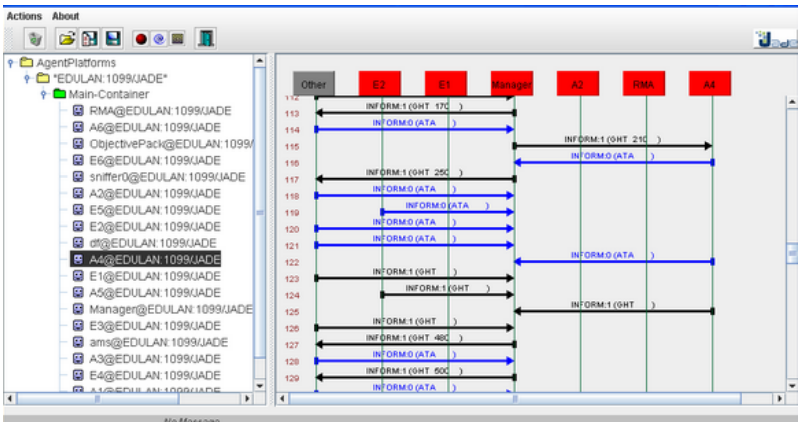


FIGURE 2.8 – Capture d'écran de l'interface de supervision et de communication des agents de JADE.

7. http ://www-desir.lip6.fr/ herpsonc/fr/project/jade/

CAF CAF [Charousset et al., 2016] procure un environnement de programmation en C++ de haut-niveau fondé sur le modèle « Acteur ». CAF est semblable à JADE dans sa possibilité de créer des agents mobiles, distribués, communiquant par messages et avec des comportements. Toutefois, ils ne sont pas compatibles avec les normes FIPA et ne proposent pas de protocoles de communication comme les enchères.

Gama GAMA est un environnement de développement, de modélisation et de simulation pour construire des simulations spatialement explicites fondées sur des agents, utilisant Java [Taillandier et al., 2019] (*cf.* figure 2.9). Cette plate-forme propose notamment un langage haut-niveau et qui est fondé sur la notion d'agent dans le but de faciliter la création de mondes virtuels, d'agents et de comportements. De nombreux travaux utilisent cette plate-forme pour modéliser leurs agents, notamment l'architecture BEN [Bourgais et al., 2021] que nous présenterons lors de cet état de l'art.

FIGURE 2.9 – Quelques exemples d'environnement GAMA proposés sur le site officiel. Cette plate-forme est capable de charger et manipuler des données spatiales et géographiques (GIS).

NetLogo Netlogo [Wilensky, 1999] est un outil de programmation/modélisation de systèmes multi-agents écrit en Scala. Il a été créé en 1999 par Uri Wilensky au Centre pour l'apprentissage connecté et la modélisation assistée par ordinateur, puis par l'université de Northwestern dans les environs de Chigago. « NetLogo est particulièrement bien adapté à la modélisation de systèmes complexes se développant dans le temps. Les modélisateurs peuvent donner des instructions à des centaines ou des milliers d'"agents" fonctionnant tous indépendamment. Cela permet d'explorer le lien entre le comportement des individus au niveau micro et les modèles au niveau macro qui émergent de leur interaction » [8] . Une version de NetLogo [9] est utilisable en ligne, où un grand nombre de modèles SMA sont disponibles. Par exemple, l'outil permet de visualiser l'évolution de la population de moutons et des loup, selon des paramètres modifiables depuis leur interface (*cf.* figure 2.10).

8. http://ccl.northwestern.edu/netlogo/faq.html
9. http://www.netlogoweb.org/launch

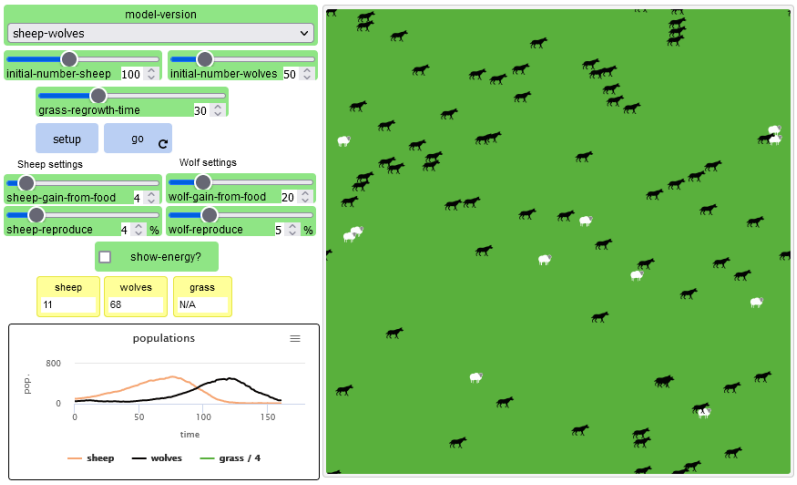


FIGURE 2.10 – Capture d'écran du modèle « Prédation des moutons par des loups » proposé par NETLOGO.

Repast Repast (Recursive Porous Agent Simulation Toolkit) Suite [North et al., 2013] est « une famille de plates-formes de modélisation et de simulation avancées, gratuites et open-source, fondées sur des agents, qui sont en développement continu depuis plus de 20 ans » [10] . Comme GAMA, ils supportent aussi le système de données géographiques (GIS), proposent un grand nombre de fonctionnalités comme des algorithmes génétiques, des réseaux de neurones, des outils de visualisation (*cf.* figure 2.11) et disposent aussi de versions pour tourner sur des clusters et des super-calculateurs. Cette suite a notamment été utilisée pour simuler Chicago avec 2.9 millions d'habitants où chaque habitant est un agent qui se déplace dans la ville, interagit, communique avec les autres agents, s'engage dans des activités, *etc.* [Macal et al., 2018].

10. https://repast.github.io/

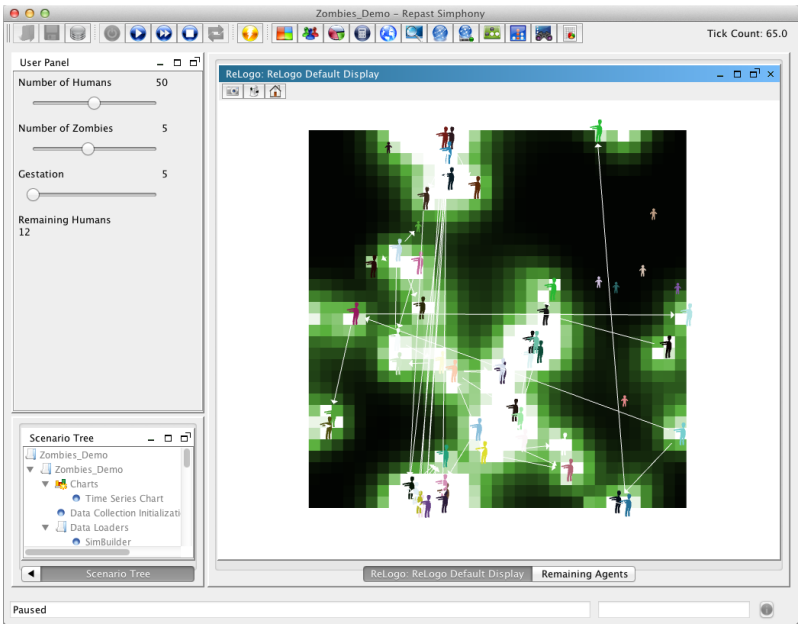


FIGURE 2.11 – Capture d'écran de l'exécution d'un modèle de zombies sous Repast, issue du site officiel.

Modèle d'agent

Nous avons vu en introduction qu'un modèle d'agent est une manière de définir la boucle perception-délibération-action. Ce modèle détermine comment l'agent sélectionne les actions qu'il effectue selon la situation. Généralement, nous allons parler de *données*, l'ensemble des informations manipulées par le modèle. Aussi, nous parlons de *fonctionnement* pour désigner l'ensemble des processus / systèmes qui manipulent ces données pour établir cette boucle. Le tout forme le modèle d'agent (*cf.* figure 2.12) [Russell and Norvig, 2021]).

Attention aux similarités

Parfois, nous parlons de processus dans le cadre d'un modèle d'agent. Il s'agit d'une étape du modèle d'agent, comme la perception, la sélection d'action, *etc.* . Il ne faut pas les confondre avec les processus cognitifs en SHS. La subtilité est que certains processus traduisent des processus cognitifs, mais il s'agit bien de deux choses distinctes.

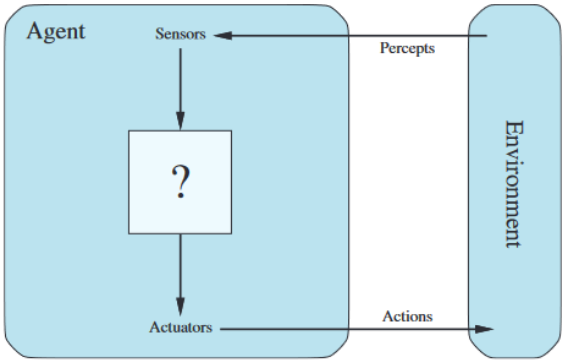


FIGURE 2.12 – Schématisation d'un cycle « Perception-Raisonnement-Sélection » d'un agent selon [Russell and Norvig, 2021].

Un grand nombre de types de modèles d'agent existent dans la littérature : réactif (*cf.* figure 2.13), apprentissage, fondé sur modèle, fondé sur modèle et buts (*cf.* figure 2.14), utilité, cognitif, *etc.*

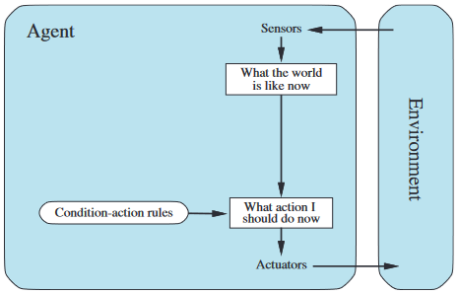


FIGURE 2.13 – Schématisation d'un modèle d'agent réflexe simple selon [Russell and Norvig, 2021]. L'agent agit selon ce qu'il perçoit.

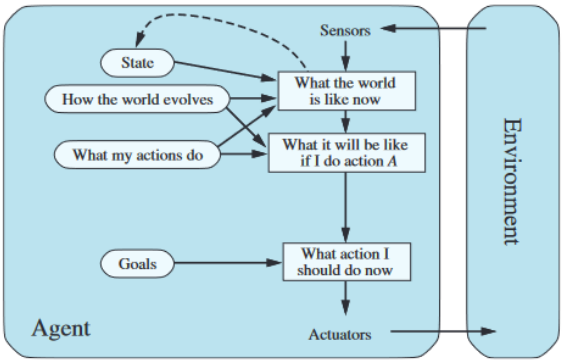


FIGURE 2.14 – Schématisation d'un modèle d'agent raisonnant selon ses buts et un modèle du monde [Russell and Norvig, 2021].

Chaque modèle a ses avantages et inconvénients : un modèle d'agent réactif est rapide et simple à mettre en place, un modèle fondé sur l'utilité simule un raisonnement rationnel, un modèle d'agent cognitif est plus coûteux et complexe à mettre en place, mais génère des comportements plus humains, *etc.* Il convient d'utiliser le modèle adéquat selon ses besoins, soit les comportements recherchés et ses contraintes, notamment selon l'environnement à disposition.

Environnement

À la suite de l'état de l'art des différentes définitions proposées dans la communauté SMA, les auteurs de [Weyns et al., 2006] proposent la définition suivante de l'environnement :

Citation 10 - Environnement

« L'environnement est une abstraction de première classe qui fournit les conditions ambiantes permettant aux agents d'exister et qui gère à la fois l'interaction entre les agents et l'accès aux ressources. »

Weyns, D., Omicini, A., and Odell, J. (2006). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*

Ainsi, l'environnement permet aux agents d'exister et d'interagir entre eux. Il contraint aussi les interactions. Par exemple, dans le contexte d'une simulation, l'environnement peut limiter les perceptions pour qu'un agent ne puisse pas voir aux travers d'un mur. Nous parlons alors d'environnement à observabilité partielle. Un autre exemple est qu'un agent distant ne peut directement communiquer, *e.g.* par la voix, avec un autre agent.

Toutefois, l'environnement peut aussi être composé d'un autre type d'entités que les agents : les artefacts. Ce sont des éléments, des objets dans l'environnement qui fournissent un ou des services aux agents [Omicini et al., 2008], *e.g.* : une paire de ciseaux, un bandage ou un téléphone pour permettre à notre agent distant de communiquer avec un autre agent, *etc.*

Dans le cadre de la robotique, l'environnement correspondrait à notre réalité, notre monde. Dans le contexte de la thèse et en simulation, nous travaillons avec des environnements virtuels, notamment des environnements dits « intelligents ». Nous pouvons disposer d'un modèle du monde sur lequel nos agents peuvent raisonner. Par exemple, les travaux de [Okreša Đurić et al., 2019] mettent en place une ontologie pour décrire les éléments qui composent le monde : cet objet est une chaise, tangible, donc perceptible, avec lequel je peux réaliser l'action « s'assoir sur ».

Une autre manière de procéder consiste à ajouter des informations dans les objets ou artefacts de l'environnement directement, sans passer par une ontologie. Ainsi, Abaci et al. [Abaci et al., 2005, Baber, 2018] propose la notion de « smarts objects ». Les objets de l'environnement émettent des affordances [Gibson, 1977, Gaver, 1991, Abaci et al., 2005] avec les conditions d'exécution et les effets, ce qui permet aux agents de planifier leurs actions, afin d'accomplir leurs buts. [11]

Nous concluons cette parenthèse sur les SMA par la figure 2.15. Elle illustre un exemple des concepts discutés : l'environnement est composé de quatre agents : un oiseau et trois insectes, et d'un artefact : la grappe de raisins permettant de satisfaire la faim des insectes. L'environnement dispose les agents dans un plan en deux dimensions où ils peuvent se mouvoir. Il contraint la vision des agents à ce qui est visible, avec un mur qui occulte la vision. Un modèle d'agent possible pour les insectes serait un modèle d'agent réactif : « "si prédateur, alors fuir" », « "si nourriture, alors manger" ».

11. Dans l'industrie, le jeu « Les Sims » est un exemple d'applications utilisant ce principe [noa, b]. Les objets émettent les aspects qu'ils peuvent satisfaire (faim, joie, *etc.* .) et le Sims pondèrent selon ses besoins.

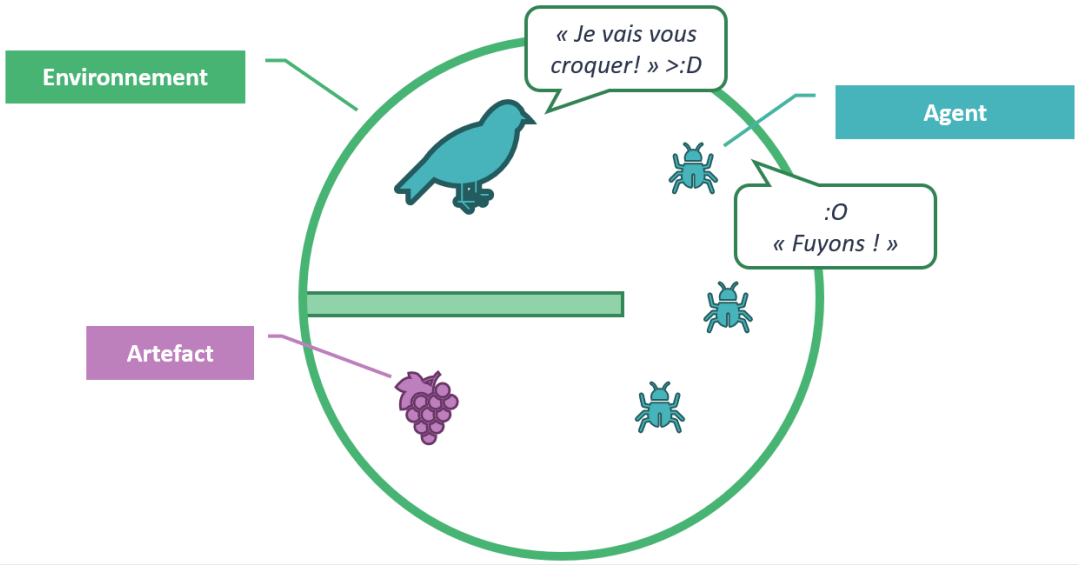


FIGURE 2.15 – Exemple illustratif d'un SMA composé de quatre agents et d'un artefact.

2.2.2 Simulation d'agents virtuels

La littérature des SMA dispose d'un grand nombre de modèles d'agent, allant des approches ad-hoc aux architectures cognitives. Nous nous focalisons ici sur les approches cognitives. Nous allons observer leur manière d'intégrer des modèles cognitifs, dans le but de généraliser et de simplifier l'intégration et la combinaison de modèles cognitifs. Nous commençons par présenter quelques approches BDI [Bratman, 1987] , puis des architectures cognitives et enfin d'autres approches hors catégories.

Modèle BDI

En 1987, Bratman [Bratman, 1987] propose un modèle de raisonnement, nommé BDI pour « Belief » (croyance), « Desire » (désir) et « Intention » (intention).

Un agent BDI possède des croyances sur l'état du monde et sur son état interne. Selon ses désirs (ou ses buts), l'agent va s'engager à réaliser ses intentions, en suivant des plans (i.e séquence d'actions) qui peuvent les satisfaire. Si nous réemployons les schémas de modèles d'agent *à la Russel et Norvig* [Russell and Norvig, 2021], nous obtiendrons la figure 2.16.

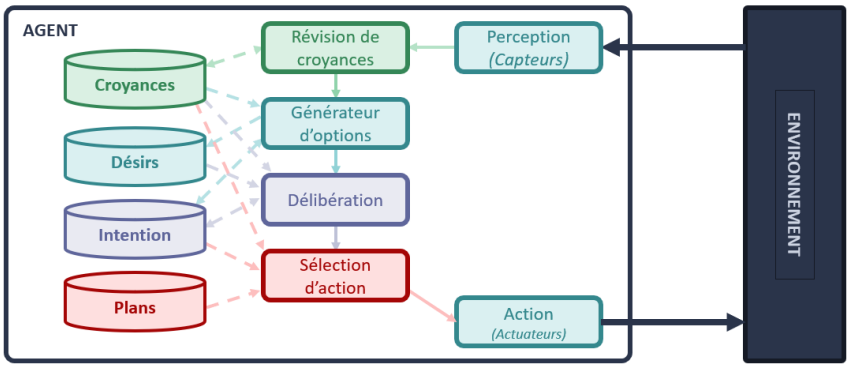


FIGURE 2.16 – Schématisation d'un modèle d'agent BDI [Bratman, 1987] .

Ce schéma est cependant une vision haut-niveau du modèle d'agent. Une implémentation aurait une forme quelque peu différente comme l'illustre la figure 2.17. Il s'agit d'une implémentation de BDI [Bratman, 1987] , réalisée au sein de la plate-forme GAMA [Taillandier et al., 2016].

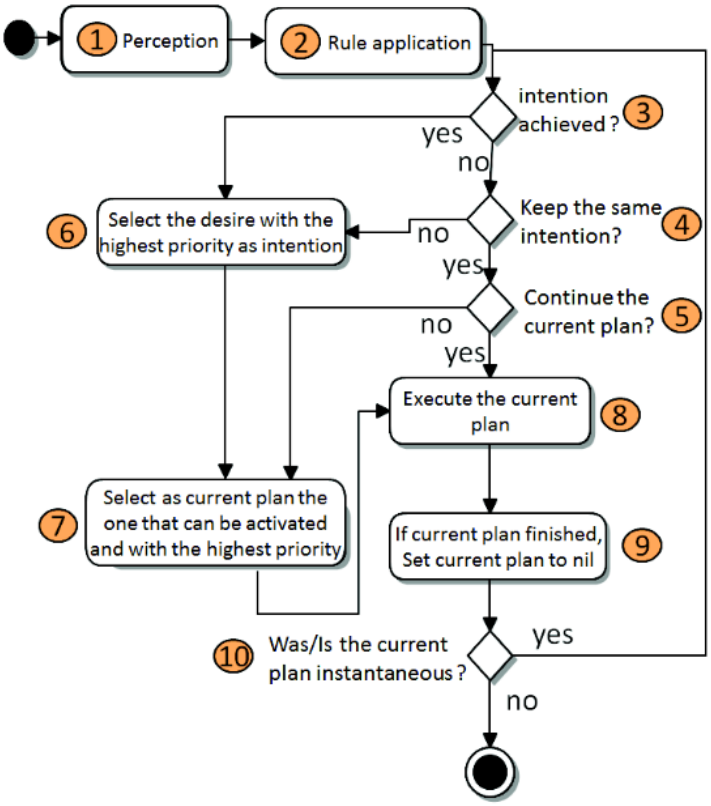


FIGURE 2.17 – Diagramme d'activité de l'implémentation du modèle BDI au sein de la plate-forme GAMA [Taillandier et al., 2016].

En simulation sociale, les approches BDI sont les plus utilisées [Adam and Gaudou, 2016]. Parmi ses avantages, il y a son aspect plus orienté psychologie et son abstraction plus haut niveau, qui facilitent la transition entre la connaissance des experts et

le comportement des agents. C'est pourquoi de nombreux travaux se fondent sur cette architecture [Adam and Gaudou, 2016, Herzig et al., 2016, Larsen, 2019]. Nous présentons quelques-unes de ces approches.

Modèles fondés sur le modèle BDI [Bratman, 1987]

Le modèle BDI a été le sujet de nombreuses extensions [Adam and Gaudou, 2016], afin d'ajouter de nouvelles dimensions dans le raisonnement. Nous en allons en voir quelques-unes, en commençant par EBDI.

EBDI [Jiang et al., 2007] L'approche EBDI [Jiang et al., 2007] intègrent une dimension émotionnelle dans le raisonnement BDI (*cf.* carré rouge dans la figure 2.18) : avec une phase de mise à jour des émotions (spécifiquement deux évaluations : primaire et secondaire) et l'ajout des émotions dans le processus de délibération. L'approche ne précise pas les émotions, ni comment elles sont actualisées, ni comment elles affectent le processus de délibération. C'est à la liberté du modélisateur.

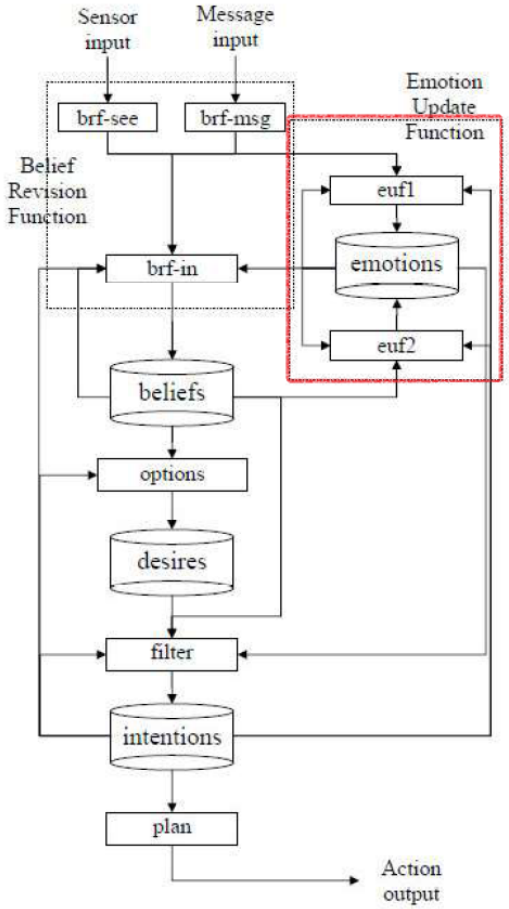


FIGURE 2.18 – Fonctionnement de l'approche EBDI [Jiang et al., 2007].

Nous voyons ici une première manière de généraliser des modèles cognitifs (uniquement émotionnels ici). L'approche propose une structure fondée sur le modèle BDI

avec une partie dédiée à l'évaluation des émotions et la modification du processus de délibération, qui prend désormais en compte les émotions. Cela suppose que cette structure suffit pour intégrer de modèles cognitifs émotionnels. Regardons désormais l'approche ABC-EBDI pour comparer.

ABC-EBDI [Sánchez et al., 2019] ABC-EBDI [Sánchez et al., 2019] est une approche plus récente qui intègre le modèle de psychologie thérapeutique ABC dans un modèle de raisonnement BDI. Comparé à EBDI [Jiang et al., 2007], ils intègrent, en plus des émotions, la notion de personnalité et d'humeur. Le fonctionnement est résumé par la figure 2.19.

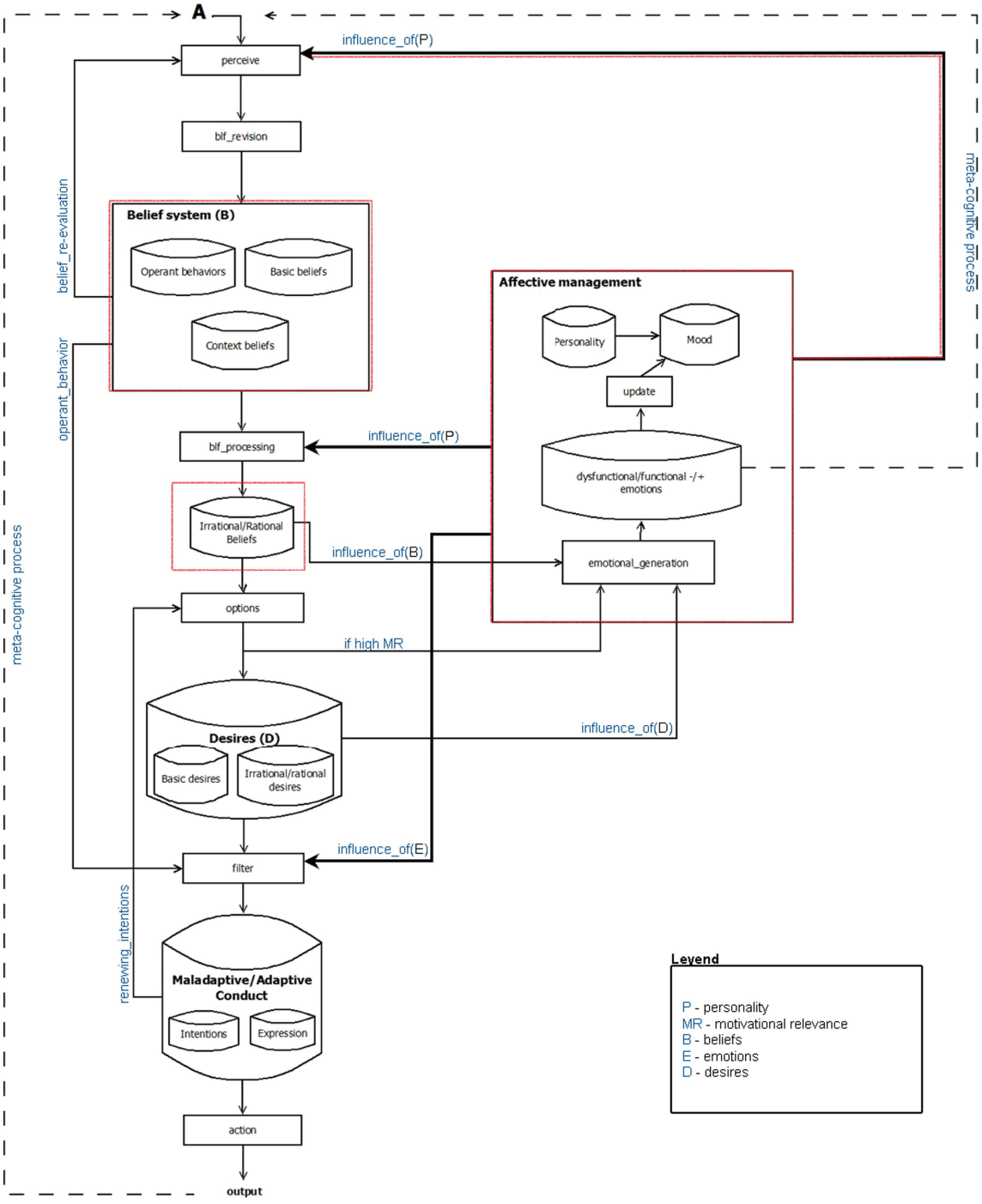


FIGURE 2.19 – Fonctionnement de l'approche ABC-EBDI [Sánchez et al., 2019].

Observations Arrêtons-nous un moment sur ce que nous pouvons déjà apprendre de ces deux approches. Toutes deux s'intéressent à l'intégration de l'affectif dans le raisonnement BDI : EBDI uniquement aux émotions, tandis que ABC-EBDI aux émotions, à l'humeur et à la personnalité (avec OCEAN). EBDI propose une manière d'intégrer génériquement des modèles cognitifs émotionnels (*cf.* carré rouge dans la figure 2.18). Si

on la compare avec la figure 2.19, nous retrouvons bien ce même carré. Il est compacté dans le processus « emotionnal_generation », situé dans le carré rouge « Affective management » de la figure 2.18. Nous pouvons remarquer que la manière de généraliser des modèles cognitifs émotionnels de eBDI n'a pas suffi. Ils ont notamment ajouté un impact du bloc vers la perception.

Il y a aussi quelques différences notables (non exhaustives), dues aux dimensions supplémentaires ajoutées, autre que les émotions :

— Des structures de données ont été ajoutées : « Personnality » et « Mood »

— Les « Beliefs » et sa mise à jour sont complexifiées. Il y a désormais deux processus au lieu d'un : révision + traitement, ainsi que de nouvelles structures de données, *e.g.* « Irrational/Rational Belief ».

Vis-à-vis de nos questions de recherche, nous notons donc les observations suivantes :

1. Une généralisation d'un type de modèles cognitifs, ici émotionnels, était trop restreinte, notamment sur les impacts sur les autres processus.
2. L'intégration d'un modèle cognitif, toujours d'émotions, a nécessité la modification du modèle d'agent par l'ajout de processus et/ou de structures de données, e.g « Affective Management ».
3. L'intégration d'un modèle cognitif à amener à modifier des processus déjà existants, *e.g.* mise à jour des « Beliefs ».

Architecture BEN [Bourgais et al., 2021] Une autre architecture fondée sur BDI est l'architecture BEN [Bourgais et al., 2021]. Elle intègre les aspects suivants : cognition, émotion (OCC [Ortony et al., 1988]), personnalité (OCEAN [Costa Jr. and McCrae, 2008]), contagion émotionnelle, normes et relations sociales ([Ochs et al., 2009]).

Une force de ce modèle est liée au choix de conception à relier les différents modèles cognitifs aux paramètres du modèle OCEAN [Costa Jr. and McCrae, 2008]. L'avantage est ainsi de limiter les paramètres à définir (contrairement à l'approche PMFServ que nous présentons juste après), tout en permettant d'obtenir des comportements cognitifs, affectifs et sociaux.

Une autre force de cette architecture est d'offrir une certaine modularité : le modélisateur peut choisir de n'utiliser que la prise de décision sans tenir compte d'autres processus comme de l'évaluation d'émotions par exemple. Toutefois, l'ensemble des opérations dépend de l'approche BDI : il est impossible d'envisager une autre architecture de prise de décision.

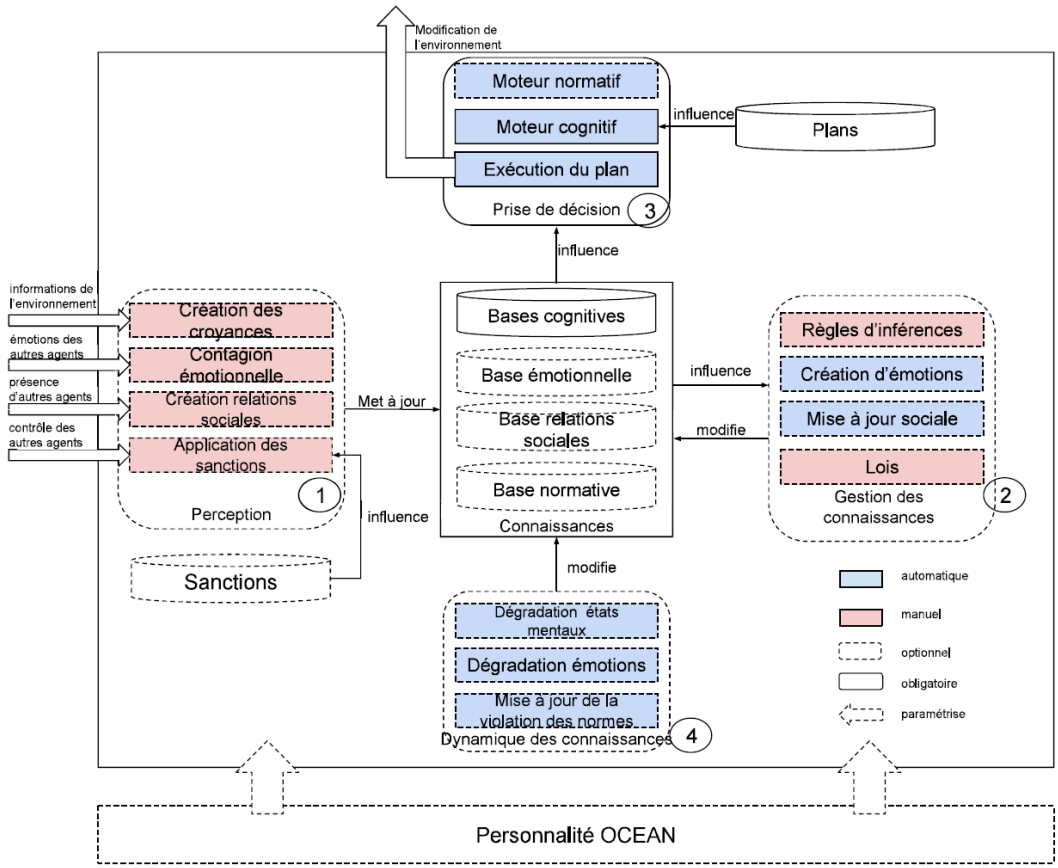


FIGURE 2.20 – Schéma de l'architecture BEN [Bourgais et al., 2021].

PMFServ Silverman a réalisé de nombreux travaux dans la simulation des comportements humains [Silverman et al., 2006b, Silverman et al., 2006a, Silverman et al., 2012] dans un contexte de formation. Une de ses applications était la simulation d'une société autonome dans un village afin de supporter des objectifs de formation inter-culturelles [Silverman et al., 2012]. Ils intègrent de nombreuses dimensions : cognitive, sociologique, économique, politique, *etc.* Silverman soulève qu'un modèle d'agent « One size fits all » semble impossible [Silverman et al., 2012]. Dans ce sens, il développe une plate-forme dénommée « PMFServ ». Il s'agit d'une « architecture permettant de synthétiser de nombreux modèles et théories des meilleures pratiques de modélisation du comportement humain » [Silverman et al., 2006b]. Il intègre ainsi OCC pour les émotions, l'utilisation de la théorie des affordances [Gibson, 1977, Gaver, 1991, Abaci et al., 2005], *etc.*

Chaque processus cognitif est assimilé à une fonction PMF pour « Performance Moderator Function ». Elle indique comment un aspect humain fonctionne (perception, mémoire, attention, *etc.*). « PMFserv synthétise des dizaines des meilleurs "PMF" dans un cadre unifié corps-esprit et offre ainsi une famille de modèles. Aucun de ces "PMF" n'est fait-maison ; ils sont plutôt issus de la littérature des sciences du comportement » [Silverman et al., 2012]. Ainsi, il utilise, par exemple, pour la partie mémoire, les concepts

du modèle BDI.

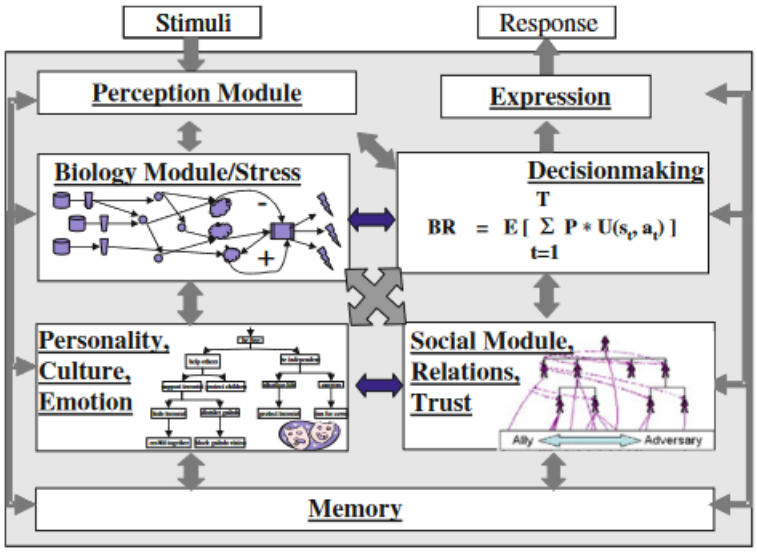


FIGURE 2.21 – Exemple d'une architecture obtenue.

L'avantage de cette approche est sa direction unificatrice, mettant en œuvre de multiples PMFs (« biologique/stress, affectif, cultruel, social, cognitifs, personnalité ») et paramètres interagissants tous ensemble. Ils disposent ainsi d'agents « très adaptables » [Silverman et al., 2006a], sans avoir besoin d'apprentissage. Toutefois, les auteurs soulèvent que plusieurs difficultés sont présentes. Tout d'abord, les comportements sont les résultats de nombreux PMFs et sous-systèmes qui interagissent entre eux. Il est donc ardu d'expliquer le comportement généré d'une part et de paramétrer la simulation d'autre part (comparé à une approche comme BEN et son choix de lier les paramètres au modèle OCEAN). Ils précisent que pour y arriver, il faut parfaitement connaître le fonctionnement de chaque PMF. C'est ce que nous souhaitons justement faciliter et rendre modulaire.

Architectures cognitives

Lors de cette section, nous présentons succinctement 3 architectures cognitives : CLARION [Sun, 2006], ACT-R [Anderson, 1996] et SOAR [Laird et al., 1986], suivi d'une première observation concernant ces trois approches en mêmes temps. Ensuite, nous présenterons les travaux de Silverman et al. [Silverman et al., 2006b, Silverman et al., 2012].

CLARION [Sun, 2006] CLARION [Sun, 2006] est une architecture développée par Sun depuis 1998. Elle cherche à reproduire les mécanismes humains de création de connaissance : l'acquisition de connaissances implicites permettant la création de connaissances explicites réutilisables. Elle est composée de quatre sous-systèmes : le module centré action (ACS), le module non-centré action (NACS), le module motivationnel (MS) et le module de méta-cognition (MCS).

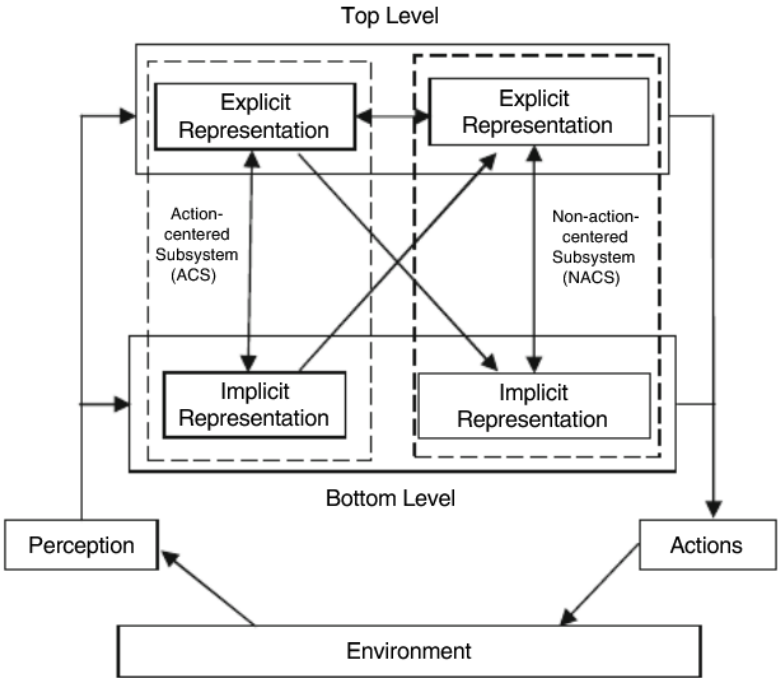


FIGURE 2.22 – Schéma de l'architecture CLARION tiré de [Chong et al., 2007]

ACT-R [Anderson, 1996] ACT-R [Anderson, 1996] est un autre exemple d'architecture cognitive. Elle dispose d'un module sensori-moteur (perception/action), d'un module de formation de but et d'un module de mémoire déclarative et procédurale. Chaque module est associé à une région du cerveau humain et en simule le fonctionnement. Le principe utilisé est de reconnaître des schémas à partir des éléments fournis par les différents modules et de sélectionner une règle de production associée à un schéma reconnu dans la situation grâce au moteur sub-symbolique. L'amélioration des processus sub-symboliques est assimilée à l'apprentissage.

SOAR [LAIRD ET AL., 1986] SOAR [LAIRD ET AL., 1986, LAIRD, 2022] est une architecture cognitive. Elle contient une mémoire à court terme qui contient l'évaluation de la situation par l'agent, obtenue par sa perception de la situation, traitée à l'aide de ses connaissances stockées dans sa mémoire à long terme. Cette architecture flexible permet une certaine forme de méta-cognition en transformant les comportements réfléchis en comportements réactifs et combine des raisonnements symboliques avec des méthodes numériques permettant de traiter et de créer des règles qu'il aurait été compliqué de créer sans apprentissage. À la différence de CLARION, SOAR doit être alimenté par des connaissances procédurales. Les modules d'apprentissage créent de nouveaux liens ou déterminent de nouveaux éléments déclenchant les procédures.

Observations Nous retrouvons dans ces architectures la boucle perception-action. Toutefois, contrairement aux approches BDI, la partie entre la perception et l'action diffère considérablement d'une architecture cognitive à une autre, ainsi que vis-à-vis de

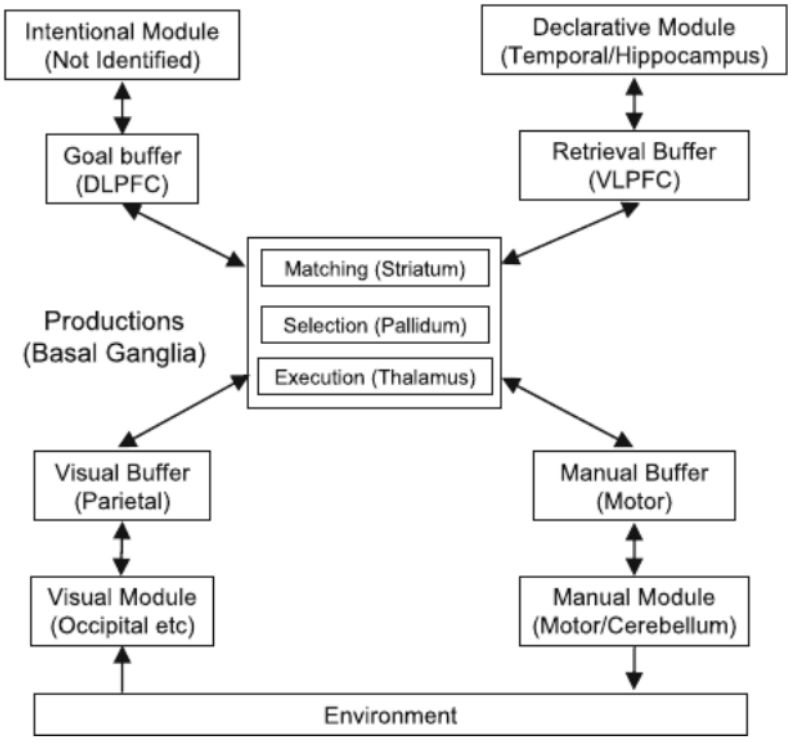


FIGURE 2.23 – Schéma de l'architecture ACT-R tiré de [Chong et al., 2007]

BDI.

Chacune de ces architectures se fonde sur des travaux en sciences cognitives et en biologie pour représenter le cerveau humain le plus précisément possible [Adam and Gaudou, 2016]. Cela rend leur utilisation plus complexe, notamment dans un contexte de formation où le processus de décision est moins important que l'expression de comportements précis. Nous ne cherchons pas à expliquer les comportements humains, mais à rendre compte de certains de ces comportements et à ce qu'ils soient cohérents et compréhensibles à des fins de formation, pour que l'apprenant comprenne l'impact de ses décisions sur le collectif.

Enfin, l'inconvénient de ces architectures cognitives est le couplage fort avec les modèles cognitifs ou psychologiques sur lesquelles elles se fondent. Le risque est d'entrer en conflit avec l'implémentation et le couplage de différents modèles cognitifs. Les comportements issus du système pourront être moins intelligibles et le système plus complexe à maintenir/utiliser.

Autres approches

Dans cette sous-section, nous présentons d'autres approches qui ne rentrent pas strictement dans les deux autres catégories, mais non moins importantes.

Replicants REPLICANTS [Huguet et al., 2018] est une approche hybride entre une approche scriptée et cognitive. Ce modèle d'agent raisonne sur le formalisme ACTIVITY-

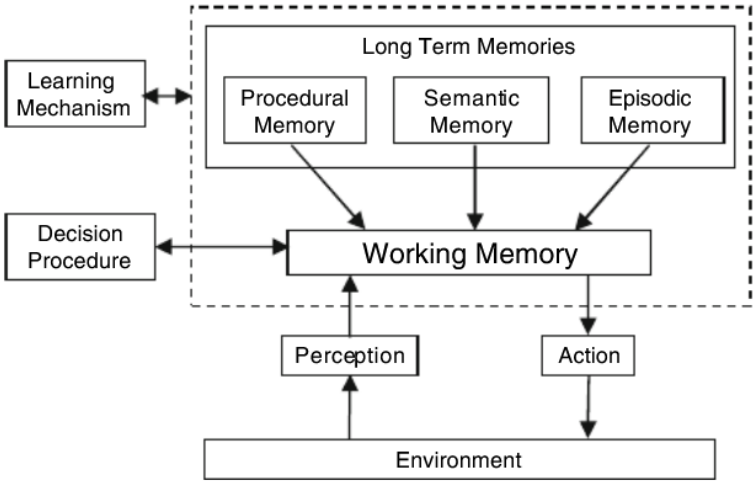


FIGURE 2.24 – Schéma de l'architecture SOAR tiré de [Chong et al., 2007]

DL, inspiré des HTNs [Erol et al., 1994]. Ce formalisme modélise l'activité humaine de manière hiérarchique, proche de la représentation mentale des experts.

Le parcours de l'arbre d'activité retourne un ensemble de tâches envisageables, ainsi que leur priorité, évaluée selon les pré-conditions.

Plus précisément, les tâches ou les buts sont décomposés en sous-tâches et les feuilles de l'arbre représentent les actions unitaires (*cf.* figure 2.25). L'organisation des sous-tâches est réalisée par des opérateurs logiques ET et OU, ainsi que des opérateurs temporels SEQ, ORD, SEQ-ORD, etc. De plus, chaque tâche possède une multitude de pré-conditions : contextuelles, réglementaires, favorables, nomologiques ; ce qui permet d'affiner l'évaluation de chaque tâche.

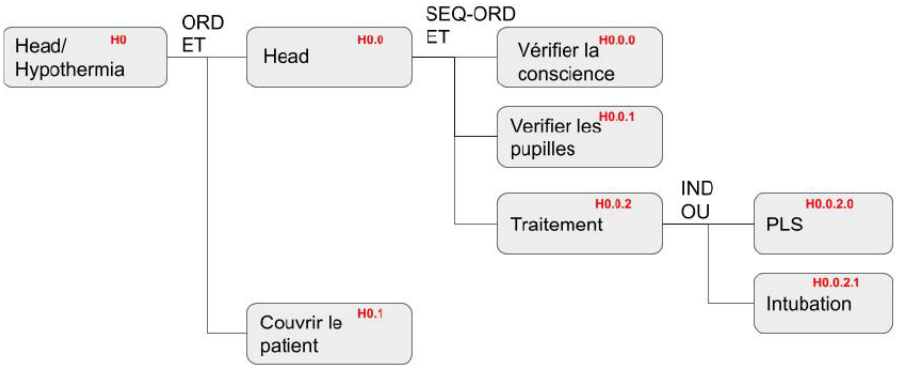


FIGURE 2.25 – Exemple d'un arbre d'ACTIVITY-DL dans le cadre du projet VICTEAMS [Huguet et al., 2016], où une procédure médicale simplifiée est représentée.

Cet arbre représente les différentes alternatives pour mener à bien une tâche. Cette variation est exploitable selon l'évaluation des pré-conditions, des opérateurs logiques et temporels. L'algorithme de sélection d'action évalue les tâches candidates, pertinentes à un moment donné de la simulation, avant de sélectionner celles dont le score ou l'utilité

est le plus important, établie par un calcul de priorité.

L'intégration de modèles cognitifs dans REPLICANTS se fait en modifiant ce calcul de priorité par l'ajout de bonus/malus en fonction du profil de l'agent et des propriétés de la tâche considérée. Par exemple, l'incorporation du modèle Demary [Demary, 2020] ajoute un bonus lorsque la tâche correspond à un ordre et que l'agent a un profil passif. Le modèle de Fadier [Fadier et al., 2003] applique un malus selon les conditions réglementaires et le profil de l'agent. Les conditions réglementaires ont été rajoutées dans le modèle d'activité, ainsi que différents tags.

Ce formalisme permet de rendre compte d'aspects humains en étant très expressif. Enfin, il est intelligible et interprétable, contrairement à des langages purement issus des travaux en ergonomie cognitive. Cependant, intégrer de nouveaux modèles cognitifs dans ce modèle, nécessite de réviser le calcul de priorité. La difficulté est alors de s'assurer que les modèles restent toujours sensibles et que les comportements restent intelligibles.

Méta-modèles En 2005, Bernon et al. [Bernon et al., 2005] étudient trois méta-modèles : ADELFE, GAIA and PASSI. Un méta-modèle est défini comme suit :

Citation 11 - Méta-modèle

« Une représentation structurelle des éléments [d'un SMA] : agent, rôle, comportement, ontologie, etc. »

Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., and Zambonelli, F. (2005). A Study of Some Multi-agent Meta-models. In Odell, J., Giorgini, P., and Müller, J. P., editors, *Agent-Oriented Software Engineering V*, Lecture Notes in Computer Science, pages 62–77, Berlin, Heidelberg. Springer

Ils avancent que la communauté SMA bénéficierait d'une standardisation. Ainsi, il propose un autre méta-modèle unifiant les aspects les plus intéressants et communs des trois méta-modèles précédents :

- ADELFE et GAIA partagent les notions de communication et d'environnement.
- GAIA et PASSI partagent les notions de rôles et de services.
- GAIA est le seul des trois à avoir la notion de responsabilités.
- PASSI est le seul des trois à avoir la notion d'ontologie.
- ADELFE est le seul des trois à avoir la notion de représentation des autres.

L'avantage de cette approche est donc de proposer une structure standardisée pour les éléments d'un SMA. Cependant, il n'y a pas de modèle d'agent associé à ce méta-modèle. Il souligne que cela serait trop dépendant des approches et que le méta-modèle est un point de départ pour définir les systèmes sous-jacents. Il soulève aussi trois questions de recherches :

- « Est-il possible d'identifier un méta-modèle à partir duquel tous les méta-modèles utilisés dans la communauté multi-agents pourraient être dérivés ? » [Bernon et al., 2005]

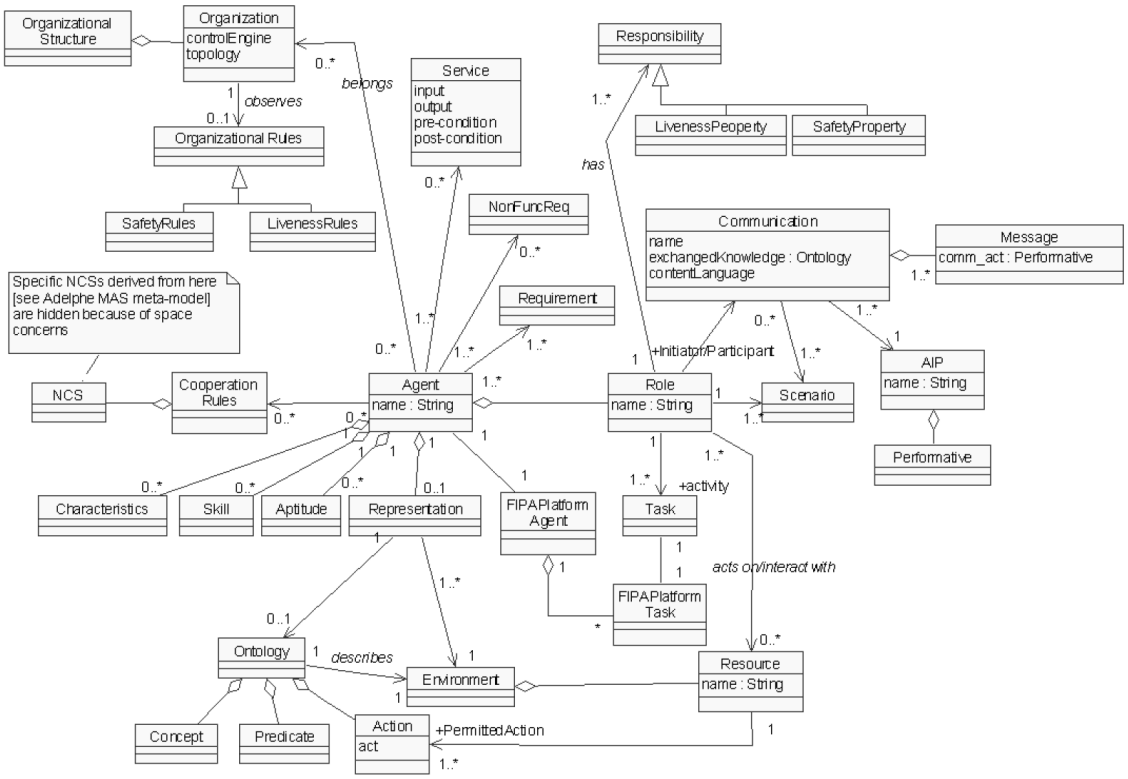


FIGURE 2.26 – Méta-modèle de [Bernon et al., 2005] unifiant trois autres méta-modèles.

— « Quel niveau de description doit être atteint dans le méta-modèle ? Par exemple, les compétences et les aptitudes dans ADELFE sont certainement utilisées pour mettre en œuvre la notion de rôle de GAIA ou PASSI » [Bernon et al., 2005].

— « Comment un concepteur peut-il choisir les éléments du méta-modèle qui l'intéressent ? Quels types d'outils pouvons-nous lui fournir pour faciliter ses choix ? » [Bernon et al., 2005].

2.2.3 Mini-bilan

Lors de cette section, nous avons présenté plusieurs modèles d'agent, afin d'étudier la manière dont ils intègrent des modèles cognitifs.

Nous rappelons qu'un modèle d'agent est une manière de définir une boucle perception-délibération-action qui amène l'agent à sélectionner une action selon son profil et la situation. Nous rappelons aussi qu'intégrer un modèle cognitif dans un modèle d'agent consiste à introduire les mécanismes nécessaires et suffisants pour retrouver les implications intelligibles du modèle cognitif sur le comportement de l'agent (produit donc par le modèle d'agent).

À la suite de cet aperçu de la littérature SMA, nous avons pu constater différentes manières d'intégrer divers modèles cognitifs : social, émotionnel, personnalité, *etc.* Cela

a été fait par la proposition d'une nouvelle boucle perception-délibération-action, avec de nouveaux processus et de nouvelles données, établissant ainsi un nouveau modèle d'agent. Il n'existe pas une manière de faire.

Certains modèles, comme EBDI [Jiang et al., 2007] ont tenté de proposer des approches pour intégrer de manière modulaire des modèles cognitifs émotionnels. Nous avons vu, toutefois, que cela n'a pas suffi, puisque des approches comme ABC-EBDI [Sánchez et al., 2019] ont dû aller plus loin que ce que procurait la modularité de EBDI. Existe-t-il un moyen plus efficace et plus générale que juste les modèles cognitifs émotionnels ?

Nous avons vu aussi que l'intégration de modèles cognitifs dans certains modèles d'agents est complexifiée par le couplage fort entre tous les modèles et la nécessité de revoir des parties du système comme des calculs de priorité. Ce dernier point est la raison pour laquelle nous avons aussi regardé les méthodes de sélection d'action, pour étudier les manières modulaires et intelligibles de considérer les impacts de modèles cognitifs lors de cette étape.

2.3 Sélection d'actions

Le dernier aspect que nous allons étudier lors de cet état de l'art, sont les méthodes de sélection d'action. Il nous semble nécessaire de consacrer une partie de notre revue de la littérature à l'étude des méthodes de sélection d'action. En effet, il est important de comprendre si ces méthodes peuvent nous permettre d'intégrer de manière modulaire et intelligible des modèles cognitifs lors de cette étape. Nous présentons les méthodes les plus populaires, ainsi que celles qui nous ont inspirés. Il ne s'agit pas d'une liste exhaustive, nous vous conseillerons plutôt des ressources comme Game AI PRO 2021 [12] .

2.3.1 Behaviour Tree (BT)

Un Behaviour Tree (BT) [Millington, 2019] est un arbre composé de nœuds :

- de contrôle de flux : sélecteur, séquence, *etc.* ,
- et d'exécution correspondant aux tâches.

Cet arbre permet la sélection d'une tâche parmi un ensemble fini de tâches, selon comment les conditions des différents nœuds sont satisfaites et selon les nœuds de contrôle (*cf.* figure 2.27 pour un exemple de *behaviour tree.*)

Un article de Collendanchise [Colledanchise and Ögren, 2018] introduit en détail les BT et compare leur utilisation à des techniques classiquement utilisées, comme les Finish State Machine (FSM) (que nous ne présentons donc pas ici), les architectures par subsomption, les arbres de décision, *etc.* Selon eux, les BT permettent de généraliser ces différentes approches. Ensuite, ils présentent plusieurs extensions, comme les

12. http://www.gameaipro.com/

BT stochastiques pour des comportements dont l'issue est probabiliste, ainsi que son utilisation pour l'apprentissage et la planification.

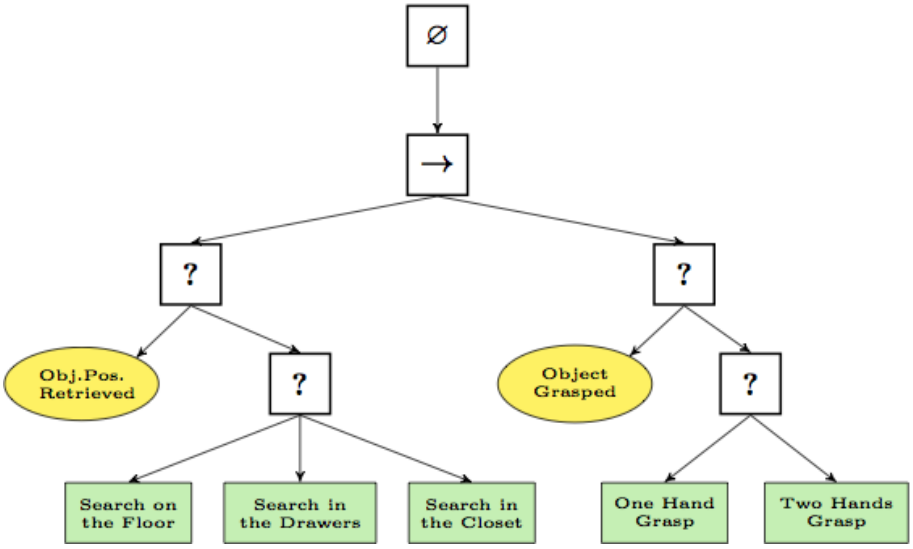


FIGURE 2.27 – Exemple d'un BT pour trouver et prendre un objet. Tiré de Wikipédia

L'utilisation de cette technique est populaire dans le domaine du jeu vidéo, ainsi que de la robotique [Colledanchise and Ögren, 2018], notamment pour son succès pour générer des comportements réactifs et modulaires pour des agents autonomes [Colledanchise and Ögren, 2018].

Aussi, les avantages de cette approche sont son expressivité, sa modularité, son expressivité, la lisibilité et son interprétabilité. Toutefois, la taille explose rapidement, surtout lorsque des variations dépendent d'intervalles numériques (*cf.* figure 2.28).

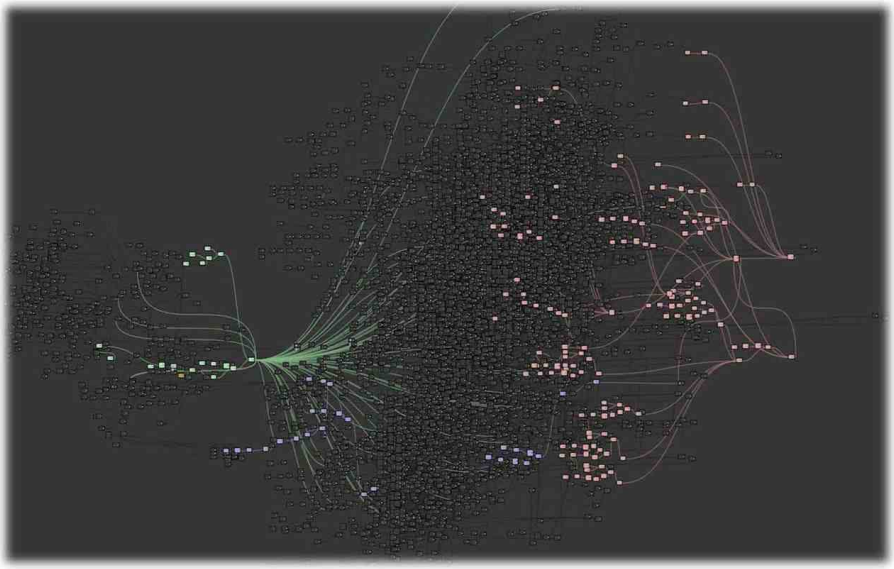


FIGURE 2.28 – Exemple d'un BT issue du jeu « The division ».

2.3.2 Approche par utilité

Une autre technique, présentée comme une alternative aux BT [13], est l'approche par utilité, par exemple utilisée dans *Dragon Age : Inquisition*, un RPG triple A [Hanlon and Watts, 2017]. Son principe consiste à évaluer le score d'un ensemble d'actions et de sélectionner les actions dont le score est le plus élevé. Cela permet dans certaines situations de simplifier la lisibilité et la maintenance, notamment lorsque le choix dépend de nombreux intervalles numériques, ce qui nécessiterait un BT de taille importante (*cf.* figure 2.29). Un désavantage cependant, est que nous devons pouvoir quantifier l'utilité d'une action face à une autre et enfin de pouvoir calibrer les scores pour obtenir les comportements souhaités.

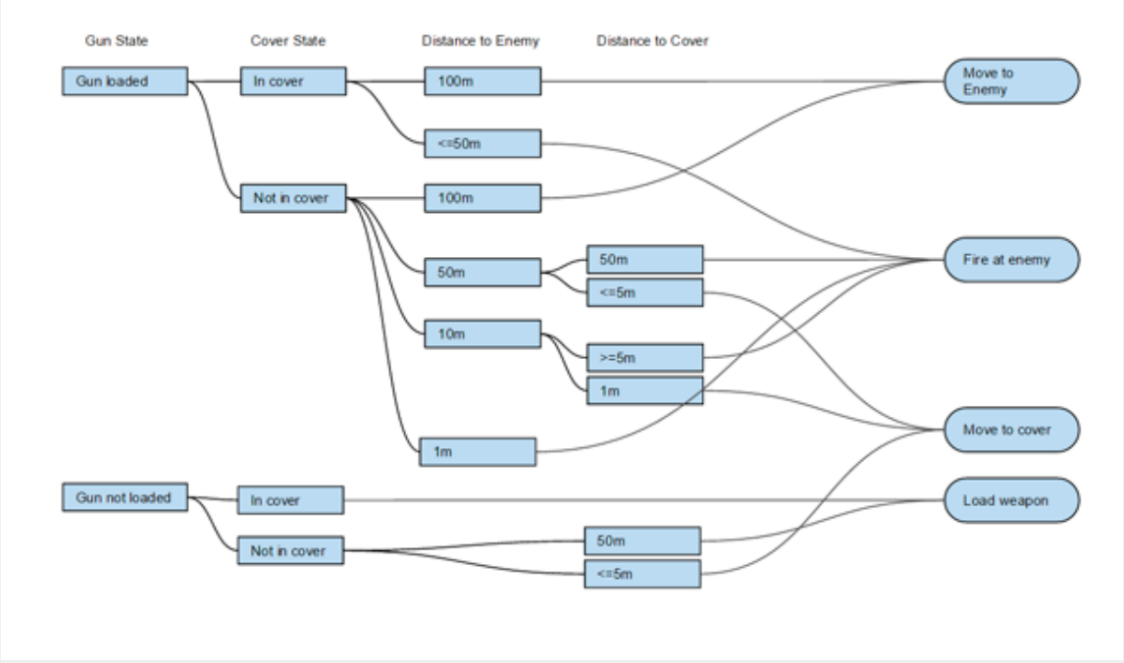


FIGURE 2.29 – Exemple d'un BT dont la lisibilité devient moindre au fur et à mesure que le nombre de conditions augmente. Tiré de https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php

2.3.3 Graphes de motivation

Une autre méthode est l'utilisation de graphes motivationnels [14] . L'agent possède un ensemble de motivations qui propagent dans un graphe de nœuds une valeur

13. https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php
14. chiva2003motivation

d'activation. Les nœuds décident, à partir de règles, la valeur propagée, jusqu'aux prochains nœuds, ainsi de suite, jusqu'à des nœuds feuilles, qui correspondent aux actions (*cf.* figure 2.30).

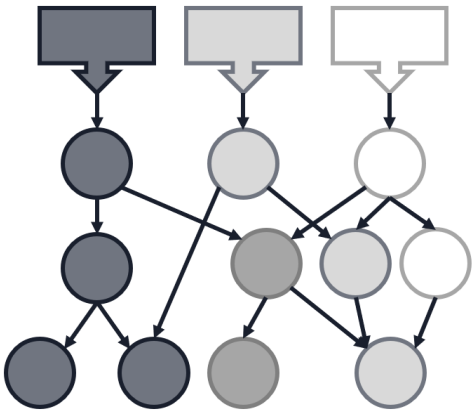


FIGURE 2.30 – Exemple d'un graphe motivationnel. Les motivations tout en haut, si elles sont vérifiées, propagent une valeur dans les nœuds-filles, qui en retour décide de comment propager cette valeur, jusqu'aux nœuds feuilles. Les actions les plus grisées sont celles qui ont la valeur la plus grande, soit les prochaines actions à réaliser.

Dans un BTs, il est impossible de sauter d'une branche à une autre. Cela veut dire que si une branche est semblable à une autre, mais située à un endroit différent de l'arbre, alors il faut la dupliquer. Ainsi, contrairement à ce genre de structure hiérarchique, les graphes de motivations peuvent le faire, ce qui augmente l'intelligibilité et la maintenance.

Toutefois, la difficulté est de bien établir les liens entre les nœuds et de calibrer les valeurs propagées par chaque nœud pour obtenir le comportement souhaité.

2.3.4 Mini-bilan

Lors de cette section, nous avons présenté plusieurs méthodes de sélection d'action. Notre objectif est de trouver une méthode pour intégrer des modèles cognitifs de manière intelligible et modulaire lors de cette étape. Nous précisons que nous ne cherchons par à remplacer un modèle d'agent par ces méthodes. La mise à l'échelle et la variabilité recherchée nous éloigne de ces approches, de par le coût d'authoring. Ensuite, nous ne disposons pas d'équipes dédiées à leur création/maintenance. Enfin, leur utilisation se fait davantage dans un cadre impératif. Il se déroule tel évènement, et en fonction de l'évaluation, je mène telle action.

Ces méthodes sont des moyens pour aider à générer des comportements selon des règles établies. Leur utilisation dans des domaines tels que la robotique et les jeux vidéo montrent que ces outils sont matures et efficaces pour certaines applications.

Concernant nos objectifs, les graphes motivationnels [Chiva et al., 2003] ont des propriétés intéressantes comme son intelligibilité. Nous aimerions adapter ce principe aux modèles cognitifs.

2.4 Conclusion de notre état de l'art

Nous avons commencé cet état de l'art, par l'étude des modèles cognitifs dans l'objectif de les généraliser pour les intégrer dans un modèle d'agent. Nous avons conclu par la difficulté de les généraliser pour deux raisons majeures. La première est que la littérature SHS propose une imposante diversité de modèles et de manières d'affecter le comportement. La deuxième est l'ensemble des interactions que les modèles peuvent avoir entre eux, *e.g.* les mécanismes du stress décrits par Driskell [Driskell et al., 2015]. Nous avons ainsi posé la question suivante : sans connaître les modèles cognitifs, comment les généraliser pour les combiner et les intégrer dans un modèle d'agent que nous ne connaissons pas non plus à l'avance ?

Nous avons ensuite étudié la littérature SMA afin d'étudier les manières dont ils intègrent des modèles cognitifs. Cette étude des travaux en informatique met en évidence la difficulté de généraliser l'opérationnalisation d'un modèle cognitif donné. Il n'existe pas une manière de faire. Ainsi, le même modèle OCC peut être implémenté de différentes manières selon les modèles d'agent utilisés (par exemple, un modèle BDI [Bourgais, 2018] ou un modèle à base de règles [Ochs et al., 2009]).

Ensuite, cette intégration amène généralement à revoir le modèle d'agent. Il ne s'agit plus du même, même lorsque ce dernier aurait été prévu pour s'y accommoder. Nous avons vu, par exemple, que la structure de EBDI pour incorporer des modèles cognitifs émotionnels est finalement insuffisante comme nous l'avons constaté avec ABC-EBDI.

Enfin, les modèles cognitifs peuvent affecter différents processus au sein du modèle d'agent, que ce soit pour la perception, pour la délibération, pour la mise à jour des données, pour la sélection d'action ou pour la communication. Ces impacts peuvent être de nature très différente, comme nous l'avons vu avec les travaux de Driskell [Driskell et al., 2018]. Par exemple, en cas de stress élevé, nous avons : filtrage d'informations lors de la perception ; réduction de la tendance à communiquer ; augmentation des conflits interpersonnels, *etc.* Il est donc difficile de proposer une opérationnalisation sur une seule de ces dimensions, car la manière dont nous implémentons le modèle cognitif pour une opération de perception peut affecter la manière dont il faut altérer le processus de décision. Cela est d'autant plus vrai que nous souhaitons être modulaire par rapport aux modèles cognitifs, tout en satisfaisant la sensibilité de chacun et l'intelligibilité du comportement. Nous avons vu, par exemple, qu'avec ACTIVITY-DL [Huguet et al., 2018], le calcul de priorité doit être révisé pour intégrer la nouvelle composante.

Nous ne cherchons pas à remettre en question ces approches. Nous les considérons plutôt comme des outils que nous pourrions utiliser/construire pour ajouter les modèles cognitifs nécessaires. La question est de savoir comment nous pouvons ajouter des modèles cognitifs de manière modulaire et intelligible, puis que nous puissions modifier le modèle de l'agent sans avoir à faire un travail considérable.

À partir de nos conclusions respectives lors de cet état de l'art, des questionnements de Bernon et al. [Bernon et al., 2005] sur les méta-modèles, le sentiment de Silverman sur l'impossibilité du modèle « One size fits all », voici notre conclusion :

Il ne semble pas pertinent, voire faisable, de proposer un modèle d'agent générique dans lequel nous pourrions intégrer n'importe quel modèle cognitif. Ce qui est pourtant notre souhait de départ. Une première raison est la difficulté de généraliser les modèles cognitifs : ils interagissent entre eux et couvrent un large spectre de dimensions. La deuxième est de disposer d'un modèle d'agent suffisamment générique pour intégrer n'importe quel de ces modèles en interaction : le champ des possibles, soit l'ensemble des impacts qu'un modèle cognitif peut avoir sur un modèle d'agent, est trop vaste ; qui plus est de manière intelligible et sensible. C'est pourquoi au lieu de proposer un modèle d'agent, qu'il faudra quoi qu'il en soit retravailler selon les modèles cognitifs et les comportements souhaités, nous nous orientons plus vers une approche qui facilite la fabrication d'un tel modèle, soit une approche par méta-modèle.

Chapitre 3

Contributions

Contents

Nous rappelons que notre objectif est de générer des comportements représentatifs, sensibles et intelligibles, grâce à l'intégration modulaire de modèles cognitifs, le tout de manière générique. Nous avons identifié suite à l'état de l'art plusieurs difficultés. L'une d'elles est la difficulté de généraliser les modèles cognitifs. Une autre est le manque d'un modèle d'agent générique permettant de modulariser et simplifier cette intégration, en posant le moins d'hypothèses.

Ainsi, nous proposons dans ce chapitre une première contribution : une approche par méta-modèle. Pour être générique, nous posons le moins d'hypothèses possibles sur les modèles cognitifs et le modèle d'agent, même dans le processus de décision. De plus, notre approche nous permet par ses fonctionnalités d'être modulaire pour simplifier l'intégration de modèles cognitifs.

Ensuite, en nous appuyant sur les fonctionnalités du méta-modèle, nous détaillons une seconde contribution : une stratégie de sélection d'action par des graphes d'influences et de préférences. Cette stratégie permet de considérer de manière modulaire et intelligible l'impact de modèles cognitifs lors de cette étape.

3.1 Méta-modèle OPACK : Aperçu

Le méta-modèle OPACK est un modèle de modèle d'agent pour la génération de comportements induite par des modèles cognitifs, dans un contexte de simulation de comportements humains. Nous cherchons à faciliter l'intégration de modèles cognitifs, tout en obtenant des comportements intelligibles. Comme nous le verrons, ce modèle fait le moins d'hypothèses possibles sur le modèle d'agent et sur les modèles cognitifs utilisés.

3.1.1 Principe général

Le principe d'OPACK consiste à abstraire un modèle d'agent autour de cinq concepts composables, dans l'objectif d'être modulaire et générique. Ces cinq concepts ont été identifiés comme les cinq aspects qu'un modèle cognitif peut affecter (*cf.* figure 3.1) :

- **O** : ensemble des opérations, soit des mécanismes de traitements d'informations, allant de la perception à la réalisation d'actions ;
- **P** : ensemble des percepts, soit ce qui est actuellement perceptible par les agents ;
- **A** : ensemble des actions à disposition et actuellement menées par les agents ;
- **C** : ensemble des caractéristiques des agents, pouvant représenter des variables physiologiques ou psychologiques comme le stress, ou bien des paramètres stables dans le temps comme la personnalité ou les qualifications ;

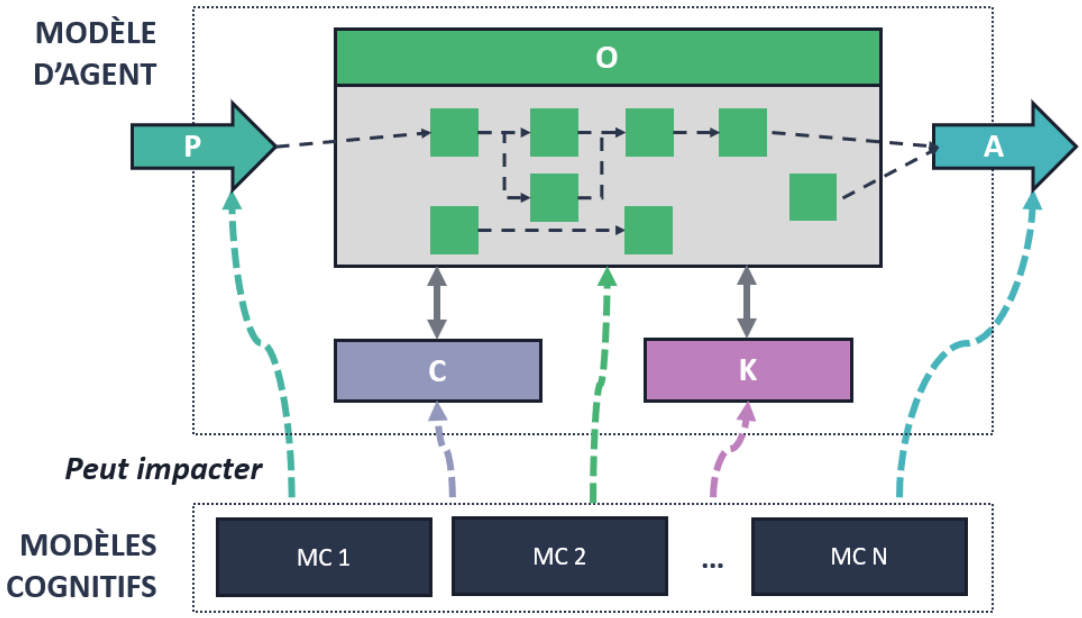


FIGURE 3.1 – Aperçu de la structure des cinq concepts d'OPACK que les modèles cognitifs peuvent impacter.

K : ensemble des connaissances des agents, qui comprend, non seulement les croyances, mais aussi les plans éventuels, *etc.*

Nous souhaitons poser le moins d'hypothèses possibles sur le modèle d'agent et les modèles cognitifs. Dans ce but, le modélisateur définit chacun de ces éléments en fonction du domaine. Il n'a pas l'obligation de tous les utiliser. Par exemple, il est possible de programmer un agent réactif, comme ceux vu lors de l'introduction des SMA avec les différents modèles d'agent [Russell and Norvig, 2021], dans lequel **K** sera vide.

3.1.2 Composabilité et généricité

La composabilité est la capacité d'un système à être modifié et recombiné en une autre structure afin de répondre à des besoins précis. L'un de nos objectifs est d'arriver à formaliser les cinq concepts d'OPACK de manière composables pour obtenir l'application voulue. D'autre part, nous souhaitons limiter le nombre d'hypothèses que nous fixons sur ces cinq concepts pour être générique.

3.1.3 Organisation

Tout d'abord, nous allons présenter un exemple qui appuiera notre discours tout au long de ce chapitre. Nous présentons ensuite le méta-modèle en deux parties.

La première partie concerne la structure du méta-modèle et la représentation des cinq concepts **O**, **P**, **A**, **C** et **K**. Nous verrons comment un modèle d'agent est représenté

de manière composable. Nous verrons aussi comment les modèles cognitifs peuvent impacter ces différents concepts. La deuxième partie détaille la dynamique du méta-modèle, soit la partie logique. C'est-à-dire que nous détaillons comment les différents concepts sont évalués et comment le modèle d'agent fonctionne à partir de sa représentation. Nous ajoutons une partie pour décrire, au travers d'un exemple, l'intégration de modèles cognitifs.

Jusque-là, nous aurons présenté le méta-modèle dans son ensemble. Nous poursuivons ensuite sur la partie suivante où nous présentons une méthode de sélection d'actions s'appuyant sur le méta-modèle, dans le but d'obtenir des comportements sensibles et intelligibles.

3.1.4 Exemple illustratif

Cette sous-section décrit un exemple illustratif que nous allons employer lors de la présentation du méta-modèle.

Nous considérons un monde virtuel composé de trois entités (*cf.* figure 3.2) : un médecin, un patient et une baignoire.

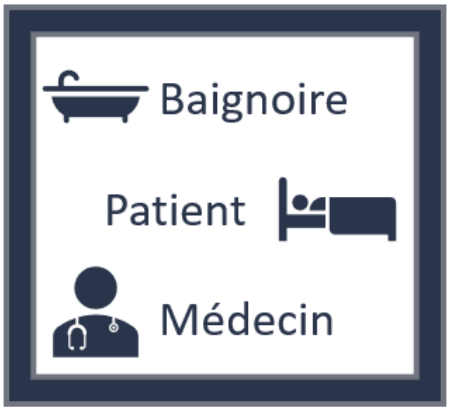


FIGURE 3.2 – Exemple illustratif des éléments composant le monde que nous allons chercher à simuler.

Le patient a des variables qui se dégradent et qui amènent le patient à tousser, si nous ne faisons rien. Le médecin peut agir pour soigner le patient. Il peut aussi utiliser la baignoire pour laver ses instruments, *etc.* Par ailleurs, le médecin peut posséder des traits de personnalité, qui vont affecter son comportement, et considérer le patient comme son ami.

3.2 Méta-modèle OPACK : Structure

3.2.1 Agents

Nous définissons un ensemble d'identifiants uniques Ω. Tous les éléments d'OPACK seront définis comme des identifiants. Nous définissons un ensemble d'agents $Agt \subset \Omega$. Les agents seront caractérisés par les éléments **O**, **P**, **A**, **C** et **K**. Les artefacts sont des agents pour lesquels **P** et **A** sont vides.

La figure 3.3 correspond à notre manière de représenter le monde de notre exemple, constitué de trois entités. Le docteur sera représenté par un agent, car c'est l'entité qui agit dans l'environnement, au sein duquel sont représentés la baignoire et le patient, sous forme d'artefacts. Ils n'agissent pas, mais peuvent posséder une certaine dynamique. Nous obtenons alors :

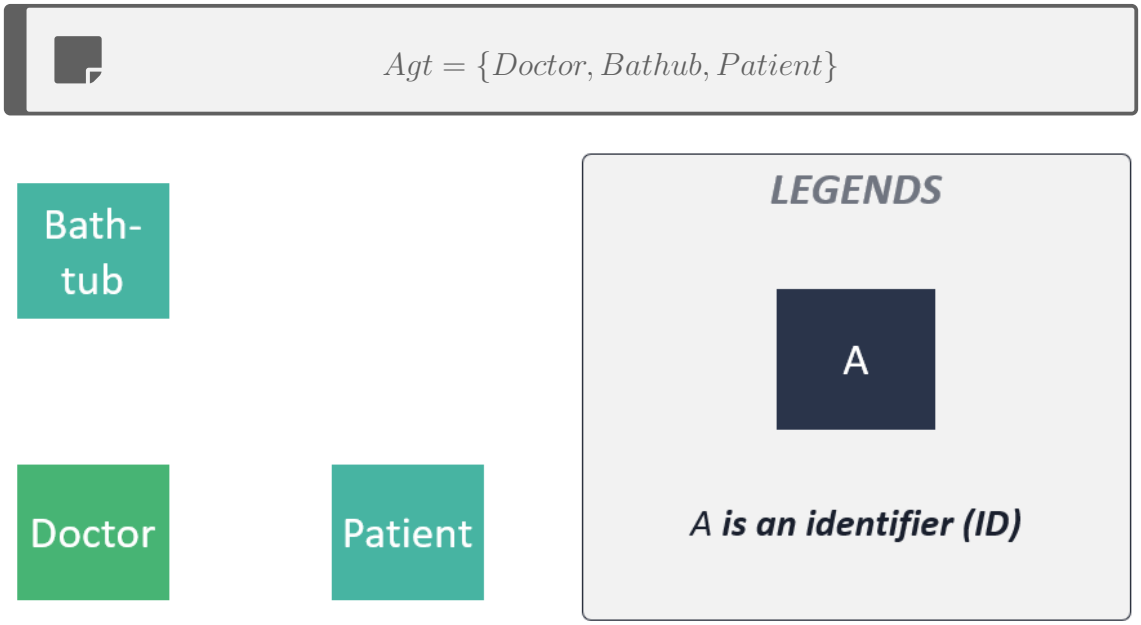


FIGURE 3.3 – Représentation de nos trois entités : $Doctor$, $Bathtub$ et $Patient$.

Désormais, nous allons décrire comment les cinq concepts **O**, **P**, **A**, **C** et **K** sont représentés. Nous distinguons deux parties : l'état de l'agent et sa dynamique. L'état d'un agent est défini par les éléments **C**, **P**, **A** et **K** et sa dynamique par **O**.

Nous commençons par décrire comment nous représentons l'état d'un agent, puis comment nous représentons la dynamique d'un modèle d'agent. L'exécution à proprement parler des modèles d'agent sera détaillée lors de la section 3.3, qui suit, où nous aborderons l'aspect dynamique du méta-modèle.

3.2.2 Caractéristique

Nous introduisons un ensemble de caractéristiques $\mathbf{C} \subset \Omega$. Les caractéristiques ne sont pas nécessairement atomiques : elles peuvent prendre en argument d'autres iden-

tifiants issus de Ω. Pour $c \in \mathbf{C}$ d'arité $k \in \mathbb{N}$ et pour $i_1, \ldots, i_k \in \Omega^k$, nous notons : $c(\cdot, i_1, \ldots, i_k)$ l'instance de la caractéristique c avec les paramètres $i_1, \ldots, i_k$.

Exemple

Considérons les ensembles suivants :

$$\begin{aligned} \Omega &= \{Arthur, Bob, Friend, Introvert\} \\ \mathbf{C} &= \{Friend, Introvert\} \\ Agt &= \{Arthur, Bob\} \end{aligned}$$

$Friend$ est une caractéristique d'arité 1, tel que : $Friend(\cdot, Bob)$ représente la propriété « être ami avec Bob ».
$Introvert$ est une caractéristique d'arité 0, tel que : $Introvert(\cdot)$ représente la propriété « être introverti ».

Pour $i \in \Omega$ et pour toute instance de caractéristique $c \in \mathbf{C}$, nous notons $c(i, \ldots)$ le fait que i est muni de la caractéristique c, avec ses paramètres.

Exemple

Considérons les ensembles suivants :

$$\begin{aligned} \Omega &= \{Arthur, Bob, Friend, Introvert\} \\ \mathbf{C} &= \{Friend, Introvert\} \\ Agt &= \{Arthur, Bob\} \end{aligned}$$

Alors voici comment traduire les éléments suivants :

1. Arthur se considère l'ami de Bob : $Friend(Arthur, Bob)$.
2. Arthur est introverti : $Introvert(Arthur)$.

Relation

Les caractéristiques d'arité supérieure ou égale à 1, *e.g.* $Friend(i, j)$ permettent de représenter des relations entre les éléments de Ω. Ainsi, nous pouvons dire que i possède une relation $Friend$ avec j.

Notations

Pour retrouver les caractéristiques, nous utilisons les notations suivantes :

- $\mathbf{C}(i), i \in \Omega$ l'ensemble des caractéristiques $c \in \mathbf{C}$ associées à l'identifiant i.
- $\forall c \in \mathbf{C}, c(\ldots, *, \ldots)$ retourne l'ensemble des identifiants i tel que $c(\ldots$,i,$\ldots)$ est vrai.

Contrainte d'unicité

Nous avons une contrainte d'unicité : uniquement une instance d'une même caractéristique peut être associée à un même identifiant. Prenons deux cas concrets :

— Dans le cas d'une caractéristique de stress d'arité 0, noté $Stress(\cdot)$. Indiquer qu'un agent $i \in Agt$ possède deux caractéristiques de stress est dénué de sens :

$$\mathbf{C}(i) = \{\ldots, Stress, Stress\} \equiv \mathbf{C}(i) = \{\ldots, Stress\}$$

Par contre, indiquer qu'un agent est caractérisé par un stress perçu et par un stress effectif a du sens. Mais, il s'agit bien de deux caractéristiques différentes :

$$\mathbf{C}(i) = \{\ldots, PerceivedStress, EffectiveStress\}$$

— Dans le cas d'une caractéristique d'arité supérieure, par exemple $Friend(\cdot, \cdot)$, indiquer qu'un agent $i \in \Omega$ est deux fois ami avec un autre agent est aussi dénué de sens :

$$Friend(i, *) = \{\ldots, j, j\} \equiv Friend(i, *) = \{\ldots, j\}, j \in \Omega$$

Par contre, indiquer qu'un agent a plusieurs amis a du sens. Mais, il s'agit bien là de deux instances différentes d'une même caractéristique.

$$Friend(i, *) = \{\ldots, j, k\}, j, k \in \Omega^2$$

Exemple

Considérons les ensembles suivants :

$$\begin{aligned} \Omega &= \{Arthur, Bob, Cyril, IsCoughing, Introvert, Friend\} \\ Agt &= \{ Arthur, Bob, Cyril \} \\ \mathbf{C} &= \{ IsCoughing, Introvert, Friend \} \end{aligned}$$

Considérons aussi que :

1. Arthur est introverti : $Introvert(Arthur)$.
2. Arthur se considère l'ami de Bob : $Friend(Arthur, Bob)$.
3. Arthur se considère l'ami de Cyril : $Friend(Arthur, Cyril)$.
4. Bob se considère l'ami de Cyril : $Friend(Bob, Cyril)$.

Alors, nous obtenons :

$$\begin{aligned} \mathbf{C}(Arthur) &= \{ Introvert, Friend \} \\ \mathbf{C}(Bob) &= \{ Friend \} \\ \mathbf{C}(Cyril) &= \{ \} \end{aligned}$$

$$\begin{aligned} Introvert(*) &= \{Arthur\} \\ Friend(Arthur, *) &= \{Bob, Cyril\} \\ Friend(Bob, *) &= \{Cyril\} \\ Friend(Cyril, *) &= \{\} \\ Friend(*, Cyril) &= \{Arthur, Bob\} \end{aligned}$$

Derrière la notion de caractéristique, il peut y avoir des structures de données complexes, au-delà du simple identifiant, comme nous le verrons dans le chapitre 4.

Caractéristique stable ou dynamique

Nous différencions **deux types de caractéristiques** :

1. **une caractéristique dynamique**, soit des variables physiologiques ou psychologiques comme le stress ;
2. **une caractéristique stable**, soit des paramètres stables dans le temps comme la personnalité ou les qualifications.

Quelle que soit sa désignation, une caractéristique ne contient aucune logique. C'est-à-dire, qu'elle n'indique pas comment elle est mise à jour. Nous différencions donc une caractéristique stable d'une caractéristique dynamique lorsqu'une opération dans **O** modifie la caractéristique en question : nous parlons alors de caractéristique dynamique, puisque sa valeur évolue. Autrement, il s'agit d'un paramètre initialisé en début de simulation : c'est une caractéristique stable.

Composabilité Les caractéristiques sont au cœur de la composabilité de notre méta-modèle. En effet, à tout moment, une caractéristique peut être modifiée, ajoutée ou supprimée d'un agent. Nous réemployons cette fonctionnalité dans nos autres concepts. Autrement dit : n'importe quelle structure de données peut être associée ou dissociée d'un agent. Par conséquent, un modèle cognitif peut aisément associer de nouvelles informations/caractéristiques à un agent, comme nous allons le voir lors de l'exemple ci-dessous.

Exemple

Reprenons notre exemple. Jusqu'ici, nous avons décrit que notre monde possédait trois entités : un agent $Doctor$ et deux artefacts, soit $Patient$ et $Bathtub$.

Supposons que nous voulons intégrer un modèle cognitif qui introduit des traits de personnalité, que nous voulons qu'une entité puisse tousser et qu'il y ait des relations d'amitié. Nous allons donc introduire trois caractéristiques, dont une est une relation :

- une caractéristique stable d'arité 0 : $Introvert$:
- une caractéristique dynamique d'arité 0 : $IsCoughing$;
- une caractéristique stable d'arité 1 : $Friend$.

$Introvert$ et $Friend$ sont des caractéristiques stables. Il s'agit donc de paramètres que le modélisateur doit attribuer. Ici, nous considérons que le médecin est introverti et ami avec le patient. $IsCoughing$ est une caractéristique dynamique. Nous considérons qu'une opération dans Ø est responsable de son ajout et de sa suppression pour notre patient. Nous obtenons ainsi la représentation de la figure 3.4 et nous avons :

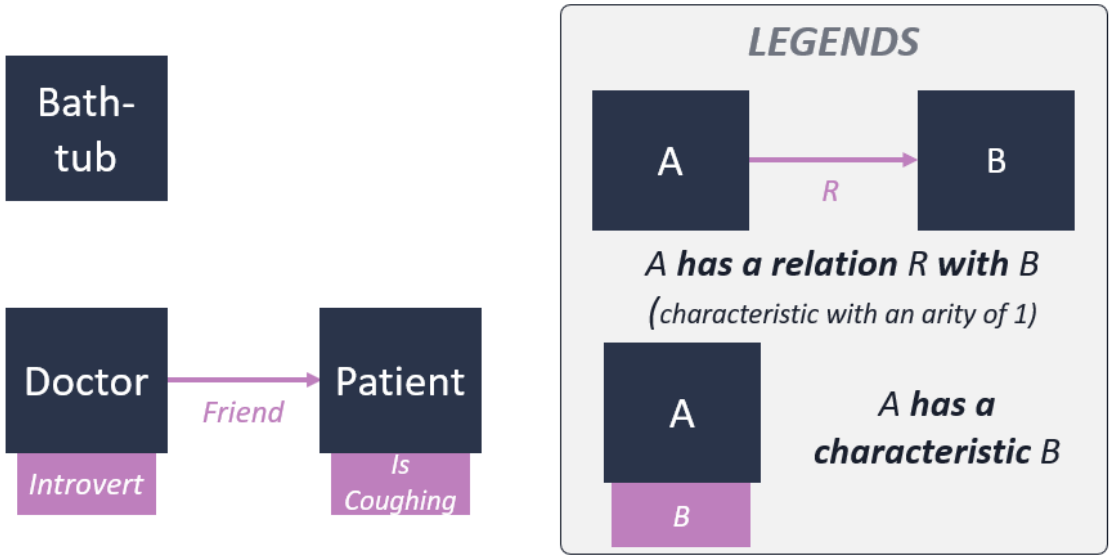


FIGURE 3.4 – Représentation des caractéristiques. Ici, nous avons rajouté trois caractéristiques : $Introvert$ à $Doctor$, $IsCoughing$ à $Patient$ et la relation $Friend$ de $Doctor$ à $Patient$. ***NOTE** : Nous ne représentons pas les identifiants des caractéristiques.*

$$
\begin{aligned}
\Omega &= \{Doctor, Patient, Bathtub, \\
&\quad Introvert, IsCoughing, Friend\} \\
\mathbf{C} &= \{IsCoughing, Introvert, Friend\} \\
\mathbf{C}(Doctor) &= \{Introvert, Friend\} \\
\mathbf{C}(Patient) &= \{IsCoughing\} \\
Friend(Doctor, *) &= \{Patient\}
\end{aligned}
$$

3.2.3 Perception

Avant de décrire la perception dans le modèle OPACK, regardons succinctement ce qu'est la perception. Selon l'APA, la perception est définie comme suit :

Citation 12 - Perception

« Le processus ou le résultat de la prise de conscience des objets, des relations et des événements au moyen des sens, qui comprend des activités telles que la reconnaissance, l'observation et la discrimination. Ces activités permettent aux organismes d'organiser et d'interpréter les stimuli reçus pour en tirer des connaissances significatives et d'agir de manière coordonnée. »

APA Dictionary of Psychology

Chez un agent virtuel, la perception est une étape du modèle d'agent, dans laquelle il peut prendre connaissance d'autres éléments de l'EV, soit sur les autres agents ou artefacts et des informations les caractérisant. **Ainsi, dans le modèle OPACK, les agents perçoivent des informations sur l'état de l'environnement aux travers de sens.**

Définition : sens

Nous notons $Sen \subset \Omega$ un ensemble de sens. Au même titre que les agents, les sens peuvent être associés à des caractéristiques. Ainsi, $\forall s \in Sen$, $\mathbf{C}(s)$ désigne l'ensemble des caractéristiques du sens s et $c(s, \ldots)$ l'instance de la caractéristique c pour le sens s.

Certaines de ces caractéristiques définissent ce que le sens **permet de percevoir**. Par exemple, si le sens $Hearing$ permet de percevoir $IsCoughing$, alors nous avons : $IsCoughing \in \mathbf{C}(Hearing)$.

Chaque agent est muni d'un ensemble de sens : $Sen(i)$ désigne l'ensemble des sens de l'agent $i \in Agt$ et $s(i)$ désigne l'instance du sens $s \in Sen$ dans l'agent i. Le modélisateur définit librement les sens, ainsi que ceux associés aux agents.

Nous introduisons une caractéristique d'arité 2 : $Perceive$. Pour un sens $s \in Sen$ donné, un agent $i \in Agt$ et un identifiant $j \in \Omega$, $Perceive(i, s, j)$ indique que l'agent i perçoit l'identifiant j avec le sens s. Comme $Perceive$ est une caractéristique, soit

elle est stable, donc le modélisateur la spécifie, soit elle est dynamique, auquel cas une opération est responsable de sa mise à jour. Nous détaillons dans la partie 3.3 comment le modélisateur définit une fonction pour déterminer ce qu'un sens peut percevoir.

Définition : percept

L'ensemble des percepts pour tous les agents est noté $\mathbf{P}$. Cet ensemble varie lors de la simulation : il représente ce qui peut être perçu à un instant t donné. On note $\mathbf{P}(i)$ l'ensemble des percepts perçus par l'agent $i \in Agt$. $\mathbf{P}(i)$ est calculé de la manière suivante :

1. Pour un sens s d'un agent i et pour un identifiant j perçu, nous calculons l'ensemble des instances de caractéristiques perçues de cet identifiant j via le sens s :

$$apc(i, s, j) = \bigcup_{c \in \mathbf{C}(j) \cap \mathbf{C}(s(i))} c(j, *, \ldots, *)$$

   Par exemple, si $Patient$ et $Hearing$ ont tous deux la caractéristique $isCoughing$, alors $apc(Doctor, Hearing, Patient) = \{IsCoughing(Patient)\}$.

2. Nous calculons ensuite l'ensemble des caractéristiques perçues pour tous les identifiants perçus via le sens s :

$$cap(s, i) = \bigcup_{j \text{ tq } Perceive(i,s,j)} apc(i, s, j)$$

   Ainsi, s'il y a plusieurs patients qui toussent, cap calcule les instances pour chaque patient perçu par le sens $Hearing$ du médecin :

$$\{isCoughing(Patient_1), isCoughing(Patient_2), \ldots\}$$

3. Enfin, nous appliquons cette opération à l'ensemble des sens de l'agent i, ce qui nous donne :

$$\mathbf{P}(i) = \bigcup_{s \in Sen(i)} cap(i, s)$$

Concrètement, $\mathbf{P}(i)$ est l'intersection des caractéristiques des agents perçus et des caractéristiques des sens de perception.

Composabilité Comme pour les caractéristiques, cette représentation est composable : nous ne fixons pas à l'avance les sens, ni leurs capacités perceptives. Le modélisateur peut construire sans restriction les sens et les informations perceptibles. Les agents peuvent posséder autant de sens que nécessaire. Ainsi, nous ne présupposons pas les capacités perceptives des agents.

Dès lors, à tout moment un sens peut être ajouté à un agent, ou bien un sens existant peut être modifié en complétant ses capacités et ses conditions de perceptibilité. Toutefois, cela relève plus d'un modèle physiologique ; les modèles cognitifs seront plus à même d'affecter la perception lors de la cognition, soit lors des opérations comme nous le verrons par la suite.

Exemple

Reprenons notre exemple. Supposons que nous voulons que notre agent perçoive que le patient tousse. Nous allons introduire un nouveau sens $Hearing$ et indiquer que ce sens permet de percevoir $IsCoughing$. Nous avons donc :

$$\begin{aligned} \Omega &= \{\ldots, Hearing\} \\ \mathbf{C}(Doctor) &= \{Introvert\} \\ \mathbf{C}(Patient) &= \{IsCoughing\} \\ \mathbf{C}(Hearing) &= \{IsCoughing\} \\ Sen(Doctor) &= \{Hearing\} \end{aligned}$$

Dans cet exemple, nous supposons aussi que notre agent perçoit auditivement toutes les entités présentes. Nous obtenons la représentation suivante 3.5 et la notation suivante :

$$\mathbf{C}(Doctor) = \{\, Perceive(\cdot, Hearing, Patient), Perceive(\cdot, Hearing, Bathtub) \,\}$$

Nous pouvons calculer :

$$\begin{aligned} \mathbf{P}(Doctor) &= cap(Doctor, Hearing) \\ &= apc(Doctor, Hearing, Patient) \cup apc(Doctor, Hearing, Bathtub) \\ &= \{\, IsCoughing(Patient) \,\} \end{aligned}$$

car $IsCoughing$ est présent à la fois dans $Hearing$ et dans $Patient$.

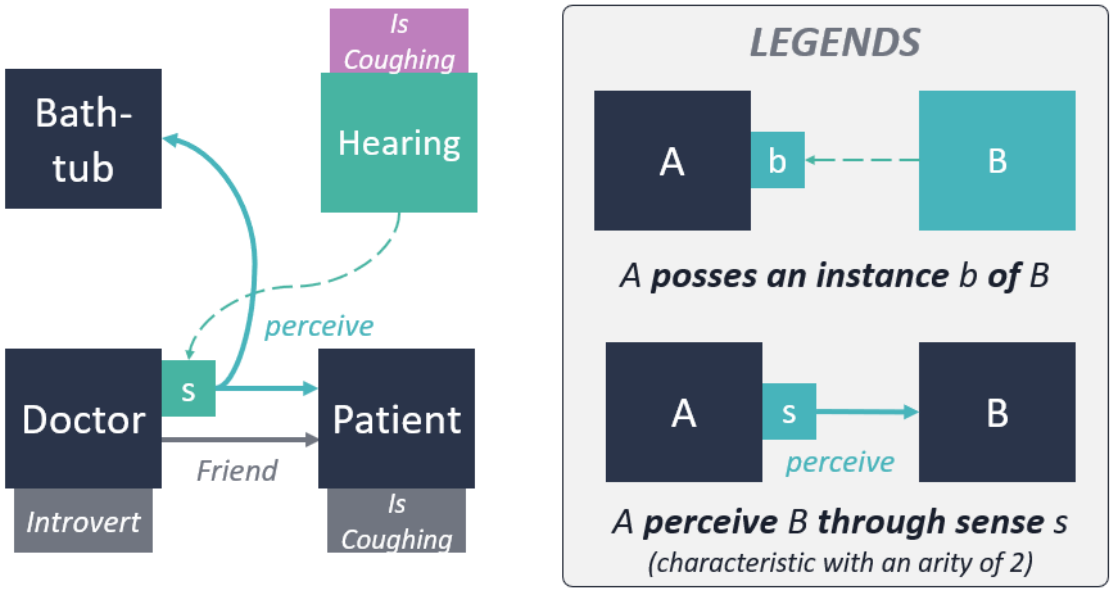


FIGURE 3.5 – Représentation des mécanismes de perception. Nous ajoutons un sens $Hearing$ et une instance de ce sens à notre agent. Puisque l'agent entend le patient, il perçoit qu'il tousse, sans avoir à explicitement indiquer qu'il perçoit qu'il tousse. La flèche bleue est la relation $Perceive$ d'un sens vers l'entité qu'il perçoit. La flèche noire et le carré bleu associé représentent le fait que $Doctor$ possède une instance (carré bleu) du sens $Hearing$.

3.2.4 Action

Une action est le fait de réaliser une intention sur un ou des éléments de l'environnement, *e.g* se déplacer, prendre un objet, activer un levier, *etc.* **Dans le modèle OPACK, un agent réalise des actions sur son environnement aux travers d'effecteurs.**

Notation : effecteur

Nous notons $Eff \subset \Omega$ un ensemble d'effecteurs. Les effecteurs peuvent aussi être associés à des caractéristiques : $\forall e \in Eff$, $\mathbf{C}(e)$ désigne l'ensemble des caractéristiques de l'effecteur e et $c(e)$ l'instance de la caractéristique c pour l'effecteur e.

Chaque agent est muni d'un ensemble d'effecteurs : $Eff(i)$ désigne l'ensemble des effecteurs de l'agent $i \in Agt$ et $e(i)$ désigne l'instance de l'effecteur $e \in Eff$ de l'agent i. Le modélisateur définit librement les effecteurs, ainsi que ceux associés aux agents.

Notation : action

Nous notons $\mathbf{A} \subset \Omega$ un ensemble d'actions. Nous pouvons encore une fois associer des caractéristiques aux actions : $\forall a \in \mathbf{A}$, $\mathbf{C}(a)$ désigne l'ensemble des caractéristiques de l'action a et $c(a)$ l'instance de la caractéristique c pour l'action a.

L'ensemble **A** désigne l'ensemble des actions à dispositions. Rien n'empêche de créer de nouvelles actions ou de les modifier, en les composant différemment. En effet, du fait que les actions soient des identifiants, le modélisateur peut librement associer des caractéristiques aux actions, soit des paramètres pour définir l'arité, le pourcentage de succès, *etc.*

Relation $Requires$ Chaque action doit nécessairement spécifier au minimum un effecteur requis pour son exécution. Par exemple, l'action « attraper » pourrait nécessiter un effecteur nommé « main ». Dès lors, nous introduisons une caractéristique d'arité 1 : $Requires$. Autrement dit : $\forall a \in \mathbf{A}, \exists e \in Eff, Requires(a, e)$ indique qu'une action a nécessite un effecteur e. C'est au modélisateur d'indiquer cette relation.

Relation $Doing$ Pour indiquer qu'un effecteur réalise une action, nous introduisons une nouvelle caractéristique d'arité 2 : $Doing$. Pour un effecteur $e \in Eff$ donné, un agent $i \in Agt$ et une action $a \in \mathbf{A}$, $Doing(i, e, a)$ indique que l'agent i effectue l'action a par le biais de l'effecteur e. Nous verrons lors de la section 3.3 comme le modélisateur la spécifie lors d'une opération.

Ainsi, nous notons $\mathbf{A}(i)$ l'ensemble des actions menées par l'agent $i \in Agt$, calculé de la manière suivante :

$$\mathbf{A}(i) = Doing(i, *, *)$$

Pour $e \in Eff, a \in AGTeta \in \mathbf{A}, Doing(i, e, a)$ indique que l'agent i effectue l'action a par le biais de l'effecteur e. Mais, cela définit aussi une instance de a : l'instance effectuée par i via e.

Ainsi $Doing(i, e, a)$ est non seulement l'instance de $Doing$ pour i, e et a (comme nous l'avons vu pour n'importe quelle caractéristique) mais c'est aussi l'instance de a effectuée par i via e.

Status d'une action Ensuite, nous considérons que les actions peuvent durer dans le temps et sont interruptibles. C'est-à-dire qu'à tout moment, malgré le fait qu'une action soit en cours de réalisation par un effecteur, une autre action peut être menée. Concrètement, Une interruption intervient lorsque la relation $Doing$ est remplacée par une autre. L'action en cours est alors annulée et remplacée par une autre. La relation $Doing$ possède alors une variable $Status$ pour indiquer le statut de l'action. Les valeurs possibles sont :

$$Status : \{Waiting, Starting, Running, Finished, Aborted\}$$

Nous verrons lors de la partie 3.3 comment le méta-modèle évalue l'état en cours. Entre-temps, voici la signification de chaque valeur :

— **Waiting** : état par défaut suite à l'instanciation de l'action.

— **Starting** : l'action vient juste de commencer.

— **Running** : l'action est en cours d'exécution.

— **Finished** : l'action s'est terminée sans interruption.

— **Aborted** : l'action s'est terminée suite à une interruption.

Contrainte d'unicité Enfin, nous imposons une contrainte : un effecteur ne peut réaliser qu'une action à la fois. Autrement dit, la relation $Doing$ est unique par effecteur.

Composabilité À tout moment, de nouvelles actions ou effecteurs peuvent être définis. Rien n'empêche aussi de modifier les actions ou les effecteurs après coup. Par exemple, prenons les ALU du modèle de Fadier [Fadier et al., 2003]. Nous pourrions ajouter d'autres actions correspondant aux activités limites tolérés par l'usage ou alors spécifier que certaines des actions déjà présentes peuvent en être. La difficulté désormais est que ces actions puissent se retrouver dans le comportement de l'agent lorsqu'il doit réaliser des ALUs. Nous verrons cela plus tard lors d'un exemple plus complet, que nous présentons après la description du méta-modèle.

Exemple

Reprenons notre exemple. La dernière fois, nous avions ajouté les perceptions. Désormais, nous allons ajouter le concept d'action. Supposons que nous voulons que notre agent réalise l'action : « se laver les mains ».

Tout d'abord, nous allons introduire un nouvel effecteur : $Hands$. Ensuite, nous allons définir une action $Wash$, requérant l'effecteur $Hands$. Nous introduisons aussi une nouvelle relation $With(a, i)$ pour indiquer que l'action a se réalise *avec* l'identifiant i.

Nous supposons que le médecin se lave les mains avec la baignoire : $Doing(Doctor, Hands, a)$, avec a une instance de l'action $Wash$. Ainsi, si nous souhaitons représenter cet état, alors nous obtiendrons la figure 3.6, ainsi que les ensembles suivants :

$$
\begin{aligned}
\Omega &= \{\ldots, Hands, With, Doing\} \\
\mathbf{C}(Doctor) &= \{Introvert\} \\
\mathbf{C}(Patient) &= \{IsCoughing\} \\
Eff(Doctor) &= \{Hands\} \\
A(i) &= \{Doing(Doctor, Hands, Wash\} \\
\mathbf{C}(a) &= \{With\} \\
With(a, *) &= \{Bathtub\}
\end{aligned}
$$

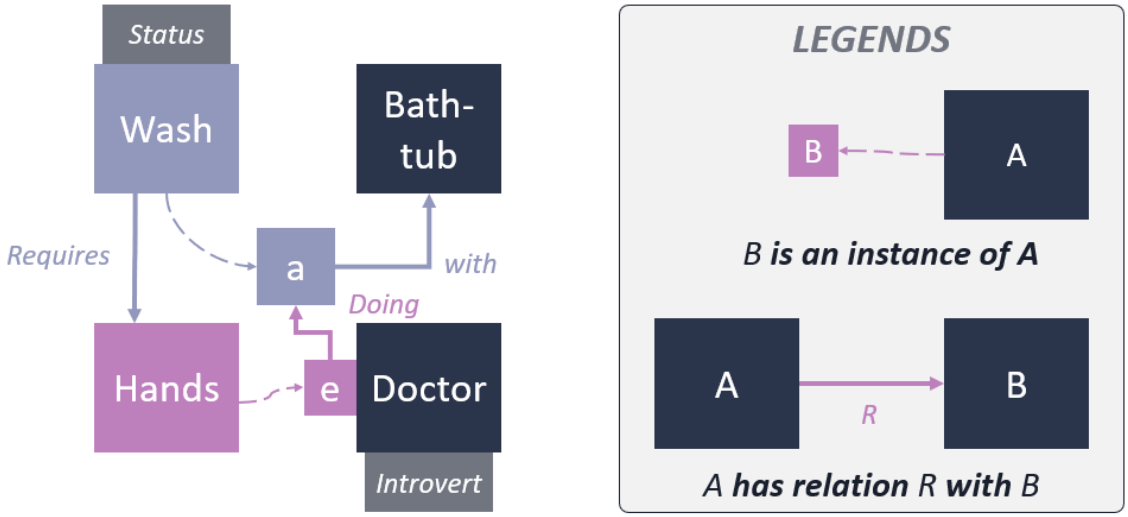


FIGURE 3.6 – Représentation des mécanismes d'action. Nous avons ajouté un effecteur $Hands$ et une instance de cet effecteur à notre agent. Ensuite, nous avons défini une action $Wash$ qui nécessite l'effecteur $Hands$. L'agent peut entreprendre cette action au travers d'une instance qui a pour paramètre l'artefact avec lequel il se lave les mains, soit la baignoire ici.

3.2.5 Knowledge (connaissances)

Nous notons $\mathbf{K} = \Omega \cup \mathbf{C}$ l'ensemble des connaissances, soit la vérité du monde simulé, car $\Omega \cup \mathbf{C}$ représente l'état du monde, soit l'ensemble des identifiants et leurs caractéristiques. Nous disons aussi que $\mathbf{K}$ est une base de connaissances. Autrement dit, pouvoir vérifier que l'agent i est dans Agt, est une connaissance au même titre que de pouvoir récupérer ses amis comme nous avions vu précédemment, *e.g* $Friend(Bob, *)$.

Toutefois, nous n'imposons aucune contrainte sur la manière de représenter les connaissances pour un agent. Le modélisateur est libre d'utiliser différents outils pour les représenter, comme des ontologies ou des bases de données relationnelles [Martínez-Cruz et al., 2011].

Par conséquent, le modélisateur peut réutiliser notre représentation, pour créer une base de connaissances partielles et potentiellement erronées, telle que : $\mathbf{K}' = \Omega' \cup \mathbf{C}'$ avec $\mathbf{K}(i)' \notin \mathbf{K}, i \in Agt$. Autrement dit, $\mathbf{K}'$ est une représentation du monde, du point de vue d'un agent, qui doit être alimenté par les opérations.

Une autre manière de réutiliser notre représentation et d'incorporer de nouveaux identifiants et de nouvelles caractéristiques. Par exemple, nous avons intégré le formalisme ACTIVITY-DL dans notre formalisme, pour que nos agents puissent raisonner sur un arbre d'activité. Les tâches ont été représentées avec des identifiants et l'arborescence au travers de relation. Ensuite, nous pouvions soit partager la racine de l'arbre entre différents agents pour qu'ils puissent raisonner sur la même instance d'un arbre, soit nous instancions l'arbre pour chaque agent. Puisque les tâches étaient des identifiants, elles sont aussi composables que le reste.

3.2.6 Opération

Les actions menées par un agent sont le produit de son modèle d'agent. Autrement dit, sa dynamique est dictée par un ensemble de mécanismes de traitements d'informations, allant de la perception à la réalisation d'actions, suite à une prise de décision, en passant par la communication avec d'autres agents. Cette dynamique met en œuvre les différentes dimensions pour intégrer des processus cognitifs ou physiologiques. **Dans le méta-modèle OPACK, O représente la dynamique d'un modèle d'agent aux travers d'un ensemble d'opérations.**

Notation

Nous notons $\mathbf{O} \subset \Omega$ un ensemble d'opérations. Comme pour le reste des concepts, les opérations peuvent aussi être associées à des caractéristiques. Ainsi, $\forall o \in \mathbf{O}$, $\mathbf{C}(o)$ désigne l'ensemble des caractéristiques de l'opération o et $c(o, \ldots)$ l'instance de la caractéristique c pour l'opération o.

Une opération $o \in \mathbf{O}$ est une fonction avec des entrées, des sorties et un rôle. Pour spécifier les entrées et sorties, nous introduisons deux nouvelles caractéristiques :

- une caractéristique d'arité 2 : $\mathbf{Output}(o, n, i), n, i \in \Omega^2$: indique que l'opération o possède une sortie, identifiée par n, de type i.
- une caractéristique d'arité 2 : $\mathbf{Input}(o', o, n)$: indique que l'opération $o' \in \mathbf{O}$ a en entrée, la sortie n de l'opération o.

Ainsi, $Input(o, *, *)$ retourne l'ensemble des entrées de l'opération o et $output(o, *, *)$ retourne l'ensemble de ses sorties. Quoi qu'il en soit, toutes les opérations ont accès à l'ensemble de l'état de l'agent : P, A, C et K.

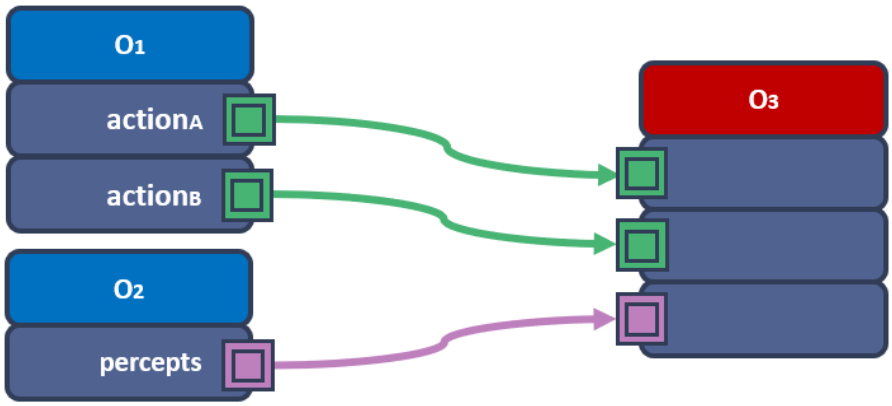


FIGURE 3.7 – Exemple de la représentation de trois opérations et de leurs entrées/sorties.

Exemple

Considérons les ensembles suivants :

$$\begin{aligned} \Omega &= \{\ldots, o_1, o_2, o_3 Action, Percepts\} \\ \mathbf{O} &= \{o_1, o_2\} \end{aligned}$$

Alors, si nous voulons représenter les entrées et sorties de la figure 3.7, nous pourrions les spécifier de la manière suivante :

1. Sorties de o_1 :
   - Une sortie identifiée par $action_A$, de type $Action$:

     $$Output(o_1, action_A, Action)$$

   - Une sortie identifiée par $action_B$, de type $Action$:

     $$Output(o_1, action_B, Action)$$

2. La sortie de l'opération o_2 , identifiée par $percepts$, de type $Percepts$:

   $$Output(o_2, percepts, Percepts)$$

3. Entrées de o_3 :
   - Une entrée correspondant à la sortie $action_A$ de o_1 :

     $$Input(o_3, o_1, action_A)$$

   - Une entrée correspondant à la sortie $action_B$ de o_1 :

     $$Input(o_3, o_1, action_B)$$

   - Une entrée correspondant à la sortie $percepts$ de o_2 :

     $$Input(o_3, o_2, percepts)$$

Et nous obtiendrons les ensembles suivants :

$$\begin{aligned} Output(o_1, *, *) &= \{\{action_A, Action\}, \{action_B, Action\}\} \\ Output(o_2, *, *) &= \{\{percepts, Percepts\}\} \\ Input(o_3, *, *) &= \{\{action_A, o_1\}, \{action_B, o_1\}, \{percepts, o_2\}\} \end{aligned}$$

Notion de rôle

Le *rôle* de l'opération cadre sa signature (soit ses entrées et sorties). Les noms des rôles proviennent des fonctions mentales décrites par l'APA [1] :

- **Manipulation** : aucune entrée et sortie (opère uniquement à partir de l'état de l'agent) ;

$$|Input(o, *, *)| = 0 \text{ et } |Output(o, *, *)| = 0$$

- **Acquisition** : aucune entrée, mais au moins une sortie ;

$$|Input(o, *, *)| = 0 \text{ et } |Output(o, *, *)| > 0$$

- **Storage** : des entrées, mais aucune sortie (agit sur l'état de l'agent) ;

$$|Input(o, *, *)| > 0 \text{ et } |Output(o, *, *)| = 0$$

- **Interpretation** : des entrées et des sorties ;

$$|Input(o, *, *)| > 0 \text{ et } |Output(o, *, *)| > 0$$

- **Transformation** : cas particulier d'interprétation où une des entrées est de même type que la sortie : c'est notre donnée manipulée. Les autres entrées, ainsi que l'état, permettent de contrôler la modification apportée à la donnée manipulée par l'opération.

La notion de rôle nous sera utile plus tard lorsque nous présenterons d'autres concepts liés aux opérations. Au-delà, l'idée était aussi de représenter les différentes opérations mentales identifiées par l'APA.

Exemple : adaptation du modèle d'agent Replicants

Pour illustrer le concept des opérations, nous allons adapter le modèle d'agent Replicants avec notre méta-modèle. Nous identifions deux opérations :

1. Une opération d'interprétation : la sélection d'action (sortie) à partir d'actions candidates (entrée).
2. Une opération d'acquisition : déterminer les actions envisageables (sortie). Dans Replicants, cela consiste à retourner les actions candidates suite au parcours de l'arbre selon l'état de la simulation. Cette opération ne reçoit pas d'entrée, mais utilise l'état de l'agent (dont l'arbre Activity-DL qui est dans **K** et les caractéristiques de l'agent **C**) pour calculer un ensemble d'actions.

1. https://dictionary.apa.org/cognitive-process

Ces deux opérations sont illustrées sur la figure 3.8. Comme on le voit sur cette figure, la sortie d'une opération est reliée aux entrées d'autres opérations : nous parlons de *flux d'opérations*.

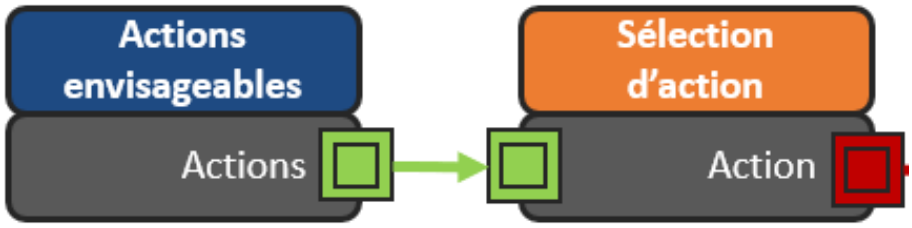


FIGURE 3.8 – Exemples de deux opérations pour implémenter le modèle d'agent REPLICANTS.

Flux d'opérations

Nous notons $Flo \in \Omega$ l'ensemble des flux d'opérations. Comme pour le reste des concepts, les flux d'opérations peuvent aussi être associés à des caractéristiques. Ainsi, $\forall f \in Flo$, $\mathbf{C}(f)$ désigne l'ensemble des caractéristiques du flux f et $c(f, \ldots)$ l'instance de la caractéristique c pour le flux f.

Ajout d'opérations dans un flux Le modélisateur définit les opérations et les relie en un flux d'opérations, à la manière de la programmation par flux de Morrison [Morrison, 1994]. Concrètement, un flux d'opérations est un graphe orienté acyclique dont les sommets sont des opérations et dont les arcs représentent les flux de données entre la sortie d'une opération et les entrées des suivantes, comme la figure 3.9 l'illustre. Nous introduisons la relation $Operation$. $Operation(o, f)$ indique qu'une opération $o \in \mathbf{O}$ fait partie du flux d'opérations $f \in Flo$.

Exemple

Considérons les ensembles suivants :

$$\begin{aligned} \Omega &= \{\ldots, o_1, o_2\} \\ Flo &= \{Replicants\} \\ \mathbf{O} &= \{o_1, o_2\} \end{aligned}$$

Alors, nous pouvons ajouter les opérations o_1 et o_2 de la manière suivante :

$$\begin{aligned} &Operation(o_1, Replicants) \\ &Operation(o_2, Replicants) \end{aligned}$$

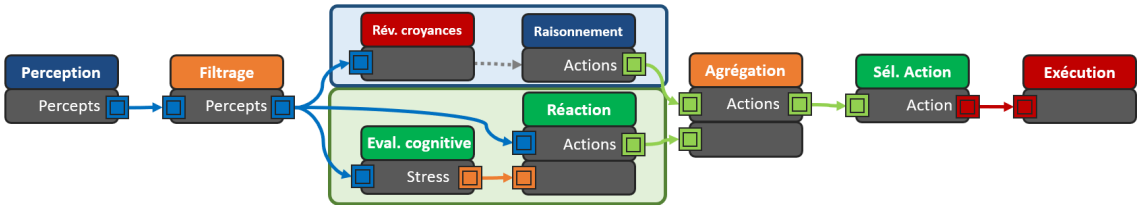


FIGURE 3.9 – Exemple d'un flux d'opérations arbitraire illustratif. Un flux d'opérations est un graphe acyclique. Les couleurs des étiquettes des sommets correspondent aux rôles des opérations, et les couleurs des arcs aux types des données échangées. Une sortie ne peut être reliée qu'à des entrées de même type. Les arcs de précédence sont dessinés en gris pointillé.

Ordre Il n'est pas nécessaire d'indiquer l'ordre des opérations. Le méta-modèle s'occupera d'ordonnancer les opérations selon les entrées et sorties des opérations composant le flux.

Relation de précédence En complément, nous introduisons la relation $Before$: $Before(o_1, o_2)$ indique que l'opération $o_1 \in \mathbf{O}$ doit avoir lieu avant l'opération $o_2 \in \mathbf{O}$. Ainsi, le modélisateur peut définir des arcs de précédence entre deux opérations, qui ne correspondent pas à un échange de données, mais à un ordre d'exécution.

La figure 3.9 donne un exemple de flux. Les couleurs des étiquettes des sommets correspondent aux rôles des opérations, et les couleurs des arcs aux types des données échangées. Naturellement, une sortie ne peut être reliée qu'à des entrées de même type. Les arcs de précédence sont dessinés en gris pointillé.

Composition d'un flux Un flux est composable de plusieurs manières :

- **Structure du graphe composable** : Le modélisateur définit des opérations et les liens entre-elles, soit par des liens de précédence, soit par les entrées et sorties. Ces informations suffisent à déterminer la structure du graphe. À tout moment, une nouvelle opération peut être ajoutée au graphe.
- **Structure des opérations composables** : À tout moment, des entrées ou sorties peuvent être ajoutées aux opérations, pour accommoder l'ajout de nouvelles opérations par exemple.

Association d'un flux à un agent Les flux sont des identifiants. Dès lors, nous pouvons ajouter des flux comme caractéristique d'un agent.

Nous notons $Flo(i)$, l'ensemble des flux associés à l'agent $i \in Agt$. Autrement dit, la dynamique d'un modèle d'agent est dicté uniquement par les flux d'opérations que l'agent possède aux travers de ses caractéristiques.

Plusieurs propriétés intéressantes en découlent donc :

1. **Multiplicité :** rien n'empêche à un agent de posséder plusieurs flux d'opérations. Par exemple, nous pourrions tout à fait ajouter un flux responsable des expressions faciales, décorrélé d'un autre flux responsable de la prise de décision.
2. **Dynamicité :** puisqu'il s'agit d'une caractéristique, alors rien n'empêche qu'il s'agisse d'une caractéristique dynamique. Autrement dit, les flux peuvent être ajoutés, supprimés et modifiés dynamiquement.

La raison derrière ce choix est de chercher à minimiser les hypothèses sur la manière de notre agent à raisonner. Qui plus est, nous supposons que cette liberté offre plus de possibilités aux modèles cognitifs pour affecter le modèle d'agent. Par exemple, la dynamicité des flux d'opérations permet de définir une cognition qui s'adapte à la situation.

Exemple Reprenons notre exemple pour introduire le concept **O**. Supposons que notre agent veuille raisonner grâce au flux d'opérations REPLICANTS que nous avons construit précédemment. Nous obtiendrons la figure 3.10 et les ensembles suivants :

$$
\begin{aligned}
\Omega &= \{\ldots, Doctor, Replicants, SuitableActions, ActionSelection, \ldots\} \\
\mathbf{O} &= \{SuitableActions, ActionSelection\} \\
Flo &= \{Replicants\} \\
\mathbf{C}(Doctor) &= \{Introvert, Replicants\} \\
Flo(Doctor) &= \{Replicants\} \\
Operation(Replicants) &= \{SuitableActions, ActionSelection\}
\end{aligned}
$$

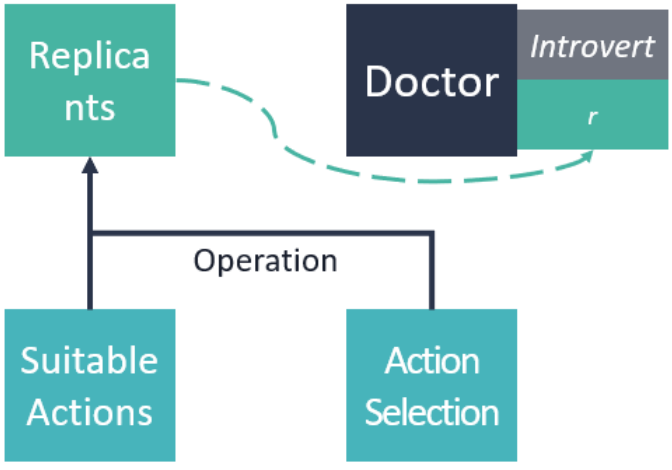


FIGURE 3.10 – Représentation des mécanismes d'opération. Ici, nous avons rajouté un flux d'opération *Replicants*. L'agent possède ce flux comme caractéristique, ce qui indique qu'il doit raisonner selon ce flux. Ensuite, nous avons défini deux opérations : *SuitableActions* qui fournit les actions pour *ActionSelection*.

Impact d'un modèle cognitif Supposons que pour cet exemple, nous voulons ajouter un modèle cognitif de stress simpliste pour que notre agent puisse stresser. Nous allons donc définir une nouvelle caractéristique de stress. Nous allons ajouter une opération pour mettre à jour le stress, la rendant ainsi dynamique. Nous obtenons ainsi la figure 3.11 et les ensembles suivants :

$$
\begin{aligned}
\Omega &= \{\ldots, Doctor, Replicants, \\
&\quad SuitableActions, ActionSelection, UpdateStress, \ldots\} \\
\mathbf{O} &= \{SuitableActions, ActionSelection, UpdateStress\} \\
Flo &= \{Replicants\} \\
\mathbf{C}(Doctor) &= \{Introvert, Replicants, Stress\} \\
Flo(Doctor) &= \{Replicants\} \\
Operation(Replicants) &= \{SuitableActions, ActionSelection, UpdateStress\} \\
&\ldots \\
Inputs(UpdateStress) &= \{\} \\
Outputs(UpdateStress) &= \{\}
\end{aligned}
$$

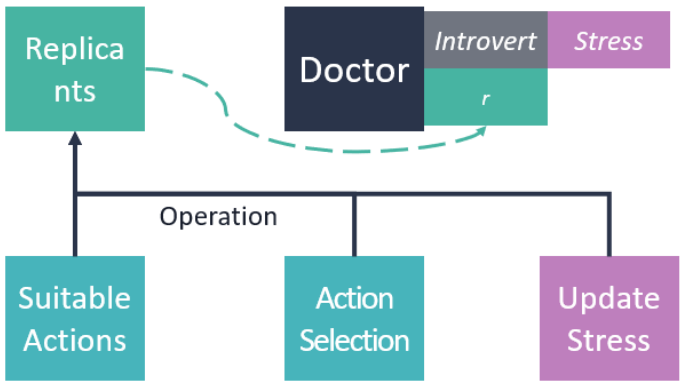


FIGURE 3.11 – Nous avons ajouté un modèle cognitif de stress en ajoutant une caractéristique $Stress$ au docteur et une opération dans le flux pour la rendre dynamique. Nous n'avons pas eu besoin de modifier quoi que ce soit d'autre, grâce à la composabilité.

Mini-bilan

Jusqu'ici, nous avons vu comment nos cinq concepts étaient composables. Cette composabilité nous permet d'être modulaire et générique. Nous avons illustré au travers d'un exemple simple comment cela fonctionnait et comment des modèles cognitifs pouvaient affecter à chaque fois les différents concepts.

Toutefois, il manque un aspect au niveau des opérations pour que le tout soit fonctionnel. Plus précisément, nous avons vu comment un flux d'opérations était composable : ajout d'opérations organisées automatiquement sous forme de graphe acyclique. Nous l'avons aussi vu pour la structure des opérations : ajout modulaire d'entrées et de sorties.

Cependant, à aucun moment, nous avons indiqué comment les opérations fonctionnaient et opéraient sur les entrées et sorties. La difficulté est donc de savoir comment le faire de manière composable (*cf.* figure 3.12). Pour y répondre, nous introduisons le concept de *stratégie* et d'*impacts*.

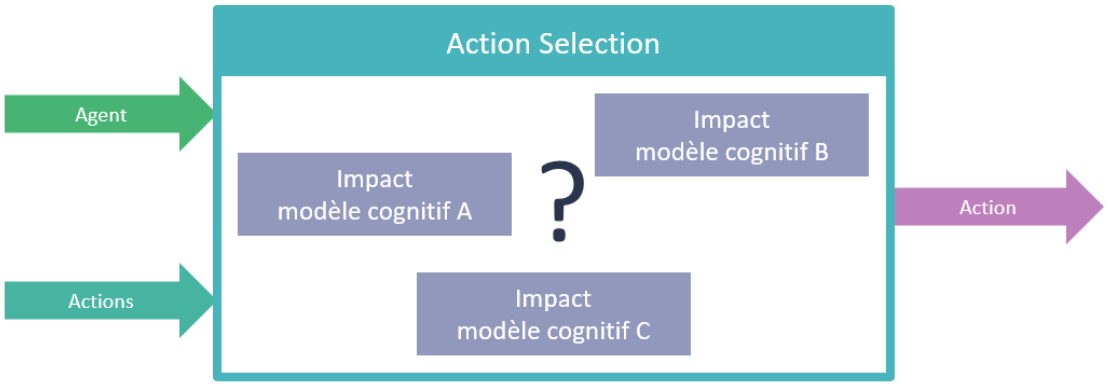


FIGURE 3.12 – Comment modifier modulairement et génériquement le fonctionnement d'une opération, comme l'opération de sélection d'actions par exemple ? Autrement dit, comment rendre le fonctionnement d'une opération composable ?

Comportement, stratégie et impact

Chaque opération doit définir une *stratégie* pour calculer le résultat de l'opération à partir d'un ensemble d'*impact*. Les modèles cognitifs ajoutent des impacts aux opérations aux travers de *comportements*.

Notation comportement Nous notons $Beh \in \Omega$ l'ensemble des comportements. $\forall b \in Beh, \mathbf{C}(b)$ retourne l'ensemble des caractéristiques du comportement b et $c(b, \ldots)$ l'instance de la caractéristique c pour b.

Fonction-impact Nous notons introduisons une caractéristique d'arité 1 : $Impact(b, o), o \in \mathbf{O}$, qui indique que le comportement b possède une fonction-impact pour l'opération o. $Impact(b, *)$ retourne l'ensemble des fonctions-impacts du comportement b. Un comportement ne peut avoir qu'une seule fonction-impact par opération.

> Rien n'empêche de transformer $Impact$ en une caractéristique d'arité 2 pour identifier la fonction-impact et permettre plusieurs fonctions-impacts d'un même comportement vers une même opération.

$Impact(*, o)$ retourne l'ensemble des comportements ayant une fonction-impact pour o. Nous verrons les entrées et sorties des fonctions-impacts, après avoir introduit la notion de stratégie.

Stratégie Chaque opération doit définir une et une seule *fonction-stratégie* notée s_o. C'est une fonction avec en entrées : l'état de l'agent, les entrées de l'opération et l'ensemble $Impact(*, o)$ potentiellement vide. La fonction-stratégie s_o calcule la sortie de o à partir des fonctions-impacts $f \in I_o$ (éventuellement vide). Donc, la fonction-stratégie impose le type de sortie attendu par les fonctions-impacts, noté $\tau(s_o)$.

Ainsi, toutes les fonctions-impacts $f \in Impact(*, o)$ ont les mêmes entrées que l'opération o et les mêmes sorties entre eux, notées $\tau(s_o)$. Ces sorties peuvent être différentes du type de sortie de l'opération o elle-même.

Autrement dit, **une fonction-stratégie calcule le résultat de l'opération**, à partir des entrées de l'opération et des résultats des fonctions-impacts. La figure 3.13 illustre une stratégie que nous proposons pour la sélection d'action où chaque fonction-impact indique quelles actions il aimerait voir ou non se réaliser. Il s'agit de la stratégie de choix par graphes d'influences et de préférences que nous détaillons dans la section 3.5.

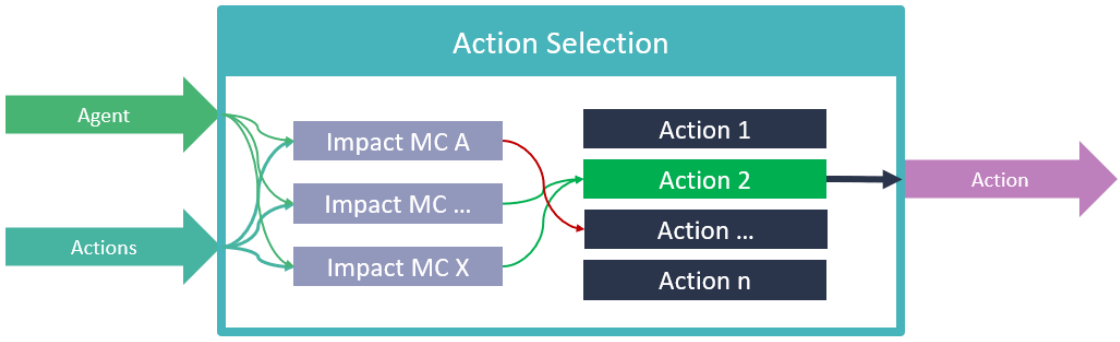


FIGURE 3.13 – Exemple d'une stratégie pour la sélection d'action que nous proposons : stratégie de choix par graphes d'influences et de préférences. Les fonctions-impacts indiquent quelles actions seraient préférables ou non selon lui. L'action la plus préférée est sélectionnée.

Exemple de comportements Il est possible de définir plusieurs fonctions-impacts pour un comportement b donné, mais pour des opérations différentes. Par exemple, pour intégrer le modèle cognitif de Demary [Demary, 2020], le modélisateur peut définir un comportement avec une fonction-impact pour chacun des quatre quadrants du tableau 3.1 que nous recopions juste en dessous.

	Communicant	**Non-communicant**
Proactif	Je raisonne, j'informe le leader, je conteste si besoin, j'obéis au final. (Exemplaire / Courageux)	Je prend des décisions, je conteste. (Loup solitaire)
Passif	Je demande quoi faire et je fais ce qu'on me dit. (Mouton / Faiseur)	Si on ne me dit rien, je suis la procédure. Sinon, je fais ce qu'on me demande (Yes people / Ressource)

TABLE 3.1 – Comportement selon le style « suiveur » et le style « communicant ». Si chaque case impactait la même opération, nous devrions définir un comportement par case pour y associer un impact.

Activation d'un comportement Une fonction-impact définit *comment* un modèle cognitif altère une opération (et le comportement de l'agent). Mais, il faut aussi définir *quand* cet impact a lieu en fonction du profil de l'agent et de son état (qui capture la situation). C'est ce que Faur [Faur, 2016] souligne dans son modèle PERSEED. Par exemple, un agent introverti se comporte de manière extravertie en présence d'amis, mais de manière introvertie en présence de collègues de travail. Il s'agit donc de déclencher des fonctions-impacts différentes en fonction de la situation.

Ce mécanisme est ajouté au travers des comportements. Nous introduisons une caractéristique d'arité 0 : $ActivationFunc(\cdot)$. Cette caractéristique définit une fonction qui prend en entrée l'état de l'agent et retourne $true$ ou $false$.

Ainsi, uniquement les fonctions-impacts des comportements actifs sont considérés. Nous détaillerons comment une opération est évaluée avec les bonnes fonctions-impacts, lors de la partie 3.3 sur le dynamisme du modèle.

Ajout d'un comportement à un agent Nous notons $Beh(i)$ l'ensemble des comportements d'un agent i. Ces comportements sont ceux que l'agent i pourrait exprimer, mais pas forcément ceux qu'il doit réaliser. Cela dépendra de si la fonction d'activation du comportement est active ou non.

C'est pourquoi, nous introduisons la caractéristique d'arité 1 : $Active(i, b), b \in Beh$ qui indique que le comportement b est actif pour i. Le méta-modèle ajoute cette relation lorsque la fonction d'activation retourne vrai, comme nous le verrons lors de la partie 3.3.

Exemple Pour la dernière fois, nous reprenons notre exemple pour une représentation complète avec les opérations. Nous avions précédemment rajouté dans notre flux, une opération pour mettre à jour le stress et une caractéristique de stress. Désormais, nous voulons ajouter un comportement de stress.

Nous allons donc définir un nouveau comportement : $BStress$. Ce comportement est actif uniquement lorsqu'un agent est stressé. Nous considérons que ce comportement impacte l'opération $SuitableActions$, soit l'ensemble des potentielles actions envisageables, en rajoutant une action de fuite par exemple.

$$Impact(BStress, SuitableActions)$$

Il impacte aussi l'opération $ActionSelection$ pour favoriser les actions en liens avec le stress.

$$Impact(BStress, ActionSelection)$$

Enfin, il impacte l'opération $UpdateStress$ pour indiquer comment mettre à jour le stress.

$$Impact(BStress, UpdateStress)$$

Nous obtenons donc la figure 3.14 et les ensembles suivants :

$$\begin{aligned} \mathbf{O} &= \{SuitableActions, ActionSelection, UpdateStress\} \\ Beh &= \{BStress\} \\ Impact(BStress, *) &= \{SuitableActions, ActionSelection, UpdateStress\} \\ Impact(*, ActionSelection) &= \{BStress\} \end{aligned}$$

Nous verrons plus précisément de quoi il s'agit lorsque nous parlerons de la méthode de sélection par graphes d'influences et de préférences, en section 3.5.

L'exemple est simple, mais si nous avions intégré un modèle de stress à la Lazarus et Folkman [Lazarus and Folkman, 1984], alors nous disposerions d'une opération retournant la stratégie de *coping* adoptée. La fonction-impact du comportement de stress pourrait ensuite impacter favorablement les actions correspondant à cette stratégie lors de la sélection d'action.

Mini-Bilan

Nous avons présenté dans cette section, comment nous avons formalisé le méta-modèle OPACK autour de la notion d'identifiants $i \in \Omega$ et de caractéristiques $c \in \mathbf{C}$. Le principe est que n'importe quel identifiant peut-être associé à d'autres par le biais de caractéristiques d'arités variées.

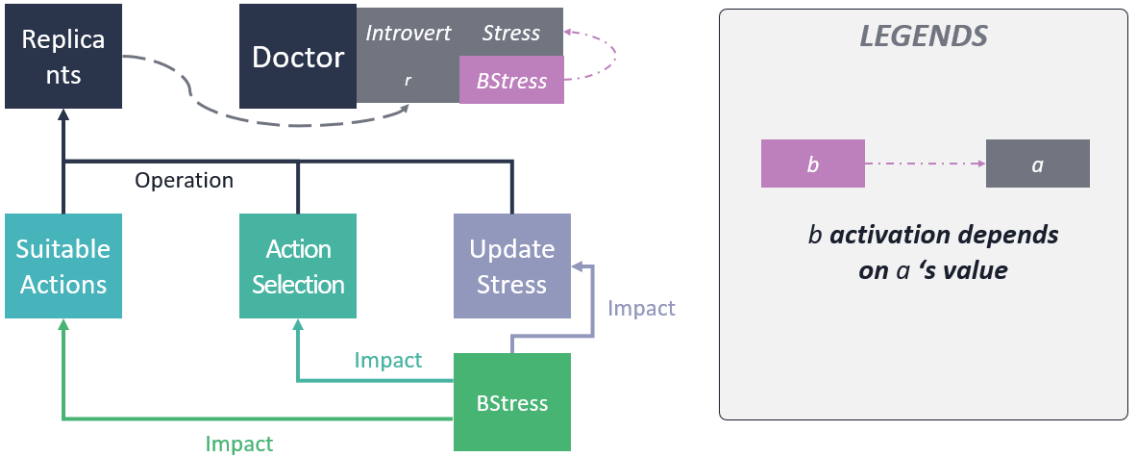


FIGURE 3.14 – Exemple d'une représentation d'un comportement et des fonctions-impacts. $BStress$ est actif uniquement en fonction de la valeur de stress de l'agent, *i.e.* si le stress est suffisamment élevé.

Exemple

Représentons : « Arthur, introverti, se considère l'ami de Bob ». Nous aurions alors :

$$\begin{aligned} \Omega &= \{Arthur, Bob, Friend, Introvert\} \\ \mathbf{C} &= \{Introvert, Friend\} \\ Agt &= \{Arthur, Bob\} \\ \mathbf{C}(Arthur) &= \{Introvert, Friend\} \\ Friend(Arthur, *) &= \{Bob\} \\ Friend(*, Bob) &= \{Arthur\} \end{aligned}$$

Nous avons appliqué ce principe au travers des différents concepts d'OPACK. L'avantage avec ce formalisme est que ces cinq concepts sont composables, ce qui permet la modularité et la généricité. De plus, nous avons vu comment grâce à la composabilité, les modèles cognitifs ont pu les impacter.

Maintenant que nous avons achevé la représentation du méta-modèle, nous allons décrire l'ensemble de ces mécanismes, soit le dynamisme du méta-modèle.

3.3 Méta-modèle OPACK : Dynamisme

OPACK n'est pas juste un méta-modèle descriptif d'un SMA *à la* Bernon [Bernon et al., 2005]. Notre modèle gère aussi l'exécution des modèles d'agent. OPACK fonctionne par cycle, découpé en trois phases successives (*cf.* figure 3.15) :

— **Phase 1 - Perception** : Mise à jour de P selon l'ensemble des sens de nos agents (*et capteurs pour les artefacts*).
— **Phase 2 - Modèle d'agent** : Exécution de l'ensemble des opérations de O, soit l'exécution des modèles d'agent.
— **Phase 3 - Action** : Mise à jour de A, soit de l'ensemble des actions menées par les agents aux travers de leurs effecteurs.

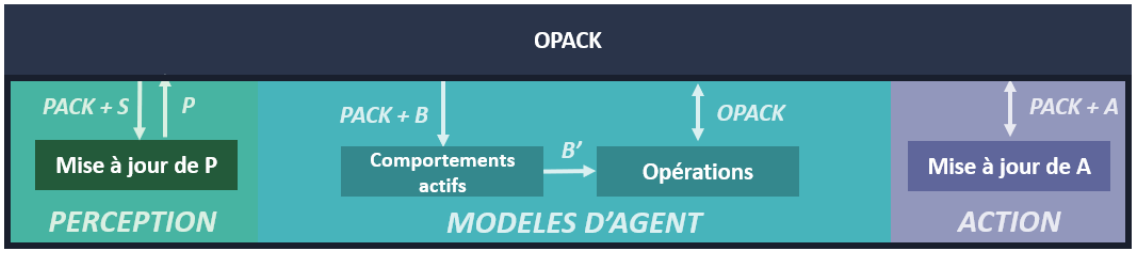


FIGURE 3.15 – Le cycle du méta-modèle est une succession de trois phases : la mise à jour des perceptions, l'exécution des modèles d'agent, puis des actions.

L'exécution d'un cycle permet de passer l'état de l'environnement au pas de temps suivant. Soit $World = \Omega, C, World(t)$ est l'état de l'environnement à un instant t. Nous introduisons la fonction $step(World)$, la fonction exécutant un cycle pour passer de $World(t)$ à $World(t + dt)$, avec dt représente un pas de temps quelconque.

Nous détaillons les trois phases, puis comment cela se traduit du point de vue d'un modèle d'agent classique.

3.3.1 Phase 1 - Perception

Lors de cette phase, OPACK va mettre à jour l'ensemble des perceptions P. Nous avons vu comment l'ensemble $P(i)$ était déterminé pour un agent i. Dès lors, nous avons :

$$P = \bigcup_{\forall i \in Agt} P(i)$$

Toutefois, la difficulté lors du calcul de P est de déterminer les identifiants qu'un sens peut percevoir (*e.g.* comment deux agents peuvent se percevoir auditivement ou visuellement). Nous avions alors indiqué que le modélisateur devait spécifier une fonction pour le faire. C'est que nous allons voir maintenant.

Fonction $perceive$ Le modélisateur doit définir une fonction $perceive : Agt \times Sen \longrightarrow \mathcal{P}(Agt)$ qui renvoie l'ensemble des identifiants perceptibles par un agent, en fonction de son état courant, pour un sens donné. Par exemple, l'agent $Doctor$ peut percevoir via le sens $Hearing$ (audition) l'artefact $Patient$, à condition de ne pas être situé trop loin du patient.

Cette fonction $perceive$ dépend des fonctionnalités souhaitées dans la simulation (*cf.* figure 3.17) : elle est définie par le modélisateur. Elle utilise les caractéristiques du sens et des agents ou artefacts pour vérifier les conditions de perception (*e.g.* ne pas être trop loin). Dans notre exemple, $Hearing$ peut ainsi avoir une caractéristique $MaxDistance$ qui définit la distance limite d'audition, et les agents $Doctor$ et $Patient$ des coordonnées dans l'espace permettant de calculer une distance.

Nous soulignons que le modélisateur a juste besoin de définir la fonction $perceive$ et les capacités perceptives des sens. Le système calcule le reste.

FIGURE 3.16 – Capture d'écran de l'EV VICTEAMS [Huguet et al., 2016]. La perception dans cet EV peut utiliser un moteur physique pour correspondre à notre manière de percevoir.

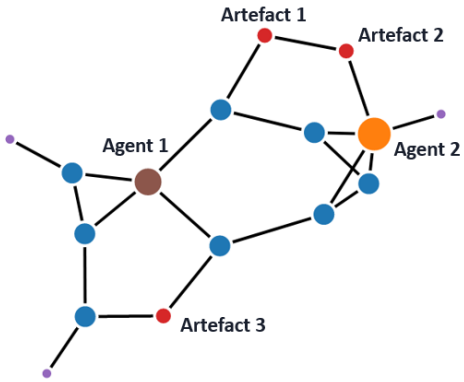


FIGURE 3.17 – EV sous forme de graphe. Comment le champ visuel est déterminé ? Est-ce qu'il s'agit uniquement des éléments du noeud où se trouve l'agent ou des noeuds avoisinants ?

3.3.2 Phase 2 - Modèle d'agent

Lors de cette phase, l'ensemble des modèles d'agent sont exécutés. Plus exactement, l'ensemble des opérations $o \in O$ sont exécutées selon leur ordre topologique. Nous rappelons que l'ordre est déterminé par les liens de précédences entre les opérations, ainsi que les entrées et sorties. Les opérations sans prédécesseurs commencent en premier.

Toutefois, avant d'évaluer les opérations, un autre ensemble doit être évalué. L'ensemble des comportements Beh sont à évaluer, à partir de leur fonction-activation, pour déterminer ceux actifs ou non. Ainsi pour tous les agents i, nous appliquons la fonction de la caractéristique $ActivationFunc$:

$$\forall b \in Beh(i), ActivationFunc(b)(i) \longrightarrow \{true, false\}$$

Si $true$ est retournée, alors la relation $Active(b, i)$ est ajoutée pour indiquer que l'agent i doit exprimer le comportement b.

Ainsi, lorsqu'une opération $o \in O$ est évaluée, uniquement l'ensemble des fonctions-impacts de cette opération, pour les comportements actifs, sont considérées. Cet ensemble est déterminé grâce à la notation suivante : $Active(i, *)$.

Ensuite, l'état $PACK$ de nos agents est manipulé à ce moment-là. Nous précisons que P est modifiable, mais que nous déconseillons de le faire. P est reconstruit à chaque nouveau cycle. Toutefois, si P est modifiée, par exemple, un agent supprime sa perception d'une paire de ciseaux, alors cet agent ne pourra plus accéder à cette information pour tout le cycle. Idéalement, un processus de perception doit transmettre un ensemble P' et laisser à d'autres processus ou flux opérer sur un ensemble P non altéré (par exemple, pour supporter des processus non-conscients : *je n'avais pas fait attention à ce moment-là, mais maintenant que tu en parles, je me souviens* ...).

Enfin, de manière générale, l'ensemble des actions A sera modifié. Des agents auraient alors décidé de mener de nouvelles actions, quitte à interrompre celles en cours. Une fois que les agents ont indiqué les actions qu'ils veulent réaliser ou qui sont en cours de réalisation, nous pouvons passer à la phase suivante.

3.3.3 Phase 3 - Action

Lors de cette étape, l'ensemble des actions A sont exécutées. Le diagramme d'activité de la figure 3.18 résume les transitions d'états des actions.

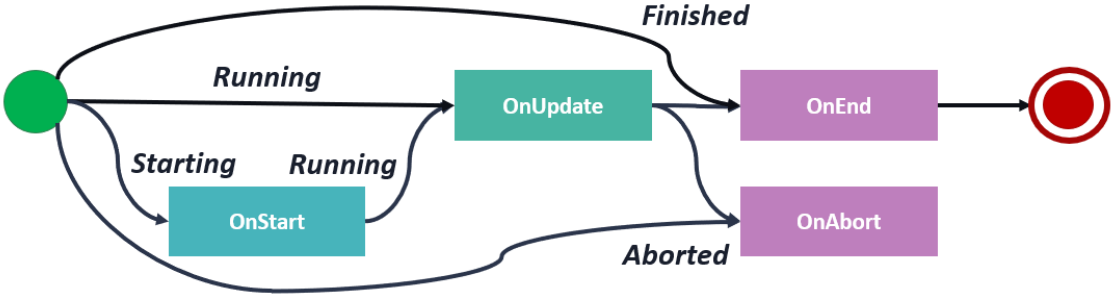


FIGURE 3.18 – Diagramme d'activité des actions. Selon leur état, les actions passent à travers différentes étapes. En un cycle, une action peut traverser plusieurs étapes : *OnStart*, *OnUpdate* et *OnEnd* par exemple.

C'est à l'environnement virtuel, soit au modélisateur, d'indiquer ce qui se passe lorsqu'une action débute, est mise à jour, se termine ou lorsqu'elle est interrompue. À tout moment, l'environnement peut indiquer lorsqu'une action est terminée, en changeant le statut d'une action a en $Finished$ ou $Aborted$.

3.3.4 Cycle procédural de l'agent

Désormais, si nous nous plaçons du point de vue d'un modèle d'agent, le cycle est exactement le même (*cf.* figure 3.19) : perception, délibération et action.

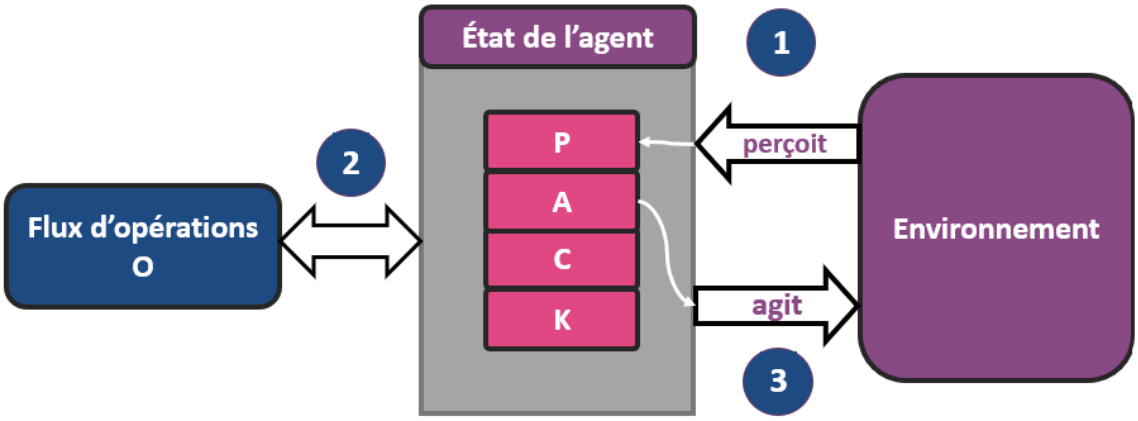


FIGURE 3.19 – Schématisation du cycle d'un agent.

Perception : OPACK communique à l'agent l'ensemble **P** de percepts.

Délibération : exécution de l'ensemble des flux de l'agent, qui modifie l'état de l'agent en agissant possiblement sur les ensembles **P**, **A**, **C** et **K**.

Action : l'ensemble **A**, éventuellement modifié suite à l'exécution du flux, est utilisé par l'environnement, qui est responsable de l'exécution des actions. Le succès ou l'échec d'une action est constaté par l'agent à travers les percepts (par exemple, j'essaye d'ouvrir la porte et je me rends compte qu'elle ne s'ouvre pas alors que l'action est terminée). L'agent peut choisir de supprimer de **A** les actions terminées mais l'environnement ne modifie jamais **A** de lui-même, bien que rien ne lui empêche concrètement.

3.4 Exemple : intégration de modèles cognitifs

Nous avons jusqu'ici décrit l'ensemble du méta-modèle OPACK : la représentation des différents concepts puis son dynamisme. Désormais, nous allons décrire un exemple de bout en bout sur l'intégration de modèles cognitifs (simplifiés).

3.4.1 Modèle de départ

Notre point de départ sera le modèle adapté de REPLICANTS que nous avons construit précédemment. Le flux est composé de trois opérations et la figure 3.20) :

— « Actions envisageables »
— « Sélection d'action »
— « Exécution »

L'opération « Exécution » permet d'acter l'action choisie par l'opération de sélection.

Nous allons montrer comment intégrer dans ce modèle les modèles cognitifs de Driskell [Driskell et al., 2018], de Demary [Demary, 2020], de Fadier [Fadier et al., 2003] et de Lazarus & Folkman [PhD and PhD, 1984].

FIGURE 3.20 – Flux de départ composé de trois 'intégration des modèles cognitifs

Ensuite, nous définissons la fonction-stratégie pour ces trois opérations, ce qui nécessite de définir 1) le type de retour $\tau(s_o)$ des fonctions-impacts de l'opération et 2) le résultat obtenu par défaut lorsque l'ensemble d'impacts $Impact(*, o)$ est vide.

- Pour *« actions envisageables »*, nous choisissons que $\tau(s_o)$ est, comme pour o, un ensemble d'actions. Par défaut, la stratégie renvoie le résultat du parcours de l'arbre défini dans le modèle REPLICANTS, stocké dans **K**, c'est-à-dire un ensemble d'actions comprenant au moins, l'action *« ne rien faire »*.
- Pour *« sélection d'action »*, nous choisissons que $\tau(s_o)$ est un ensemble de couples $(action, score)$. Par défaut, lorsque cet ensemble est vide, le système choisit une action aléatoirement dans la liste des actions fournies en entrée de l'opération. [2]
- Pour *« exécution »*, nous choisissons que $\tau(I_o)$ est vide. Par défaut, la stratégie modifie A en ajoutant l'action en entrée, mais exécute l'ensemble des fonctions-impacts qui peuvent modifier l'action choisie.

Nous souhaitons désormais impacter le comportement de l'agent par le biais de modèles cognitifs.

3.4.2 Ajout d'un impact : modèle cognitif de followership

Commençons par un comportement simplifié de Demary [Demary, 2020] : un agent *passif* suit les ordres de son supérieur.

Nous ajoutons dans C une nouvelle caractéristique d'arité 0 stable : $Passive$, pour déterminer si un agent est *passif* ou non. Idem, nous ajoutons dans K la possibilité

2. Il s'agit ici d'une vision simplifiée par rapport au modèle REPLICANTS. Celui-ci calcule en réalité un ordre total sur les actions en fonction des pré-conditions des actions.

d'indiquer si une action est *ordonnée* ou non par un supérieur, grâce à la caractéristique d'arité 0 $Ordered$

Enfin, nous ajoutons une fonction-impact dans l'opération *« sélection d'action »*. Elle est active uniquement si l'agent est *passif*. Elle donne, pour chaque action, un score de 1 aux actions ordonnées par un supérieur et de -1 si elle a été interdite par un supérieur et de 0 sinon.

De même, nous pourrions réappliquer cette méthode pour ajouter des fonctions-impacts pour chacun des trois autres cases du tableau du modèle de Demary.

3.4.3 Ajout d'un impact : modèle cognitif de Fadier

Ensuite, nous allons ajouter les ALU du modèle de Fadier [Fadier et al., 2003] avec une interprétation simplifiée : une personne non-réglementaire pourrait réaliser des ALU.

Nous ajoutons dans C une nouvelle caractéristique stable, soit un tag pour déterminer si un agent est *réglementaire* ou non. Idem, nous ajoutons dans K la possibilité d'indiquer si une action est une ALU.

Enfin, nous ajoutons une fonction-impact dans l'opération *« sélection d'action »*. Elle est active uniquement si l'agent n'est pas *réglementaire*. Elle donne, pour chaque action, un score de 1 aux actions ALU ou de 0 sinon.

3.4.4 Ajout d'une opération : modèle cognitif de Lazarus & Folkman

Supposons désormais que nous souhaitons opérationnaliser un modèle de stress, comme celui de Lazarus [PhD and PhD, 1984], utilisé dans le modèle cognitif de Driskell. Nous avions vu dans l'état de l'art qu'il y avait deux évaluations : primaire et secondaire pour évaluer le contrôle et le stress perçu. Pour cet exemple, nous allons considérer qu'une seule opération : l'évaluation cognitive (*appraisal* en anglais). Sa fonction est de travailler sur un ensemble de variables d'évaluation dans l'environnement pour calculer un état émotionnel, avec notamment le stress.

Nous ajoutons donc une variable de stress dans l'ensemble **C** de l'état de l'agent. Ensuite, nous allons ajouter deux opérations : notre opération d'évaluation cognitive, mais aussi une opération de perception. L'opération d'évaluation cognitive opère sur les percepts de l'agent et son état (qui correspondent aux variables d'évaluation cognitive) pour calculer la variable de stress. Nous ajoutons donc une fonction-impact dans cette opération pour traduire cet effet. Enfin, nous ajoutons un lien de précédence entre cette opération d'évaluation et la sélection d'action pour s'assurer qu'elle ait lieu avant la sélection.

3.4.5 Ajout de plusieurs impacts : modèle cognitif de Driskell

Enfin, pour refléter les impacts du stress sur les différentes opérations, nous allons nous appuyer sur les travaux de Driskell [Driskell et al., 2018]. Lors de l'état de l'art, nous avions décrit plusieurs mécanismes, en voici quelques-uns :

1.a : réduction du champ de vision.

4.a : impact physiologique, comme le tremblement.

5.a : réduction de la tendance à aider les personnes.

Nous traduisons ces fonctions comme fonctions-impacts dans les opérations adéquates, actives lorsque l'agent est stressé :

1.a : Au sein de l'opération de perception : une fonction-impact serait, par exemple, de filtrer les percepts dans la périphérie de la zone de vision.

4.a : Au sein de l'opération de sélection d'action : une fonction-impact serait, par exemple, l'attribution d'un score -1 aux actions aidant d'autres personnes.

5.a : Au sein de l'opération d'action : une fonction-impact serait, par exemple, l'ajout du tremblement aux actions qui le supportent.

3.4.6 Discussions

Au final, nous obtenons la figure 3.21 :

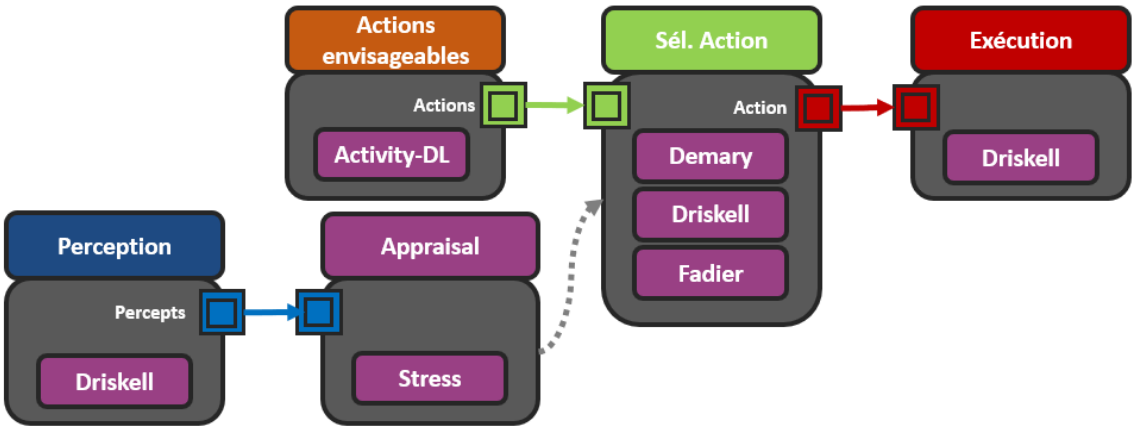


FIGURE 3.21 – Nouveau flux suite à l'intégration des modèles cognitifs

Cet exemple simple montre comment il est possible d'intégrer plusieurs modèles cognitifs au sein d'un modèle d'agent dans le méta-modèle OPACK. Notamment, comment grâce à la notion de fonctions-impacts et d'opérations, les modèles cognitifs peuvent impacter modulairement diverses étapes du modèle d'agent.

Nous verrons, lors du chapitre 6, quelques évaluations préliminaires de nos propriétés de sensibilité, d'intelligibilité et de représentativité. Avant cela, nous présentons

une autre contribution : une stratégie de choix pour la sélection d'action utilisant des graphes d'influences et de préférences, pour l'intelligibilité et la sensibilité de génération de comportements.

3.5 Graphes d'influences et de préférences

Le comportement de l'agent est le résultat direct des actions sélectionnées par l'agent. Lors de cette section, nous complétons le méta-modèle en proposant une stratégie d'opération de sélection d'action. Elle permet d'intégrer modulairement l'impact de modèles cognitifs de manière intelligible et sensible. Cette stratégie est composée de deux éléments :

- un graphe d'influences qui ordonne les actions selon la préférence des modèles cognitifs
- un graphe de préférences qui ordonne les modèles cognitifs entre eux.

Si le graphe d'influences ne sort pas de vainqueurs, alors le graphe de préférences permet de décider, sinon la sélection est aléatoire parmi les actions candidates.

Dans le contexte de cette section, nous considérons les termes « comportement » et « modèle cognitif » comme interchangeables, idem pour « action » et « tâche ».

3.5.1 Principe général

Une opération de « sélection de tâches » (*cf.* l'algorithme 1) prend en entrée un ensemble de tâches candidates $T \subset A$ et renvoie une tâche sélectionnée $t \in T'$. Le graphe d'influences est un graphe biparti avec, dans ce contexte, des comportements $b \in B$, ou plus précisément des fonctions-impacts, indiquant quelles tâches sont appropriées ou non, symbolisées par des influences positives ou négatives. Le score d'une tâche est la somme des influences positives ($+1$) et négatives (-1). La tâche ayant le score le plus élevé indique celle qui satisfait le plus les comportements. Elle est donc sélectionnée et correspond à la sortie du processus. Les tâches sont des identifiants, possédant un nom et un ensemble de caractéristiques. Ces informations permettent aux comportements de raisonner et de décider si elles doivent être favorisées ou non. L'ensemble des caractéristiques dépend de l'application. Nous pouvons supposer qu'au moins le coût sera présent. Ce graphe est construit automatiquement à partir des comportements considérés et des tâches candidates.

3.5.2 Fonction d'influence

Chaque comportement impactant cette opération définit au travers de sa fonction-impact une fonction d'influence. Cette fonction définit une liste d'influences positives ou

Algorithm 1: Task-selection process

Input a set of candidate tasks T, a set of behaviours beh.
Output $t \in T$
$influence\_graph \leftarrow \{\}$;
for $b \in beh$ **do**
  for $influence \in compute(b, T)$ **do**
    $add\_edge(influence\_graph, influence)$;
$highest\_tasks \leftarrow highest\_score(influence\_graph)$;
if $size\_of(highest\_tasks) == 0$ **then**
  return t_0;
else
  return $random\_element(highest\_tasks)$;

négatives envers certains éléments $t \in T$. Par exemple : influencez positivement la tâche la plus courte, influencez négativement la tâche la moins chère.

Comme OPACK permet de composer librement **K**, un modèle peut rajouter des informations associées aux actions pour permettre le fonctionnement d'une fonction d'influence. Par exemple, pour le modèle de Fadier [Fadier et al., 2003], nous avions ajouté des tags (soit des caractéristiques d'arité 0 sans autre donnée que l'identifiant en lui-même) aux actions pour justement supporter la fonction d'influence.

3.5.3 Sélection

La tâche ayant le meilleur score est sélectionnée. Si plusieurs tâches ont le meilleur score, une tâche est sélectionnée de manière aléatoire. Si nous avions une raison de sélectionner l'une des tâches, cette raison serait une règle de haut niveau, c'est-à-dire une fonction d'influence.

3.5.4 Préférences

Par ailleurs, à elles seules, les influences ne suffisent pas à décrire certains comportements. C'est pourquoi nous proposons de les coupler avec un graphe de préférences. Ce graphe permet d'affiner le choix du comportement à sélectionner en cas d'indécision. Le principe du graphe de préférences est d'ordonner partiellement les comportements, soit en fonction d'une justification issue d'un modèle cognitif : *« dans une situation de stress intense, une réduction de la communication est observée »*, soit si le scénario favorise un comportement. Par exemple, ils peuvent indiquer une préférence sur l'une des tâches. Nous allons donc sélectionner celle-ci, plutôt que de tirer au hasard. Comme le graphe d'influences, ce graphe est déduit automatiquement en fonction des règles de préférence établies et des comportements considérés.

La combinaison de ces deux graphes améliore l'expressivité de la stratégie, ce qui nous permet finalement d'opérationnaliser les comportements plus finement.

Nous aurions pu utiliser des poids à la place des préférences, comme l'approche d'ACTIVITY-DL (*cf.* état de l'art, section 2.2). Mais, cela aurait été en contradiction avec nos objectifs de modularité et de généricité. En particulier, au lieu de spécifier des préférences entre les seuls comportements concernés, nous aurions dû ajuster les poids de tous les comportements. En outre, l'utilisation des préférences est plus intelligible que celle des poids.

3.5.5 Intelligibilité

Si la représentativité nécessite une certaine variabilité dans les comportements exprimés, nous voulons que les actions de l'agent aient du sens pour l'observateur et que ses décisions apparentes soient toujours explicables, quitte à avoir des comportements plus tranchés (d'autant plus que nous sommes dans un contexte d'apprentissage). Selon [Chakraborti et al., 2019], l'explicabilité inclut la prise en compte du modèle de l'observateur pour fournir les explications pertinentes. Cependant, l'explicabilité dépasse les objectifs fixés par cette thèse. C'est pourquoi, nous nous limitons à fournir un moyen intelligible. Du point de vue système et/ou modélisateur, nous voulons aussi permettre l'identification des causes d'un comportement, à travers les modèles cognitifs et les stratégies utilisées.

Le mécanisme des graphes d'influences et de préférences est un moyen visuel et intelligible pour identifier les facteurs (c'est-à-dire les règles des modèles cognitifs) qui ont conduit à ce comportement, par rapport à d'autres possibilités. Notre volonté était de pouvoir retracer l'influence de chaque modèle cognitif sur la sélection.

Un autre mécanisme, si la visualisation du graphe n'est pas envisageable, est de fournir une explication textuelle qui prend la forme suivante : « *L'agent a fait* X*, parce que* Y*, il fait* Z*.* » X est le comportement réalisé. Y est l'information sur l'état interne (c'est-à-dire le profil et l'évaluation de la situation) pertinente pour la règle Z. Z est la règle de haut niveau qui a conduit à X. Si plusieurs modèles cognitifs sont liés à la sélection du comportement, chacun fournit ses Y et Z. Nous utilisons d'ailleurs ce principe lors d'une expérimentation préliminaire en fin de chapitre 6.

3.5.6 Exemple

La figure 3.22 est un exemple de graphe d'influences. En haut, se trouvent trois comportements et en bas, des tâches. Avant de détailler le fonctionnement de cet exemple, nous allons détailler les données.

Tâches

Nous supposons que les actions possèdent les caractéristiques suivantes :

— **cost B** : coût de base pour exécuter cette tâche ;

— **cost R** : coût réglementaire additif si l'on suit les règles .

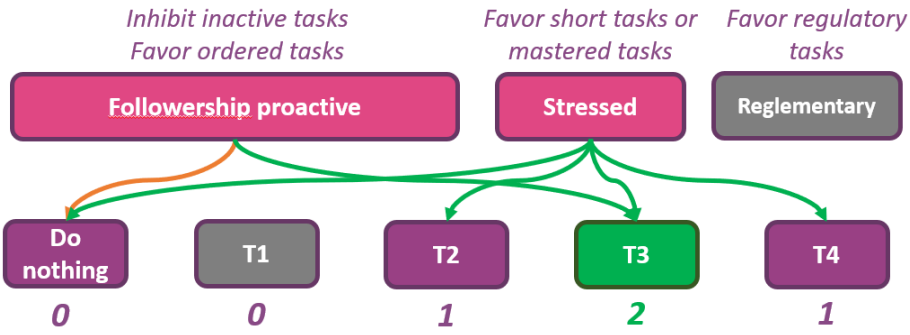


FIGURE 3.22 – Diagramme de la stratégie de « choix par graphe d'influences ». Dans cet exemple, la tâche T3 possède le score maximal. C'est donc elle qui est sélectionnée.

— **Tags** : ensemble de symboles pour décrire les tâches. Nous utilisons ici :
  — SCX, où X est le niveau de compétence de l'PVA. Plus le chiffre est élevé, mieux c'est. Par exemple, $SC3$ est un niveau de compétence supérieur à $SC1$.
  — *Ordered*, indique un ordre de leur chef.

Plus l'ensemble des caractéristiques est avancé, plus les fonctions d'influence peuvent l'être. Le tableau suivant décrit les tâches utilisées par la figure 3.22 :

Name	cost B	cost R	Tags
Do nothing	*0*	*0*	*None*
T1	*1*	*0*	*SC3*
T2	*2*	*0*	*SC1*
T3	*3*	*0*	*SC2 - Ordered*
T4	*2*	*5*	*SC2 - Regulatory*

TABLE 3.2 – Ensemble de nos tâches.

Comportements

Le tableau suivant décrit les comportements utilisés de la figure 3.22 :

Scénario

Disons pour cet exemple que nous avons les paramètres suivants. Notre agent est : **proactif**, **non réglementaire** et **stressé**. La situation est considérée comme **urgente**.

Description

Avec ces données et ce scénario, nous obtenons le graphe d'influences présenté dans la Fig.3.22. Selon les règles d'activation, les deux comportements « followership

Behaviour	Activation function	Influence function
Followership Proactive	*is proactive*	$-$ *inactive tasks* $+$ *ordered tasks*
Stressed	*is stressed*	$+$ *mastered tasks* *($\leq$ to skill)* *or shortest tasks*
Regulatory	*situation is* $\neg$ *urgent* ***xor*** *It is urgent and agent is regulatory*	$+$ *regulatory tasks*

TABLE 3.3 – Ensemble de comportements.

proactif » et « stressé » sont activés. Le comportement « réglementaire » n'est pas activé, car la situation est urgente et il n'est pas réglementaire. L'exemple est assez simple pour ne pas tout détailler. Nous ne précisons le fonctionnement que pour les deux tâches suivantes :

— L'action « Ne rien faire » est influencée négativement par « followership proactif » mais positivement par « stressé » en tant que tâche courte.

— « T3 » est influencée positivement par « followership proactive » car cette tâche lui a été ordonnée, de même que positivement par « stressed » car c'est une tâche qu'il maîtrise.

3.5.7 Discussion

L'utilisation d'un graphe d'influences se justifie par son intelligibilité et sa modularité. En effet, nous pouvons retracer quels comportements sont responsables de la sélection. De plus, nous pouvons ajouter d'autres comportements, sans avoir à modifier les comportements existants et l'opération affectée. Mais, à elle seule, elle n'est pas suffisante pour décrire certains comportements. C'est pourquoi nous proposons de le coupler avec un graphe de préférences, pour affiner le choix du comportement à adopter en cas d'indécision.

À première vue, si nous ne prenons pas en compte les préférences, mais plutôt les influences dans un premier temps, c'est que nous souhaitons privilégier ce que l'agent préfère, plutôt que les préférences d'utilisateurs externes, comme le scénario. Cependant, l'inverse est tout à fait possible et pourrait constituer une autre stratégie.

3.6 Bilan

Lors de cette section, nous avons d'abord présenté notre méta-modèle OPACK. Il abstrait un modèle d'agent autour de cinq concepts : **O**peration, **P**erception, **A**ction, **C**aractéristique et **K**nowledge (connaissances en français). Ces concepts sont composables pour permettre la modularité, généricité et l'intégration de modèles cognitifs.

Notamment, la dynamique d'un modèle d'agent est décrite par le biais de flux d'opérations. La liberté de conception du graphe offre beaucoup de possibilités au modélisateur. L'opérationnalisation de modèles cognitifs est simplifiée par la nature du flux et des opérations.

Ensuite, en ce qui concerne la génération de comportements sensibles et intelligibles, nous avons proposé une stratégie de sélection d'action pour notre méta-modèle : graphes d'influences et de préférences.

Toutefois, une limite de cette approche est que, puisqu'il ne s'agit pas d'une implémentation fine de modèles cognitifs, nous devons tester les comportements obtenus pour vérifier s'ils sont bien intelligibles, *i.e* en correspondance avec la combinaison de modèles cognitifs.

De plus, le fait que chaque opération possède en entrée l'agent brise une propriété cruciale de la programmation par flux de Morrison [Morrison, 1994] : les opérations n'agissent pas uniquement sur les données du flux. Elles ont aussi accès à l'ensemble des données de l'agent. Des effets de bords sont donc possibles, ce qui complexifie beaucoup l'implémentation d'une logique concurrentielle, mais est nécessaire pour les opérations.

Récapitulatif Dans le tableau 3.4, nous résumons les principales fonctionnalités proposées dans notre méta-modèle et comment elles soutiennent les propriétés que nous avons identifiées dans l'état de l'art :

TABLE 3.4 – Tableau approximatif de l'impact des fonctionnalités sur les propriétés

	Modularité	**Généricité**	**Représentativité**	**Sensibilité**	**Intelligibilité**
Dissociation des comportements et des données	+	+	~	-	-
5 concepts composables	+	+	+	-	-
Opérations, stratégies et impacts	+	+	~	+	+
Graphes d'influences	+	+	-	+	+

Chapitre 4

Implémentation

Contents

Dans ce chapitre, nous présentons l'implémentation d'une approche pour la simulation d'agents virtuels suivant les principes du méta-modèle OPACK.

4.1 Introduction

La simulation d'agents virtuels est une opération coûteuse, surtout lorsque le nombre d'agents et la complexité de leur raisonnement, *e.g* cognitifs, augmentent. Dès lors, certaines plateformes comme Repast HPC (High Performance Computing) [Dubitzky et al., 2012] s'orientent vers des approches distribuées où les ressources computationnelles sont très importantes (*cf.* superordinateur [1]).

Au-lieu de considérer une abondance de ressources, nous souhaitons explorer dans ce chapitre la simulation d'agents virtuels avec des ressources limitées. Plus exactement, nous considérons des simulations exécutées sur des ordinateurs personnels. Notre volonté est de réduire leur coût computationnel. Notre motivation principale est de rendre ces simulations viables en temps interactifs, *e.g. comme cela se fait dans les jeux vidéo.* Il est donc nécessaire de laisser des ressources pour d'autres tâches, comme les graphismes, la physique *etc.* .

Nous employons donc le terme de « simulation interactive ». « Interactive » induit qu'un utilisateur interagit avec la simulation et affecte son évolution de manière continue. Il doit pouvoir observer les effets de ses interactions dans un temps raisonnable. Ensuite, nous sous-entendons un second aspect : une approche synchrone, soit un fonctionnement par cycle à la GAMA [Taillandier et al., 2019] ou Netlogo [Wilensky, 1999]. Les approches asynchrones comme Jade [2] sont plus pertinentes dans le cadre d'une simulation distribuée.

4.1.1 Problématique

Ces simulations interactives imposent des contraintes sur les performances du système, pour proposer une expérience fluide à l'utilisateur. Par performance ici, nous entendons que plus un système réalise un cycle rapidement, plus il est performant. Plus il est performant, plus l'expérience est fluide pour l'utilisateur.

> Nous ne traitons pas ici la question de la qualité de l'expérience, aux travers des comportements générés par exemple.

Or, comme nous l'avons mentionné au début de ce chapitre, la simulation d'agents virtuels est coûteuse. Cela met en danger la fluidité, et donc l'expérience utilisateur. En 2020, Sébastien Picault et Vianney Sicard [Picault and Sicard, 2020] amorce une discussion autour d'une nouvelle approche pour les SMA : identifier ce qui doit être agentifié ou pas, et l'incorporation de méthodes issues d'autres disciplines scientifiques : statistique, physique, *etc.* , pour la simulation. Paradoxalement, il faut chercher à simuler le moins d'agents, pour notamment réduire le coût computationnel, mais également intégrer l'expertise d'autres disciplines comme la physique. Ils introduisent ainsi la notion de simulation multi-paradigmes et la dualité agent/information :

1. https://fr.wikipedia.org/wiki/Superordinateur
2. https://jade-project.gitlab.io/

Citation 13 - Dualité agent/information

« [...] il n'est pas nécessaire de toujours simuler toutes les entités possibles d'un modèle. [...] En particulier, la notion de phéromone est un bon exemple de ce qui, en toute rigueur, devrait être modélisée par des agents (ce sont bien, après tout, des molécules qui se déplacent, s'évaporent, etc.) mais qu'il est plus « simple » de représenter par une information. Cette dualité agent/information évoque la notion de champ en physique et mériterait d'être généralisée. »

Picault, S. and Sicard, V. (2020). Les meilleurs agents sont ceux qu'on ne simule pas: vers des architectures de simulation multi-paradigmes? In *28e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2020)*. Cépaduès

Toutefois, comme l'énoncent les auteurs de [Picault and Sicard, 2020], les difficultés sont nombreuses : l'interdisciplinarité, le manque de méthode, *etc.* .

Citation 14 - Extrait sur les difficultés d'une telle approche

« Quelles méthodes mobiliser pour construire des architectures de simulation hybrides, avec quels formalismes, quels algorithmes, issus de quelles disciplines ? C'est précisément le cœur du sujet et le programme des recherches à mener durant la prochaine décennie. »

Picault, S. and Sicard, V. (2020). Les meilleurs agents sont ceux qu'on ne simule pas: vers des architectures de simulation multi-paradigmes? In *28e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2020)*. Cépaduès

Ensuite, un autre domaine fait écho à la proposition de l'article précédent [Picault and Sicard, 2020]. En programmation, Design orienté-data (DOD) est une approche de design pour bien structurer son programme selon ses données, *i.e* les SMA dans notre cas, de manière à efficacement utiliser des ressources limitées à disposition. En résumé, nous pensons que l'application des principes DOD aux concepts de SMA seraient une piste en réponse aux questionnements de l'article [Picault and Sicard, 2020].

Une autre difficulté est donc d'appliquer les concepts des SMA et des modèles d'agent selon une approche DOD dans le but de gagner en performance lors de la simulation d'agents virtuels.

Selon notre état de l'art, section 2.2, il n'existe pas de plateformes dans le domaine SMA proposant une approche DOD des SMA, avec cet aspect multi-paradigmes. Au contraire, elles adoptent une approche Programmation orienté-objet (POO) qui complexifie les performances et l'aspect multi-paradigmes, notamment par le couplage des données et de leur logique. Nous pensons qu'une approche DOD serait plus pertinente.

4.1.2 Objectifs

Ainsi, notre objectif est de simuler un grand nombre d'agents virtuels de complexités variées et adaptatives dans le contexte d'une simulation interactive, grâce à l'application du méta-modèle OPACK et des principes DOD.

Le méta-modèle OPACK est une approche pour obtenir des simulations multi-paradigmes grâce à la dissociation des informations de l'agent (**P**, **A**, **C** et **K**) de sa dynamique (**O**). Nous ne posons aucune contrainte sur le paradigme utilisé pour dicter la dynamique. Notre méta-modèle est composable.

4.1.3 Organisation

C'est pourquoi dans ce chapitre, nous présentons une implémentation d'un outil de simulation d'agents virtuels suivant les principes du méta-modèle OPACK et de DOD.

Tout d'abord, nous commençons par introduire des principes de DOD, avec des exemples pour illustrer les gains potentiels. Ensuite, nous détaillons comment ces principes ont été appliqués au méta-modèle OPACK. Puis, nous finissons par la présentation de quelques fonctionnalités de cette implémentation.

Nous présentons dans la section 6.5 du chapitre 6 des comparaisons de performances de notre outil vis-à-vis d'autres outils existants dans la communauté SMA.

4.2 Avant-propos

Lors de notre présentation de DOD, nous allons illustrer quelques principes qui amènent à des gains de performances comparés à une approche classique. Dans ce chapitre, nous considérons dans le terme performance, uniquement le temps d'exécution. Nous supposons que la qualité de la solution reste inchangée.

4.2.1 DOD : Ressources utilisés

La majorité de nos explications et informations proviennent du livre de Richard Fabian [Fabian, 2018] et du livre en cours de rédaction par Sergey Slotin, disponible en ligne : « Algorithms for Modern Hardware [3] ».

Nous vous recommandons aussi de visionner les conférences suivantes à ce sujet :

- **de Mike Akton**, lors de CppCon 2014 : Data-Oriented Design and C++ [4]
- **de Stoyan Nikolov**, lors de CppCon 2018 : OOP Is Dead, Long Live Data-oriented Design [5]
- **de Sergey Slotin**, lors de Meeting C++ 2022 : CPU Cache effects [6]

3. https://en.algorithmica.org/hpc/
4. https://youtu.be/rX0ItVEVjHc
5. https://youtu.be/yy8jQgmhbAU
6. https://youtu.be/mQWuX_KgH00

4.2.2 Enjeux de la mémoire

La complexité algorithme ne suffit pas à estimer la performance d'un algorithme [Slotin, wip]. Le temps d'exécution d'un programme dépend toujours du temps pour réaliser l'ensemble des instructions. Mais, le temps d'exécution d'une instruction n'est pas juste le temps pour réaliser l'opération, mais également le temps requis pour récupérer les données nécessaires. Dès lors, la rapidité d'accès à la mémoire peut être un facteur limitant les performances.

En 2002, [Carvalho, 2002] soulève l'écart de performances entre les processeurs et la mémoire qui s'intensifie de plus en plus : l'un s'emploie à aller plus vite, tandis que l'autre cherche à augmenter ses capacités de stockage. La conséquence est que la latence et la bande passante de la mémoire peut ne pas suivre la rapidité du Central Processing Unit (CPU) et donc ralentir l'exécution.

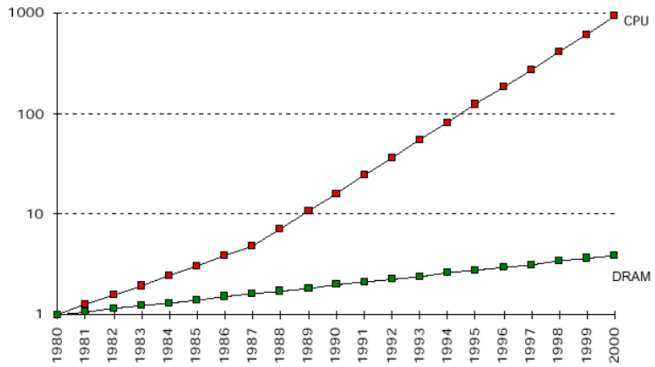


FIGURE 4.1 – Évolution de l'écart de performance entre la vitesse des processeurs et de la mémoire au fil des années (tiré de [Carvalho, 2002].)

Plus précisément, lorsqu'une instruction est exécutée, *e.g.* une addition, il faut que les données de l'opération soient à disposition, *e.g.* les opérandes de l'addition. Mais, en fonction de là où se trouve la donnée, le temps requis peut considérablement s'allonger. Pour représenter la différence de vitesse, [Carvalho, 2002] emploie l'exemple suivant :

Citation 15 - Exemple illustratif de la différence de vitesse entre la mémoire et le processeur

« Prenons l'exemple d'un ordinateur hypothétique avec un processeur qui fonctionne à 800 MHz (un Pentium III, par exemple), connecté à une mémoire par un bus de 100 MHz (SDRAM PC-100). Considérons que ce processeur manipule 800 millions d'éléments (instructions et/ou données) par seconde et que la mémoire atteint un débit (envoi ou réception) de 100 millions d'éléments par seconde. Dans cet ordinateur, pour chaque accès à la mémoire, 8 cycles d'horloge du processeur se sont écoulés. De cette façon, 7 sur 8 cycles d'horloges sont gaspillés à attendre des éléments. Cela représente un coût très élevé. »

Carvalho, C. (2002). The gap between processor and memory speeds

Si nous rentrons un peu plus dans les détails de l'architecture d'un ordinateur, voici la différence de vitesse entre les différentes mémoires d'un ordinateur en nombre de cycles CPU, selon Sergey Slotin [Slotin, wip] :

- Registre : 1 cycle
- L1 : 5 cycles
- L2 : 10 cycles
- L3 : 40 cycles
- RAM : 100 ns (200 cycles)
- SSD : 0,1 ms (10^5 cycles)
- HDD : 10 ms (10^7 cycles)
- Internet : 100 ms (10^8 cycles)

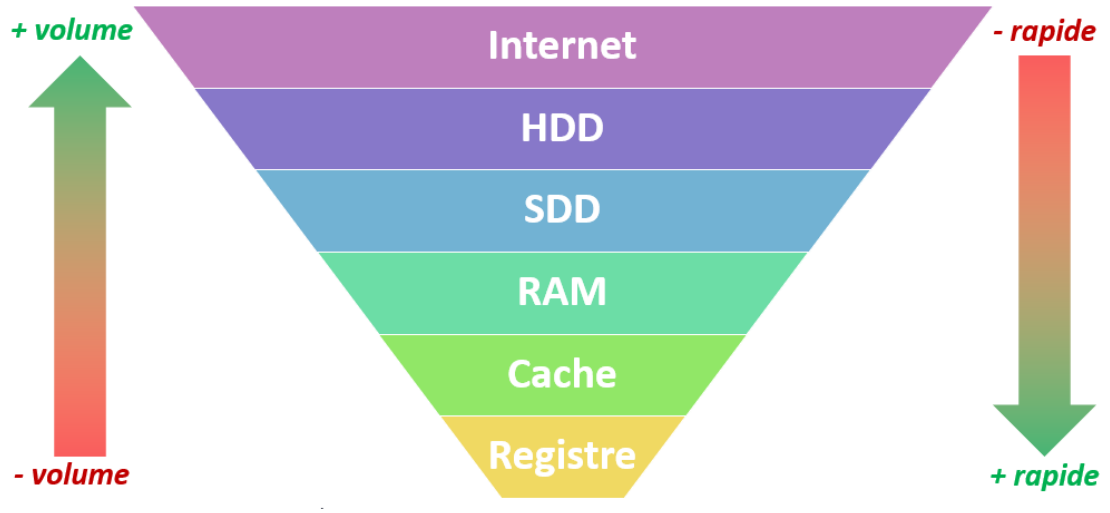


FIGURE 4.2 – Différents types de mémoire. Plus une mémoire est rapide, plus elle est coûteuse (financièrement), moins il y a de volumes (espace de stockage).

Selon ces données, nous aimerions idéalement que nos données soient situées dans le registre ou dans le cache (composé de L1, L2 et L3). Cependant, plus une mémoire est performante, plus l'espace disponible est réduit. La difficulté est donc de bien organiser ses données pour qu'elles puissent être stockées dans le cache.

Nous allons voir en quoi DOD cherche à favoriser une bonne utilisation du cache.

4.3 Design orientée-données

Le Design Orientée-Données, *Data-Oriented Design* en anglais (DOD), est un concept qui existe depuis plusieurs décennies [Fabian, 2018], mais a été officiellement nommé en

tant que tel par Noel Llopis en 2009 [Llopis, 2009]. Richard Fabian résume DOD de la manière suivante :

Citation 16 - Design orientée-données

« Façon de penser et de présenter vos données et de prendre des décisions concernant votre architecture. »

Fabian, R. (2018). Data-oriented design. *framework*, 21:1–7

Autrement dit, un aspect important de DOD est de bien réfléchir aux données : comment sont-elles liées ? Quels sont les traitements appliqués ? *etc.* Le but est d'organiser ses données pour utiliser efficacement les ressources à notre disposition. Les ressources sont le matériel informatique où le code sera exécuté. Il est important de savoir quel est notre matériel cible pour adapter l'architecture : systèmes embarqués, téléphones, consoles, ordinateur personnel, supercalculateur, *etc.* . D'où la phrase suivante de Richard Fabian : « Connaissez vos données ; connaissez votre matériel » [Fabian, 2018]

DOD est un sujet vaste qui ne s'intéresse pas uniquement à la performance, mais également à la maintenance, réutilisabilité, *etc.* . La présentation de DOD en détail excède nos compétences et la portée de la thèse. Cependant, nous allons présenter plusieurs aspects pratiques et les gains envisageables, comparé à une approche classique, comme la POO.

4.3.1 Principe de localité

Une manière d'estimer qualitativement la performance d'un algorithme vis-à-vis du cache est d'employer les termes suivants :

Citation 17 - Principe de localité

*« La **localité spatiale** fait référence à l'utilisation d'éléments relativement proches les uns des autres en termes d'emplacement mémoire, de sorte qu'ils sont probablement récupérés dans le même bloc mémoire.*
*La **localité temporelle** fait référence à l'accès répété aux mêmes données dans un laps de temps relativement court, de sorte que les données restent probablement en mémoire cache entre les requêtes.*
En d'autres termes, on parle de localité temporelle lorsqu'il est probable que ce même emplacement de mémoire sera bientôt demandé à nouveau, tandis que la localité spatiale est lorsqu'il est probable qu'un emplacement proche sera demandé juste après. »

Slotin, S. (2022-wip)). Algorithms for modern hardware. *Open-access online book*

Un algorithme avec une bonne localité spatiale et temporelle permet au CPU de prédire plus efficacement quelles données seront utilisées. Elles seront ainsi précharger

dans le cache.

Sergey Slotin propose un cas d'études sur la multiplication de matrices [7] [Slotin, wip]. Par exemple, le simple fait de transposer la seconde matrice permet d'obtenir un gain de performance de 35%, comparé à l'approche naïve. À elle seule, cette technique donne peu de gains ; mais elle possède l'avantage de faciliter la vectorisation (ce qui améliore le principe de localité). Avec la vectorisation, il obtient cette fois un gain de 537%.

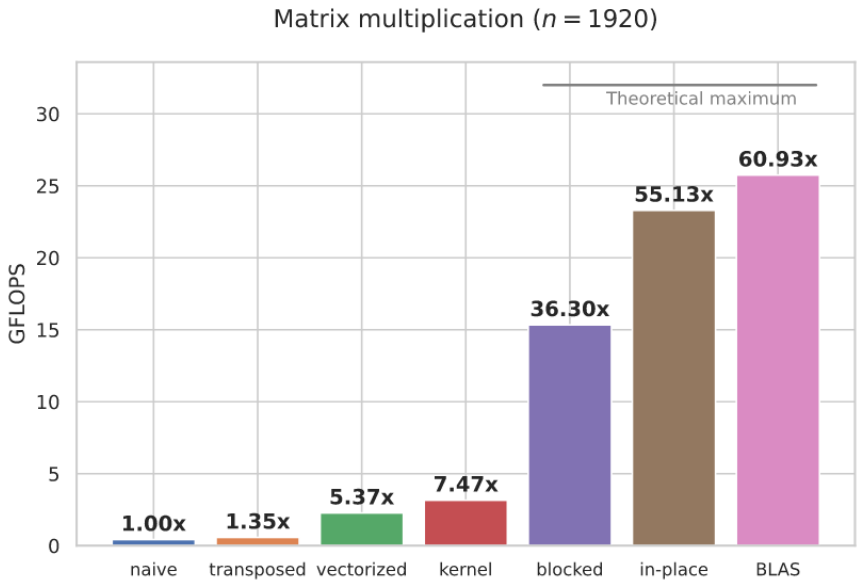


FIGURE 4.3 – Gain de performances lors d'un scénario de la multiplication de matrices, issu de [Slotin, wip], selon différentes techniques.

4.3.2 Toutes les opérations ne se valent pas

Ensuite, un aspect important qui n'est pas considéré par la complexité asymptotique est que toutes les opérations d'un CPU ne se valent pas.

Prenons, par exemple, le cas d'une instruction avec une condition comme if. Nous avons vu précédemment que le CPU essaye de prédire les données pour les précharger dans le cache. Mais, le CPU va aussi prédire la suite des instructions et leurs résultats. Or, dans un programme, le flux d'exécution n'est pas nécessairement linéaire. Il peut y avoir des embranchements. Avec un if, deux branches sont envisageables : si la condition est vérifiée ou non. Le CPU va prédire quelle branche est la plus probable pour la charger dans le cache. S'il prédit correctement, alors tout va bien. S'il ne prédit pas correctement, alors nous avons un *cache miss*. Il doit oublier sa prédiction et re-charger l'autre branche. Ce qui a un coût non négligeable lorsque les *cache miss* s'accumulent lors d'itération.

7. https://en.algorithmica.org/hpc/algorithms/matmul/

Pour illustrer le coût de cette mauvaise prédiction, regardons l'infographie 4.4 issue de Ithare [8] . Nous pouvons remarquer qu'un *cache miss* est 10x plus coûteux que lorsque la bonne branche est prédite. Dès lors, si nous accumulons ce genre de « raté » dans nos cycles, les performances sont affectées.

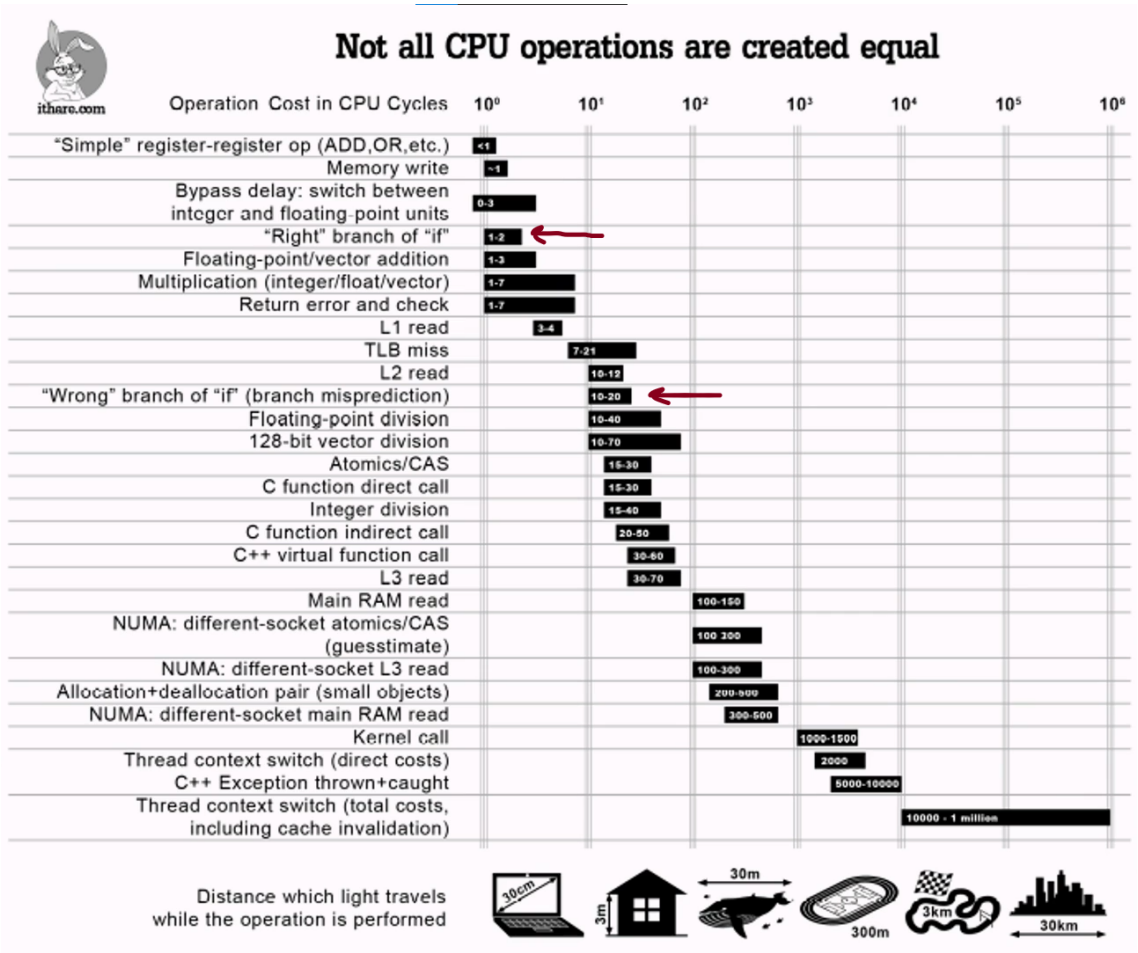


FIGURE 4.4 – Infographie issue de ITHARE qui compare le coût en cycles des opérations du CPU. Nous retrouvons les coûts indiqués par Sergey Slotlin sur la vitesse d'accès des différentes mémoires. Surtout, nous pouvons voir qu'une condition mal-évalué lors d'une branche d'un *if*, engendre un coût important avec un ordre de grandeur de différence de cycles.

Exemple : traitement existentiel Une manière pratique de limiter les embranchements est de réaliser un traitement existentiel. Pour illustrer ce principe, nous allons considérer que nous itérons sur N agents virtuels. Supposons ensuite que nos agents incrémentent un compteur, uniquement si une de leurs conditions est vérifiée.

Une manière de le traduire naïvement en orientée-objet serait d'ajouter un booléen

8. http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/

et d'incrémenter uniquement lorsqu'il est vrai. Nous obtiendrons alors le pseudo-code suivant (*cf.* algorithme 2) :

Algorithm 2: Pseudo-code de la mise à jour des agents virtuels selon une approche orientée-objet

```
Input number of agents N;
struct {
    Integer counter;
    Boolean need_update;
    update() {
    if a.need_update then
        a.counter+ = 1;
    }
} Agent;
Agt[N];
for a ∈ Agt do
    a.update();
```

Nous voyons bien ici que deux cas sont possibles. Nous avons donc une probabilité d'avoir un « cache miss » à chaque itération. Qui plus est, nous avons rajouté une variable à notre agent, ce qui réduit notre efficacité vis-à-vis du cache. Notre agent prend plus de place.

Une autre manière de faire pour ne pas avoir à considérer cette branche est de séparer en deux ensembles les agents qui ont besoin d'une mise à jour des autres. Non seulement nous itérons sur un ensemble réduit, mais en plus, nous supprimons la vérification d'une branche. Le code est ainsi plus linéaire que l'autre version. Toutefois, il faut désormais maintenir et déplacer les agents accordement, ce qui a un coût non négligeable. C'est pourquoi d'autres approches comme Entity-Component-System (ECS) [Härkönen, 2019] sont plus adaptés et performantes dans ce genre de traitement.

4.4 OPACK et DOD

Dans cette section, nous allons formaliser un SMA en suivant les principes DOD. Pour nous aider à suivre ses principes, nous avons choisi d'utiliser le modèle de programmation ECS, qui facilite l'application de ces principes.

4.4.1 ECS

ECS pour Entité, Composant et Système, est un design de programmation séparant les données des comportements, pour notamment faciliter la réutilisatiblité et la composabilité du code. Les données sont stockées d'une manière favorable à une bonne

utilisation du cache. Nous avons vu que c'est un aspect important pour la performance. Nous illustrons ECS avec le diagramme 4.5 issu du manuel du moteur de jeu Unity [9] .

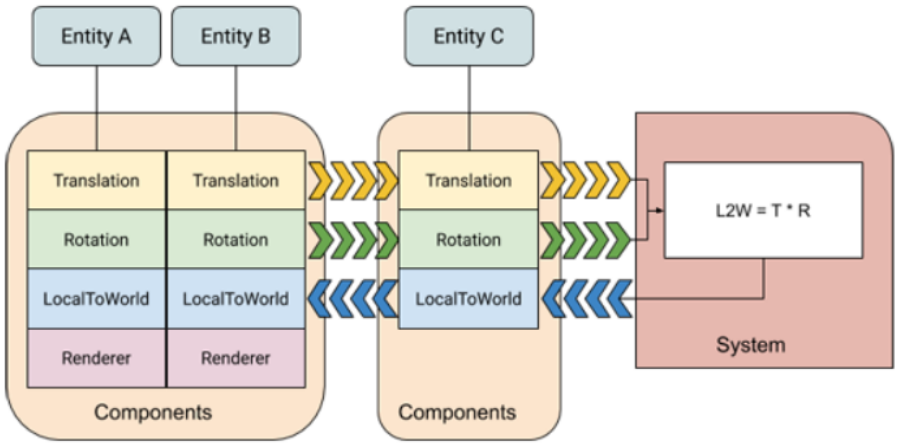


FIGURE 4.5 – Illustration des concepts de ECS. Le système itère uniquement sur entités qui possèdent les composants : TRANSLATION et ROTATION ; puis écrit le résultat dans un composant LOCALTOWORLD.

Voici la signification des trois concepts :

- **Une entité** est un identifiant auxquels sont associées des composants. Sur le schéma 4.5, il s'agit de : $EntityA$, $EntityB$ et $EntityC$.
- **Un composant** est une structure de données quelconques. Sur le schéma 4.5, il s'agit de : $Translation$, $Rotation$, $LocalToWorld$ et $Renderer$.
- **Un système** est une fonction qui s'applique sur un sous-ensemble d'entités dont les composants correspondent à ce que le système requiert. Sur le schéma 4.5, il s'agit de la fonction : $L2W = T * R$. Elle ne s'applique uniquement sur les entités qui ont les composants $Translation$ et $Rotation$. Ensuite, la composabilité permet au système de stocker le résultat dans un composant $LocalToWorld$.

Toutefois, une limite de son approche est son manque d'expressivité pour représenter des relations, comme dans un modèle entité-association [10] . Ensuite, nous ne pouvons pas associer plusieurs fois le même composant à une même entité. C'est pourquoi, nous considérons une extension de cette approche : ECS-R.

ECS-R

L'approche ECS-R de Sander Mertens qui introduit la notation de relation [11] . Une relation est un composant permettant de relier soit deux entités entre elles, soit plusieurs fois le même composant à une même entité. Il est désormais possible d'associer

9. https://docs.unity3d.com/Packages/com.unity.entities@0.1/manual/ecs_core.html
10. https://fr.wikipedia.org/wiki/Mod%C3%A8le_entit%C3%A9-association
11. https://ajmmertens.medium.com/building-games-in-ecs-with-entity-relationsh

plusieurs fois un même composant, grâce à différentes relations. Cette fonctionnalité va nous permettre d'appliquer en partie le méta-modèle OPACK au formalisme ECS-R.

4.4.2 Application

Dans OPACK, l'ensemble des concepts **O**, **P**, **A**, **C** et **K** sont représentés par des identifiants associables aux travers de caractéristiques de différentes arités.

Pour transposer notre formalisme, il suffit d'indiquer les choses suivantes :

— Une entité est un identifiant.
— Les caractéristiques d'arité 0 sont des composants.
— Les caractéristiques d'arité 1 sont des relations.
— Les caractéristiques d'arité $1+$ sont représentées par une succession de relations.

Désormais, nous allons présenter la librairie et ses fonctionnalités.

4.5 Fonctionnalités

Lors de cette section, nous présentons plusieurs fonctionnalités de la bibliothèque. Nous n'allons pas décrire toutes les manières d'obtenir un même résultat. Nous verrons ici la manière la plus simple et rapide. Ensuite, pour illustrer les fonctionnalités, nous allons reprendre l'exemple que nous avons utilisé lors de la présentation du méta-modèle (*cf.* figure 4.6 et section 3.1.4).

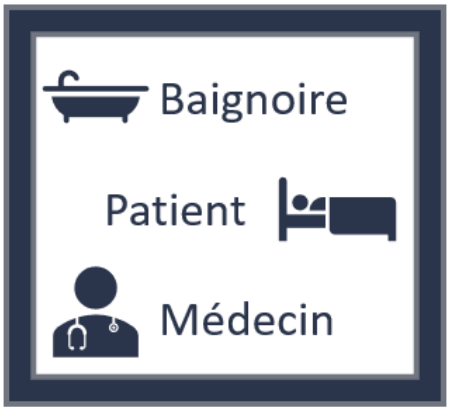


FIGURE 4.6 – Exemple illustratif des éléments composant le monde que nous allons chercher à simuler.

4.5.1 Squelette d'utilisation

Tout d'abord, voici le squelette minimal de toute utilisation de la bibliothèque :

Code 1: Squelette de notre bibliothèque.

```cpp
#include <opack/core.hpp> // Core header to use the library

int main()
{
    // 1. Create an empty world.
    auto world = opack::create_world();
    // 2. That's it.
}
```

`#include <opack/core.hpp>` inclue les fonctionnalités de base pour utiliser la libraire. `auto world = opack::create_world()` est le point d'entrée de la libraire. Il s'agit en quelque sorte du Ω de notre méta-modèle.

Désormais, nous allons voir les fonctionnalités à partir de ce squelette.

4.5.2 Entité

La première chose que nous allons faire est de créer nos trois entités : un agent et deux artefacts. Pour se faire, nous utilisons la notion de *Prefab* de Flecs [12] . Succintement, un *prefab* est une entité-modèle que nous pouvons instancier. Notre bibliothèque propose la fonction `opack::spawn<MyPrefab>(world)` pour faciliter cette mécanique. Plusieurs prefabs sont à disposition, notamment `Agent` et `Artefact` pour rapidement créer des agents ou des artefacts. Ainsi, pour obtenir nos trois entités, nous avons le code suivant :

Code 2: Définition des entités

```cpp
auto doctor = opack::spawn<Agent>(world, "Doctor");
auto bathtub = opack::spawn<Artefact>(world, "Bathtub");
auto patient = opack::spawn<Artefact>(world, "Patient");
```

4.5.3 Caractéristiques

Ensuite, nous allons rajouter les caractéristiques : `Introvert`, `IsCoughing` et `Stress`. Ici, nous allons simplement considérer qu'il s'agit de deux tags et d'une valeur continue :

Code 3: Définition des composants

```cpp
struct Introvert {};
struct IsCoughing {};
struct Stress {float value{0.0f}; };
```

Il est possible de définir des composants exclusifs, *e.g* $IsCoughing$ et $IsNotCoughing$, pour qu'une entité ne puisse pas posséder certains composants simultanément.

12. https://github.com/SanderMertens/flecs

Comme les agents et les artefacts sont des entités, ils sont composables. Autrement, nous pouvons associer n'importe quelle structure de données à ces derniers. Nous obtenons donc l'association des caractéristiques de l'exemple de la manière suivante :

Code 4: Composition des entités

```
doctor.add<Introvert>();
patient.add<IsCoughing>();
doctor.set<Stress>({10.f});
```

4.5.4 Perception

La perception des agents est réalisée aux travers de sens. La première étape consiste donc à définir un sens et indiquer ce qu'il permet de percevoir.

Code 5: Mécanisme de perception : sens

```
OPACK_SENSE(Hearing); // Define a struct called Hearing

opack::init<Hearing>(world) // Mandatory to create entity associated to "Hearing"
and compose it accordlingy.

opack::add_sense<Hearing, Agent>(world) // Needs to be added before instanciation,
otherwise it must be manually added to the entity.
opack::perceive<Hearing, IsCoughing>(world)
```

Nous disposons désormais d'un sens qui permet de percevoir si une entité tousse ou non. Ensuite, nous allons indiquer qui perçoit qui. Ici, nous le faisons manuellement. Normalement, un système est responsable pour le faire automatiquement, e.g toutes les entités à moins d'une certaine distance.

Code 6: Mécanisme de perception

```
opack::perceive<Hearing>(doctor, patient); // Doctor is now perceiving patient
throught its hearing
opack::perceive<Hearing>(doctor, bathtub); // and the bathtub

// Let's interoggate our perception api
auto p_doctor = opack::perception(doctor);
auto p_patient = opack::perception(patient);

p_doctor.perceive<Hearing>(patient); // returns true;
p_doctor.perceive<Hearing>(bathtub); // returns true;
p_patient.perceive<Hearing>(doctor); // returns false;

p_doctor.value_of<Hearing, Stress>(patient); // returns nullptr since patient has no
 stress and hearing isn't capable of perceiving it.

p_doctor.each<opack::Artefact>([](opack::Entity subject){
    // will be called once for each artefact, so bathtub and patient
});

p_doctor.each<IsCoughing>([](opack::Entity subject){
    // will be called only once for artefact patient
});
```

4.5.5 Action

Les agents réalisent des actions par le biais d'effecteurs. La première étape consiste donc à définir un effecteur et les actions réalisables.

Code 7: Mécanisme de perception : effecteur et action

```
OPACK_ACTUATOR(Hands);
opack::init<Hands>(world)
opack::add_actuator<Hands, Agent>(world) // Needs to be added before instanciation,
otherwise it must be manually added to the entity.

OPACK_ACTION(Wash);
opack::init<Wash>(world)
    .require<Hands>()
    .duration(60.0f)
    .on_action_begin<Wash>([](opack::entity action){/*..*/}
    .on_action_update<Wash>([](opack::entity action, float dt){/*..*/}
    .on_action_end<Wash>([](opack::entity action){/*..*/}
    .on_action_abort<Wash>([](opack::entity action){/*..*/}
```

Nous avons défini un effecteur dénommé `Hands` que tous nos agents posséderont. Ensuite, nous avons défini une action `Wash` qui requiert cet effecteur et dure par défaut 60 secondes. Enfin, la dernière étape consiste à réaliser une action :

Code 8: Mécanisme d'action : agissement

```
struct With {};
auto action = opack::spawn<Wash>(world);
action.add<With>(bathtub);
opack::act(doctor, action);
```

Désormais, l'agent `Doctor` va réaliser une instance de l'action `Wash` pendant 60 secondes. Une fois ce délai terminé, l'instance sera détruite, mais pas avant d'avoir appelé les *callbacks* `on_action_[begin, update, end]` définis selon l'environnement. Ce délai peut tout à fait être modifié, notamment lors des opérations.

4.5.6 Opération

Nous allons désormais créer un flux d'opérations, soit celui de l'exemple, qui est composé, pour le moment, de trois opérations : $SuitableActions$, $ActionSelection$ et Act.

Code 9: Mécanisme d'opération : flux et opérations

```
// 1. Define flow and build it
OPACK_FLOW(Replicants);
opack::FlowBuilder<Replicants>(world).interval(1.0f); // Called every seconds
instead of every cycle

// 2. Define operations
struct SuitableActions : opack::operations::Union<opack::Action_t> {}; // Union
strategy
struct ActionSelection : opack::operations::SelectionByIGraph<SuitableActions> {};
// Influence Graph strategy
struct Act : opack::operations::All<opack::df<ActionSelection, opack::Action_t>> {};
 // All strategy

// 3. Add them to the flow
opack::operation<Replicants, SuitableActions, ActionSelection, Act>(world)
```

Nous avons construit un flux d'opération `Replicants`, avec les trois opérations :

1. **SuitableActions** : Détermine les actions envisageables grâce à la stratégie *Union*. C'est-à-dire que l'ensemble des actions, retournées par les fonctions-impacts actives.
2. **ActionSelection** : Sélectionne l'action à réaliser à partir des actions envisageables et des comportements actifs. La stratégie employée est celle du *graphe d'influences* que nous avons présenté précédemment.
3. **Act** : Exécute l'action. La stratégie employée est *All*. C'est-à-dire que l'ensemble des fonctions-impacts actives vont s'exécuter. Il n'y a pas de valeur de retour.

Désormais, nous pouvons définir les fonctions-impacts par défaut :

Code 10: Mécanisme d'opération : impact par défaut

```
// By default, Wash action is always suitable.
opack::default_impact<SuitableActions>(world, [](opack::Entity e, auto& inputs)
{
    SuitableActions::iterator(inputs) = opack::entity<Wash>(world);
    return opack::make_outputs<SuitableActions>();
});

// By default, selected action is always acted.
opack::default_impact<Act>(world, [](opack::Entity agent, auto& inputs)
{
    auto action = std::get<opack::df<ActionSelection, opack::Action_t>&>(inputs).
value;
    opack::act(agent, action);
    return opack::make_outputs<Act>();
});
```

Nous avons juste indiquer que par défaut, l'action « Wash » est envisageable et que l'action selectionnée est exécutée.

4.5.7 Comportements

Enfin, nous allons ajouter un comportement de stress qui va juste impacter l'exécution de l'action. Si l'agent est stressé, alors il peut paniquer et/ou trembler. La différence est que nous allons considérer, pour l'exemple, que « paniquer » est une action et que « trembler » est un modificateur d'état pour une action.

Code 11: Mécanisme d'opération : ajout d'actions

```
struct CanShake {}
struct Shake {}
OPACK_ACTION(Panic);
opack::init<Panic>().requires<Hands>();
opack::entity<Wash>().add<CanShake>();
```

Donc, nous avons défini une nouvelle action `Panic`. Nous avons aussi ajouté une caractéristique ou un composant pour indiquer si une action peut trembler. Nous l'avons ajoutée à l'action `Wash`. Cela nous sera utile plus tard. Ensuite, définissons le comportement de stress :

Code 12: Mécanisme d'opération : ajout d'un comportement

```
OPACK_BEHAVIOUR(BStress) //1. Identify a behaviour with a type
opack::behaviour<BStress>(world, opack::with<Stress>) //2. Behaviour is only active
for agent with a "Stress" component.
```

Nous avons défini un comportement dénommé `BStress`, actif uniquement lorsque notre agent est stressé. Ensuite, nous allons définir les impacts de ce comportement sur notre modèle d'agent, soit sur les opérations :

Code 13: Mécanisme d'opération : impact sur les actions envisageables

```
opack::default_impact<SuitableActions>(world, [](opack::Entity e, auto& inputs)
{
    SuitableActions::iterator(inputs) = opack::entity<Panic>(world);
    return opack::make_outputs<SuitableActions>();
});
```

L'opération `SuitableActions` est impactée en ajoutant l'action `Panic` dans les actions envisageables.

Code 14: Mécanisme d'opération : impact sur la sélection

```
opack::impact<ActionSelection, BStress>(world, [](opack::Entity agent,
ActionSelection::inputs& inputs)
{
    ActionSelection::each(inputs, [](auto& graph, const auto& action)
        {
            if (opack::is_a<Panic>(action))
                graph.positive_influence(action);
        });
    return opack::make_outputs<ActionSelection>();
});
```

L'opération `ActionSelection` est impactée en ajoutant une influence positive lorsqu'une action est considérée comme une action `Panic`. Tout autre raisonnement serait valide. Tout dépend de l'environnement considéré et des informations à disposition.

Code 15: Mécanisme d'opération : ajout des impacts

```
opack::default_impact<Act>(world, [](opack::Entity agent, auto& inputs)
{
    auto action = std::get<opack::df<ActionSelection, opack::Action_t>&>(inputs).
value;
    if(action.has<CanShake>())
        action.add<Shake>;
    if(action.has<Duration>())
        action.get_mut<Duration>->value *= 1.2;
    return opack::make_outputs<Act>();
});
```

Enfin, pour l'opération `Act`, le comportement de stress l'impacte en faisant trembler les actions compatible, soit celles possédant la caractéristique `Shake` définit précédemment. Aussi, la durée de réalisation est allongée par un facteur arbitraire.

Rien n'empêche que ce facteur arbitraire soit un paramètre calculé selon le profil de l'agent grâce à une caractéristique.

4.5.8 Mini-bilan

Au travers de cet exemple simple, nous avons vu comment grâce à la librairie OPACK, nous avons pu simuler notre monde avec trois entités et avec un flux d'opérations REPLICANTS (sans arbre d'activité pour simplifier, mais rien n'empêche par la suite d'ajouter une fonction-impact pour ajouter le parcours de l'arbre dans les actions envisageables). Nous avons ensuite ajouté un comportement de stress, impactant les trois opérations du flux. Le coût de la généricité et modularité est que le modélisateur doit définir beaucoup de choses. Il nous reste à voir une dernière fonctionnalité : l'exécution de cycles, soit le contrôle de la simulation.

4.5.9 Contrôle de la simulation

La bibliothèque propose plusieurs manières de contrôler l'évolution de la simulation :

- un mode autonome où la simulation tente d'atteindre un nombre de cycles par secondes ;
- un mode manuel où l'utilisateur contrôle les cycles.

Mode autonome

Tout d'abord, l'utilisateur peut indiquer un nombre de cycles par secondes souhaités. Par défaut, la librairie essaye de s'exécuter autant de fois que possible :

Code 16: Cycle par secondes souhaité en mode autonome.

```
// Indicates how many times it should tried to run per second
opack::target_fps(world, 30);   // --> There will be no more than
                                // 30 cycle per second.
```

Ensuite, nous disposons de deux modes automatiques : un mode pour tourner avec une application web, ce qui permet d'inspecter l'état du monde (*cf.* figure 4.7) ou sans pour plus de performances.

Code 17: Deux modes automatiques

```
// -- 1. Run with web app activated
opack::run_with_webapp(world);
// -- 2. Run without web app
opack::run(world);
```

Enfin, pour arrêter la simulation, il suffit de réaliser l'appel suivant dans un système :

Code 18: Arrêt du mode automatique

```
// Stop running world
opack::stop(world);
```

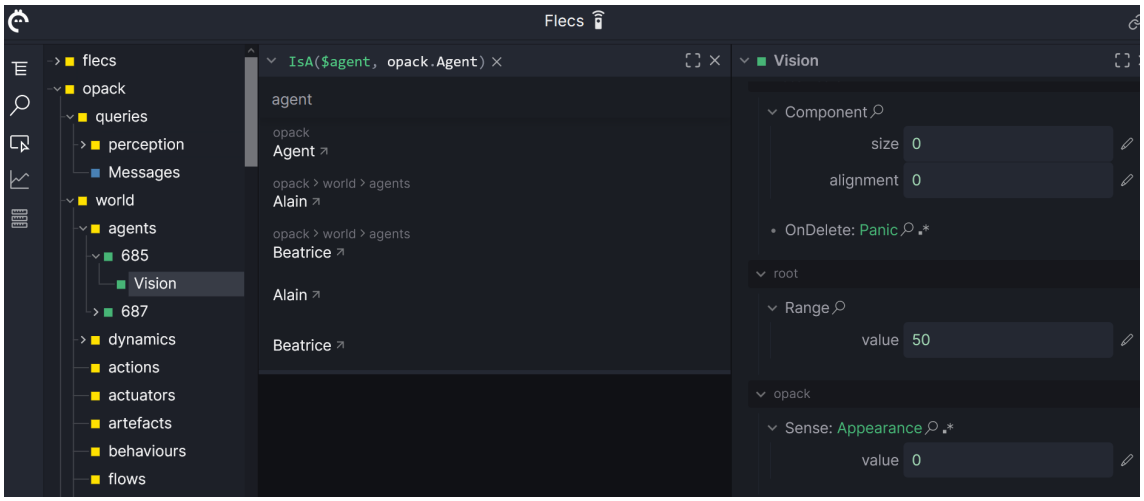


FIGURE 4.7 – Capture d'écran de l'inspecteur web fourni par flecs.

Mode manuel

En mode manuel, l'utilisateur doit indiquer quand le passage d'un cycle doit être réalisé. La raison de ce mode est nous voulons être intégrable dans d'autres applications. Nous ne voulons donc pas prendre le contrôle du flux de l'exécution. Nous développons une bibliothèque plutôt qu'une plate-forme, bien que nous fournissons un mode automatique.

Code 19: Modification de l'échelle du temps.

```
// Indicates how fast time flies.
// 1.0 -> 1 second in real life means 1 second in simulation time
// 2.0 -> 1 second in real life means 2 second in simulation time
// 0.5 -> 1 second in real life means 0.5 second in simulation time
opack::time_scale(world, 2.0f);
```

Code 20: Quelques fonctions pour interroger l'état de la simulation.

```
// 4. Returns number of elapsed tick.
opack::tick(world);

// 5. Returns delta time or elapsed time between two ticks.
opack::delta_time(world);

// 6. Returns total elapsed time in simulation (so it consider time scale).
opack::time(world);
```

Code 21: Plusieurs manières de réaliser des cycles.

```
// 7. Advance simulation by
// ---- 1. one step with automatic delta time computation
opack::step(world);

// ---- 2. one step with specified delta time (no time scale applied)
opack::step(world, 2.0f);   // 2 seconds have passed in simulation time

// ---- 3. n steps with automatic delta time
opack::step_n(world, 10);   // 10 cycle will happen.
```

```
// ---- 4. n steps with specified delta time (no time scale applied)
opack::step_n(world, 10, 2.0f); // 10 cycle will happen with 2 seconds passed
between each cycle.
```

4.5.10 Intégration

Une autre manière est d'intégrer notre bibliothèque est envisageable pour les applications employant Flecs [13] . En effet, nous avons pensé notre bibliothèque comme un module de Flecs. La raison est que Flecs est utilisé pour la création de jeu et de moteur de jeu. Cette bibliothèque est compatible avec l'ajout de moteur graphique, physique, *etc.* . [14]

Par ailleurs, c'est ce que nous avions commencé dans un autre projet, où nous avions ajouté une api graphique (Vulkan [16]) et une bibliothèque d'interface graphique ImGui [17] . L'idée était de développer notre propre inspecteur. Nous avions notamment la possibilité d'inspecter les flux d'opérations (cf. figure 4.8). Toutefois, le développement du moteur a été stoppé : la bibliothèque évoluait trop rapidement pour maintenir le moteur a côté.

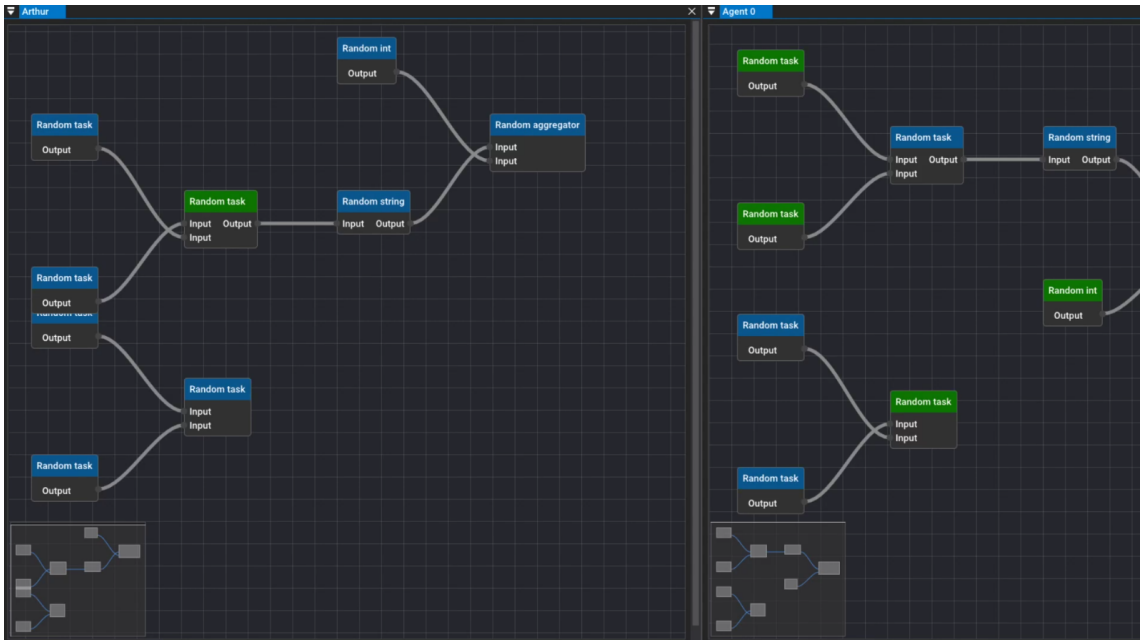


Figure 4.8 – Capture d'écran de notre inspecteur pour visualiser les flux d'opérations et les opérations en cours.

13. https://github.com/SanderMertens/flecs
14. Par exemple, TheForge [15] est un moteur graphique intégrant Flecs.
16. https://www.vulkan.org
17. https://github.com/ocornut/imgui

4.6 Bilan

Lors de ce chapitre, nous présentons l'implémentation d'une bibliothèque pour la simulation d'agents virtuels selon notre méta-modèle OPACK et suivant les principes DOD, notamment grâce à l'utilisation de ECS.

Notre outil propose des mécanismes pour une simulation multi-paradigmes. D'une part, nous dissocions les données du comportement (principe de ECS), contrairement aux approches actuelles qui suivent des principes de programmation orientée-objet. D'autre part, nous n'imposons pas le paradigme du modèle d'agent : le modélisateur est libre de définir le paradigme associé aux travers des flux d'opérations. Qui plus est, le modélisateur peut dynamiquement modifier les flux d'opérations, soit dynamiquement modifier le paradigme employé par les agents.

Ensuite, nous visons l'usage de cette bibliothèque dans un contexte de simulation interactive avec des ressources limitées. C'est un autre raison pour laquelle nous nous sommes orientés vers une approche DOD plutôt que de POO. Afin d'évaluer ce dernier point, nous avons réalisé des évaluations comparatives de performance de notre outil vis-à-vis d'autres plateformes de simulation multi-agents. Les résultats sont disponibles dans la section 6.5 du chapitre 6.

Avant cela, nous allons d'abord présenter le projet ORCHESTRAA qui finance la thèse et nos contributions au projet : nous avons appliqué le méta-modèle et cette bibliothèque pour simuler des postes d'un centre de commandement aérien.

Chapitre 5

Application : ORCHESTRAA

Contents

Cette thèse est financée par la Direction Générale de l'Armement (DGA) avec le projet ORCHESTRAA.

Ce projet s'inscrit dans le domaine de la **formation avancée au sein des centres de commandement et de contrôle d'opérations militaires ou des centres de gestion de crise civile**, et plus spécifiquement au niveau de l'entraînement du personnel. L'entraînement à la conduite d'opérations aériennes, avec comme partenaire et utilisateur final le Centre d'Analyse et de Simulation pour la Préparation aux Opérations Aériennes (CASPOA), a été choisi comme cas d'étude/d'inspiration.

Dans ce chapitre, nous commençons par présenter le projet et ses objectifs. Pour des raisons de confidentialité, nous ne présentons pas en détail le domaine métier, mais uniquement une introduction pour comprendre le scénario sur lequel nous travaillons. Puis, nous détaillerons nos contributions, avant de présenter les résultats obtenus dans le prochain chapitre.

5.1 Projet ORCHESTRAA

ORCHESTRAA a pour but de proposer une plate-forme d'entraînement en réalité virtuelle s'appuyant sur :

— une simulation d'**environnement virtuel** généraliste reproduisant le décor et les interactions au sein d'une salle de conduite,

— une **application pédagogique** multi-noeuds et multi-modale pour exécuter, piloter, superviser, diagnostiquer et faire évoluer des scénarios d'entraînement complexes,

— un moteur hybride d'**IA** pour **gérer à la fois plusieurs Personnages Virtuels Autonomes (PVA)**, ainsi qu'un autre moteur pour le **déroulement d'un méta-scénario pédagogique assistant les formateurs humains.**

Sur une durée de 36 mois, le projet a pour ambition de réaliser une première version de cette plate-forme d'entraînement avec un objectif d'évolutivité pour la recherche et l'exploitation réelle. Cette plate-forme sera utilisée pour implémenter avec l'équipe Thales LAS France/AOW et les formateurs du CASPOA quelques phases d'exercices d'entraînement ciblés puis les tester lors d'une phase d'expérimentations. En effet, une des principales difficultés auxquelles sont confrontées les équipes pédagogiques est le « coût » de la préparation/mise en œuvre des formations lors des phases de training. Par conséquent la problématique d'adéquation entre taux de remplissage des sessions et moyens mise en oeuvre se pose souvent. Une des clés pour optimiser les ressources (humaines notamment) est peut-être à rechercher dans l'usage de la réalité virtuelle.

Le but d'ORCHESTRAA est justement d'apporter de nouvelles modalités pédagogiques permettant, d'une part, d'améliorer l'entraînement grâce à l'usage des technologies de la Réalité Virtuelle (RV) et de l'IA qui offrent une capacité d'immersion et la possibilité de simuler une plus grande variété de scénarios, notamment au niveau de la coordination et des interactions entre personnes, et d'autre part, de diminuer la charge de travail et les ressources humaines nécessaires à l'élaboration et à la conduite des sessions par des solutions de pilotage et de gestion intuitives et simplifiées. Pour optimiser l'emploi des moyens de formation, les IA seront essentiellement non-scriptées et capables de s'adapter aux situations d'apprentissages souhaitées par les formateurs.

La problématique est la suivante : Comment combiner les apports de l'IA et de RV pour immerger en condition quasi-réelle et mieux former par l'entraînement les personnels aux diverses situations de commandement et de gestion de crises dans le secteur militaire, mais aussi dans le secteur civil.

Le projet ORCHESTRAA vise :

— La **recréation en virtuel d'un salle de conduite**, incluant une reproduction d'outils d'informations et de collaboration typiquement utilisés dans le civil et militaire : écrans vidéo, Chat, échanges vocaux (réalisé par Reviatech [1]).

— L'utilisation de PVA, dotés d'une IA et de modèles de comportements simulant les interactions humaines. Ces PVA peupleront la salle de conduite virtuelle, donneront la réplique à l'apprenant humain et dérouleront les trajectoires de formation (ma contribution).

— La conception d'outils de **scénarisation et de recommandations pédagogiques**, donnant une capacité de pilotage et d'inflexion des scénarios aux formateurs, incluant des outils de préparation amont, de pilotage temps-réel et de débriefing des sessions d'entraînement (réalisé par Romain Goutiere d'Heudiasyc).

1. https://reviatech.com/

— Un **système de méta-scénarisation assistant les formateurs dans la supervision** des sessions et leur révélant des opportunités d'action s'appuyant sur l'anticipation du déroulé futur et une étude probabiliste du déroulé du scénario réalisé VS scénario planifié (réalisé par Romain Goutiere d'Heudiasyc).

5.2 Introduction au commandement aérien

La salle de conduite, aussi appelée *Current Ops Room*, est une salle organisée en cellules où se déroule la conduite et le contrôle des opérations aériennes. Chaque cellule possède un champ de responsabilité précis, comme le suivi des opérations offensives, des opérations défensives, *etc.* ; en-voici quelques-unes :

— CURRENT OPS
— OFFENSIVE
— SUPPORT
— DEFENSIVE
— INTEL
— *etc.*

Ensuite, au sein de chaque cellule, les opérateurs occupent des postes avec une responsabilité précise (*cf.* figure 5.1). Globalement, l'objectif des opérateurs est de veiller au bon déroulement des missions aériennes ordonnées par un ordre d'opérations aériennes quotidien : Air Task Order (ATO). En cas de déviation du plan et d'imprévus, ils doivent proposer des solutions. Par exemple, la cellule en charge du Time Sensitive Target (TST) répond dans une fenêtre d'action très courte, aux besoins de ce processus lorsqu'une cible apparaît.

Le moyen de communication privilégié est un système de tchat, nommé JCHAT (pour des raisons d'organisation et de tracabilité notamment).

Le projet se focalise sur la formation d'une position, celle du Close Air Support Duty Officer (CAS-DO), jugée la plus pertinente selon les besoins du projet. Cette position fait partie de la cellule OFFENSIVE. Une des missions de cet îlot est de gérer les Troop In Contact (TIC). C'est-à-dire que, suite à une demande de troupes au sol alliées, la salle doit fournir un support aérien en réponse à la présence de troupes au sol ennemies. Le CAS-DO doit alors fournir une solution entravant le moins possible les opérations en cours, tout en répondant à l'urgence de la demande (les troupes ne peuvent pas attendre indéfiniment le support aérien, au risque de pertes). Pour prendre sa décision, le CAS-DO récolte les informations nécessaires aux travers des outils à sa disposition.

Un de ces outils est la Recognized Air Picture (RAP) (Recognized Air Picture) qui est une visualisation radar de l'espace aérien (*cf.* écran projeté sur le mur de la figure 5.1). ICC est l'outil principal actuel de programmation et de conduite des opérations aériennes. Il est constitué de modules permettant l'opérateur de disposer d'informations sur les aéronefs planifiés sur l'ATO du jour et sur l'état d'avancement des missions.

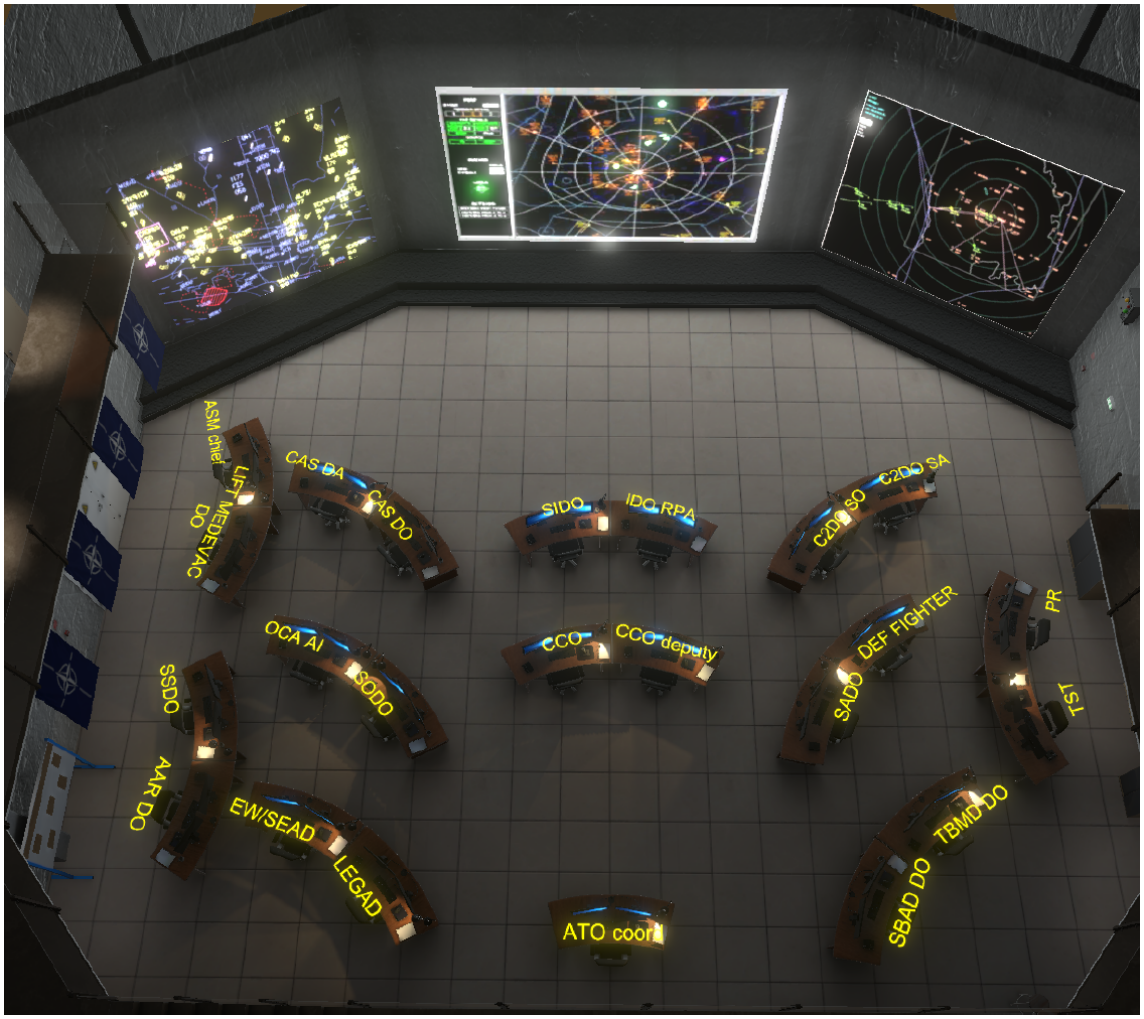


FIGURE 5.1 – Capture d'écran de l'environnement virtuel simulant un centre de commandement aérien, développé par Reviatech. Les textes jaunes désignent le nom des positions.

Enfin, un autre outil est le JCHAT que nous avons déjà mentionné. Cet outil est central dans notre contribution. En effet, nous nous focalisons sur les messages échangés dans cet outil, car il s'agira de l'unique moyen pour nos agents d'agir. Nous allons donc le détailler.

5.2.1 JCHAT : outil de messagerie

JCHAT est un système de messagerie instantanée utilisé dans les états-major du Joint Force Air Component (JFAC), structure de commandement et conduite des opérations aériennes, au sein de l'état-major interarmées et au sein du CASPOA (également nommé *NATO Air Opérations Centre of Excellence*).

Cet outil doit permettre de communiquer efficacement les informations sur l'évo-

lution de l'espace aérien, ainsi que de transmettre des ordres. Le JFAC n'est pas le seul utilisateur de cet outil. Des acteurs externes au JFAC sont également sur le chat. Ils ont besoin de se coordonner avec le JFAC, avec le Joint Task Force Head Quarter (JTF HQ) ou ont besoin d'utiliser l'espace aérien, ou encore de requérir le soutien de la composante aérienne.

Afin d'organiser les discussions, les messages échangés sont répartis dans des salles ; au moins une par cellule et quelques autres. En fonction de son poste, l'opérateur filtre des salles, pour surveiller uniquement celles pertinentes à son périmètre.

La lecture des messages est une tâche complexe, notamment pour les connaissances métiers requises, le nombre important d'acronymes et d'abréviations et un contexte de crise avec de nombreuses situations entre-mêlées. Le CASPOA nous a fourni plusieurs documents, dont certains étaient des échanges de messages réalisés sur JCHAT. Nous avons donc réalisé une première étude des messages, afin d'extraire les actions et les messages envoyés. Nous désignons les échanges de messages lors d'une session par *dump*. Nous avions à notre disposition plusieurs *dump* de sessions jouées par le CASPOA.

JChat Tools

Afin de faciliter la lecture des messages fournis par le CASPOA, nous avons développé un outil de lecture : JChat Tools. Les fonctionnalités sont les suivantes :

- Séparation par onglet de chaque dump (pour visualiser plusieurs dump simultanément).
- Séparation par onglet de chaque salle.
- Dictionnaire avec l'ensemble des acronymes :
  - avec une recherche par proximité,
  - ou affichant la signification des acronymes de la ligne du tchat sélectionnée.
- Création d'une base de données relationnelles Neo4j [2]

Un autre objectif était de permettre une analyse des messages grâce à la base de données relationnelles. Avec l'ajout d'annotation sur les messages, nous aurions pu, par exemple, récupérer l'ensemble des manières de confirmer une information. Cependant, l'outil n'a été exploité par aucun partenaire.

Formalisme proposé

Nous avions proposé de formaliser les messages sur un formalisme allégé de FIPA-ACL [3] . Nous avions juste rajouté un champ $room$ obligatoire pour indiquer la salle de tchat du message. Il n'est pas nécessaire de spécifier un destinataire. Cependant, le formalisme suivant a été retenu par les partenaires :

2. https://neo4j.com/fr/
3. http://www.fipa.org/specs/fipa00061/SC00061G.html

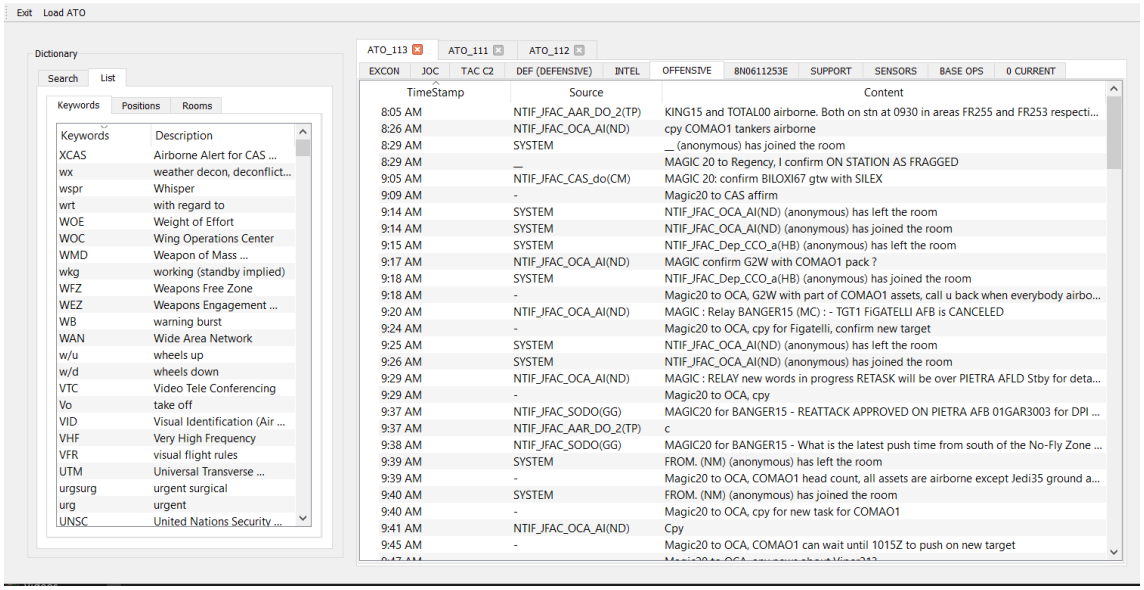


FIGURE 5.2 – Capture d'écran de l'outil de lecture des messages.

- un envoyeur,
- une salle,
- une liste, potentiellement vide, de destinataires,
- un horodatage de la date de réception,
- un champ sémantique composé de :
  - un performatif issu de ceux proposés par FIPA-ACL [4] ,
  - un sujet,
  - et une liste d'arguments attendus.
- un champ pour contenir la version en langage naturel :

> Le CASPOA impose aux utilisateurs du JCHAT de toujours indiquer un destinataire. Nous permettons la possibilité d'omettre un destinataire dans le message pour que nos agents puissent se tromper.

5.2.2 Gestion d'un TIC

Lors de nos visites au CASPOA, nous avons pu observer la cellule OFFENSIVE gérer un scénario avec un TIC. Voici le squelette des interactions :

1. L'AOCC demande un soutien aérien suite à la présence troupes ennemies.

4. http://www.fipa.org/specs/fipa00037/SC00037J.html#_Toc26729691

2. La cellule OFFENSIVE, mais surtout le CAS DO, doivent trouver une solution pour y répondre.
3. Selon la performance de la salle (rapidité et pertinence de la solution), le TIC est résolu plus ou moins rapidement et en gênant le moins possible les autres opérations.
4. L'AOCC informe de la résolution ou non de la situation.

C'est au niveau des points 2 et 3 que nos agents pourront se comporter différemment. Une des difficultés du projet est que les comportements sont exprimables uniquement aux travers des interactions sur le JCHAT. Voici un résumé de ce que nous avons retenu des positions pertinentes dans le cadre d'un TIC :

AOCC Ce poste est en lien avec les troupes au sol. Il va notamment relayer la demande d'un support aérien. Il informe aussi lorsque le TIC est résolu (si plus d'hostile sont présents), suite à la réalisation du plan proposé par le CAS DO.

CAS DO Ce poste est joué par l'apprenant. Son objectif est de gérer les TICs. Il doit notamment avoir connaissances des missions des avions et de leurs priorités.

SODO Ce poste dirige les postes de la cellule OFFENSIVE, comme celui du CAS DO. Il peut venir aider le CAS DO et transmettre les informations du TIC directement au CAS DO. Il peut aussi aider le CAS DO, en lui proposant une solution et transmettre les ordres pour répondre au TIC.

5.3 Application du méta-modèle

Nous appliquons OPACK au projet ORCHESTRAA. La première étape consiste à sélectionner un modèle d'agent adéquat, ainsi que les modèles cognitifs voulus. Pour ce projet, nous avons décidé d'utiliser un modèle d'agent fondé sur Replicants, afin de raisonner avec ACTIVITY-DL.

La première étape est de définir le modèle d'agent. Nous allons réutiliser le modèle construit lors du chapitre 3, fondé sur Replicants. Ensuite, la seconde étape est de décrire les comportements avec ACTIVITY-DL pour générer des comportements représentatifs et variables. Enfin, pour utiliser la variabilité offerte par ACTIVITY-DL, nous allons partiellement intégrer trois modèles cognitifs. Ils ont été choisis, car ils décrivent des comportements d'équipes en situations extrêmes, ce qui correspond à notre cadre applicatif :

1. Driskell [Driskell et al., 2018]
2. Fadier [Fadier et al., 2003]
3. Demary [Demary, 2020]

5.3.1 Construction du modèle d'agent

Le modèle d'agent que nous envisageons doit permettre à l'agent de raisonner sur un arbre d'activité et d'envoyer et répondre à des messages sur le JCHAT.

Nous utilisons donc un modèle d'agent fondé sur Replicants et ACTIVITY-DL. Nous avons présenté dans la section 2.2.2 de l'état de l'art, Replicants et ACTIVITY-DL. Succinctement, Replicants consiste à raisonner sur un arbre d'activité formalisé avec ACTIVITY-DL. Cet arbre d'activité permet de représenter les différentes manières temporelles et logiques de réaliser l'activité métier, comme gérer le TIC.

Définition de l'agent

Sens et effecteur La première étape est de définir les capacités d'actions et de perceptions de nos agents selon les contraintes de l'environnement.

- L'unique canal d'actions considéré est l'écriture de messages. Nos agents ne peuvent écrire qu'un seule message à la fois et ne peuvent pas communiquer par la voix. Nous définissons donc qu'un seul effecteur : les mains.
- L'unique canal de perception est celui des messages échangés sur le JCHAT. Nous définissons donc qu'un seul sens : la vision qui permet de prendre connaissance des messages visibles.

Dès lors, pour un agent i, nous avons :

$$\begin{aligned} Sen(i) &= \{Vision\} \\ Eff(i) &= \{Hands\} \end{aligned}$$

Résumé effecteur/sens

Un effecteur : les mains pour écrire des messages.
Un sens : la vision pour percevoir les messages

Caractéristiques Ensuite, nous simulons trois agents à trois postes différents. Nous ajoutons donc dans **C**, une caractéristique $Position$ pour spécifier leur poste : $AOCC$, $SODO$ ou CAS_DO

$$\mathbf{C} = \{\ldots, Position, \ldots\} \forall i \in Agt, Position \in C(i).$$

Connaissances Les connaissances de nos agents correspondent aux messages perçus et à l'activité métier. Nous avons défini un arbre d'activité pour la gestion d'un TIC, avec le squelette de son déroulement : déclenchement, analyse, résolution. Les tâches correspondent aux différentes tâches pour chaque position, décrivant ainsi l'activité dans

sa globalité et pas uniquement du point de vue d'une position. Les feuilles de cet arbre correspondent à des messages que les agents doivent observer sur le JCHAT.

Dès lors, l'arbre d'activité nous sert d'une part, à décrire l'activité, mais également comme base de connaissances. En effet, nous comptons, au travers des opérations et de la perception, indiquer les messages perçus dans cet arbre. Nous allons désormais décrire les opérations des agents.

La figure 5.3 est une version allégée de l'arbre d'activité. Les capsules noires sont les constructeurs logiques et temporels. Les capsules grises sont des informations additionnelles, rajoutées aux tâches grâce à la composabilité. Elles servent notamment à indiquer les positions responsables de l'exécution de la tâche. Les agents peuvent ainsi raisonner sur l'activité globalement. Par exemple, l'AOCC peut réprimander le CAS DO s'il ne propose pas une solution au bout d'un certain laps de temps.

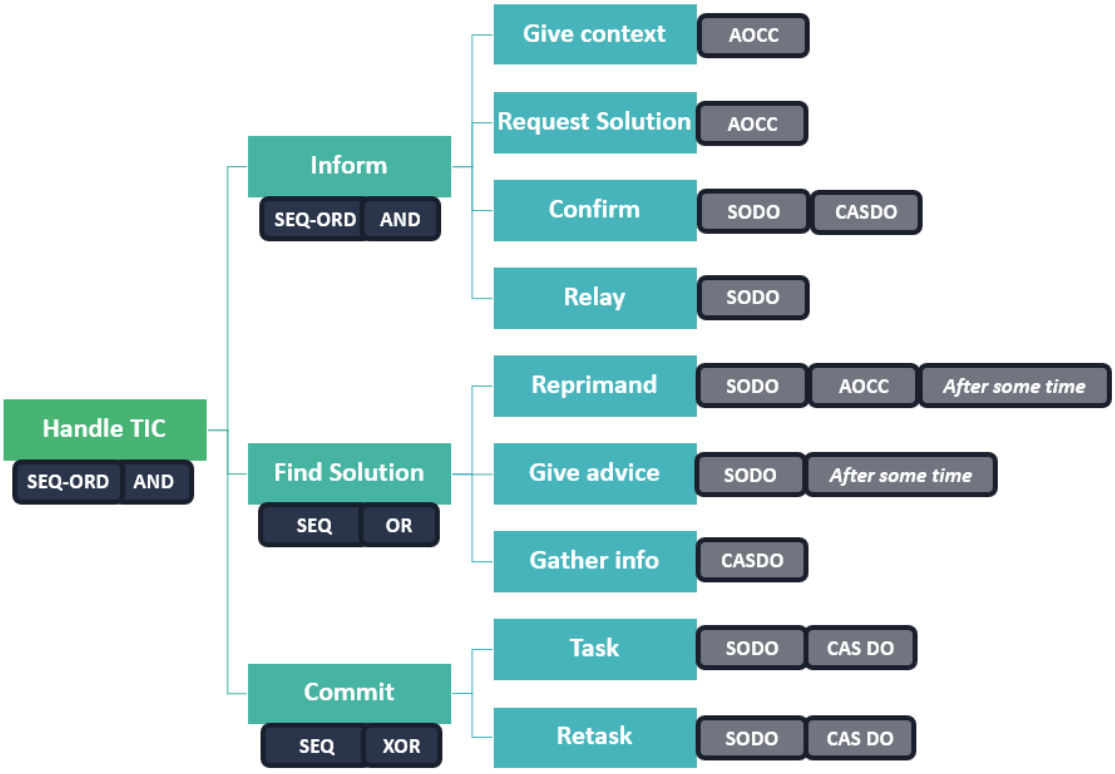


FIGURE 5.3 – Représentation d'un arbre d'activité simplifié pour résoudre un TIC.

Flux d'opérations Enfin, nous établissons les opérations nécessaires pour définir le raisonnement de nos agents. Nous réemployons le flux que nous avons construit lors du chapitre 3, avec quelques modifications (*cf.* 5.4).

Succinctement, nous avons une opération pour déterminer les actions envisageables, suivie d'une opération pour sélectionner une action. L'opération « Actions envisageables » est composée d'une fonction-impact par défaut qui parcourt l'arbre d'activité pour déterminer les actions envisageables selon ses connaissances.

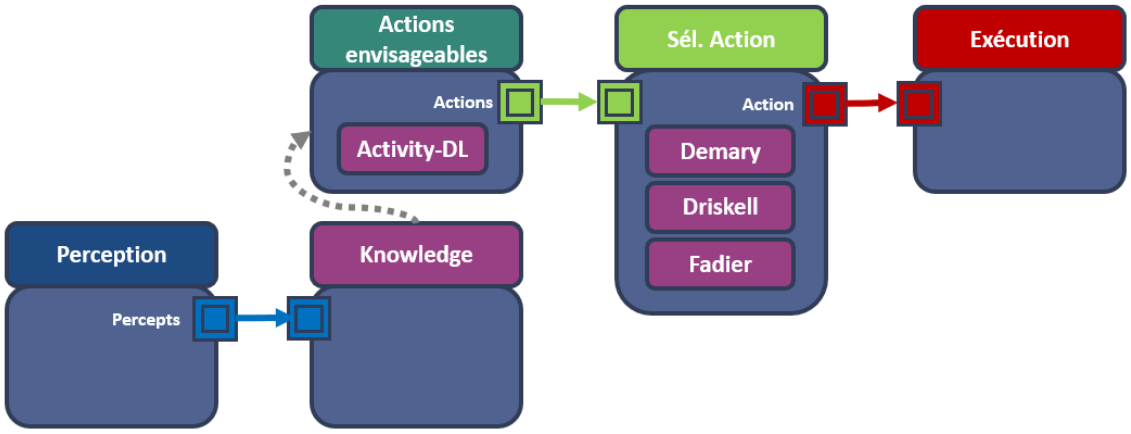


FIGURE 5.4 – Flux d'opérations pour le projet ORCHESTRAA

> Nous avons aussi ajouté un mécanisme de questions / réponses pour les besoins de la démo.

Ensuite, l'action sélectionnée est exécutée. Initialement, nous avions prévu d'ajouter un impact du stress sur l'exécution, pour altérer la qualité du message (en se trompant dans un sigle par exemple). Mais, par faute de temps, nous sommes allés au plus simple.

Enfin, nous avons deux autres opérations qui s'enchaînent : la perception puis la mise à jour des connaissances. Lors de la perception, l'agent prend connaissances des messages. Lors de la mise à jour, il regarde quels messages correspondent à une tâche de l'arbre d'activité pour indiquer si elle a été réalisée.

5.3.2 Intégration de modèles cognitifs

Nous réemployons l'intégration des modèles que nous avons réalisée précédemment, lors du chapitre 3.

Voici un résumé :

Notion de tag

Un tag est une caractéristique d'arité nulle, sans aucune structure de données. Autrement dit, il s'agit d'une méta-donnée, d'une étiquette, que le modélisateur peut librement associer aux identifiants. Cette notion permet d'ajouter simplement des informations pour le raisonnement.

Demary Ce modèle ajoute dans C, les traits suivants :

— Pro-actif / Passif

— Communicant / Non-communicant

Ce modèle ajoute dans K, les tâches suivantes :

— La vérification du plan proposé par le CAS DO.

— Ajout de tags pour les tâches pour indiquer celles qui correspondraient aux traits ajoutés.

Ce modèle ajoute dans des impacts lors de l'opération de sélection d'action. L'impact se traduit par des règles d'influences :

— Lorsque l'agent est pro-actif, une influence positive est appliquée aux tâches avec un tag $ProActive$.

— Lorsque l'agent est communicant, une influence positive est appliquée aux tâches avec un tag $Communicative$.

Driskell Ce modèle ajoute dans C, les traits suivants :

— Stressé / non-stressé

Ce modèle ajoute dans K, les tâches suivantes :

— ajout du tag STRESSÉ.

Ce modèle ajoute des impacts dans plusieurs opérations :

— Sélection d'action

  — Lorsque l'agent est stressé, une influence négative est appliquée aux tâches avec un tag $Communicative$.

— Exécution

  — Lorsque l'agent est stressé, le temps de réalisation d'une action est multiplié par un facteur arbitraire.

Fadier Nous avons ajouté un impact lors du parcours de l'arbre d'activité pour donner la possibilité au SODO de supplanter le CAS DO en proposant lui-même une solution.

5.4 Conclusion

Lors de ce chapitre, nous avons présenté le projet ORCHESTRAA dans son ensemble, ainsi que nos contributions.

Tout d'abord, nous avons pu appliquer le méta-modèle OPACK pour modéliser les opérateurs d'un centre de commandement aérien lors de la gestion d'un TIC. Nous avons d'une part modélisé un modèle d'agent : son état et son flux d'opérations, fondé sur REPLICANTS. Ce modèle a été appliqué pour simuler le fonctionnement de deux postes : le SODO et l'AOCC. Le troisième poste : CAS DO, central à la résolution du TIC, n'est pas simulé car il est joué par l'apprenant. Nous représentons le poste du CAS DO par un agent, mais il ne possède aucun flux d'opérations.

Ensuite, afin de simuler nos deux postes et que le modèle d'agent puisse produire des comportements métier, nous avons représenté l'activité des opérateurs grâce à Activity-DL. Dans cet arbre d'activité, nous avons représenté les différentes étapes de gestion d'un TIC, avec notamment les messages attendus. Cet arbre inclus la responsabilité des différents rôles et pas uniquement celle d'un rôle. Cela permet à chacun de raisonner sur l'ensemble de l'activité, selon leurs connaissances, pour permettre de nouveaux comportements, comme un rappel à l'ordre.

Enfin, nous avons réemployé l'intégration des modèles cognitifs du chapitre 3 pour cette application. Notre objectif était de générer des comportements variés selon le stress et le profil des agents, afin de rendre compte d'une bonne ou d'une mauvaise salle.

Le chapitre suivant présente les différentes évaluations et les résultats de nos différentes contributions, avec les retours du projet ORCHESTRAA.

Chapitre 6

Évaluations et résultats

Contents

Dans ce chapitre, nous présentons les différentes évaluations et expérimentations menées dans le cadre de la thèse. Nous commençons par deux évaluations préliminaires

des différentes propriétés recherchées : sensibilité, intelligibilité et représentativité. Elles ont été menées dans le cadre du projet VICTEAMS. Ensuite, nous présentons les retours de l'application de nos travaux au projet ORCHESTRAA Enfin, nous concluons sur des évaluations de performances de notre bibliothèque, vis-à-vis d'autres plate-formes de la communauté SMA.

6.1 Évaluation préliminaire de la sensibilité

Pour évaluer la sensibilité des modèles cognitifs sur le comportement de l'agent, nous avons proposé une première expérimentation qui mesure les actions sélectionnées par le modèle d'agent en fonction des modèles cognitifs. Nous allons vérifier que chaque modèle cognitif modifie bien ces mesures comme il le devrait. Par exemple, si nous ajoutons un modèle cognitif qui réduit la communication d'une personne stressée, alors les mesures en lien avec les actes de communication doivent évoluer dans ce sens. Au contraire, les autres mesures ne devraient pas être impactées.

Nous utilisons trois modèles cognitifs : Driskell [Driskell et al., 2018], Demary [Demary, 2020] et Fadier [Fadier et al., 2003].

6.1.1 Protocole

Dans notre scénario (entièrement abstrait), les agents doivent choisir parmi huit actions qui ont les caractéristiques suivantes (*cf.* table 6.1 pour la répartition) :

- *ordre* : est-ce que l'action est un ordre transmis par un supérieur ?
- *com* : est-ce que l'action est une action de communication ?
- *coop* : est-ce que l'action se fait à plusieurs ou aide les objectifs de l'équipe ?
- *alu* : est-ce que l'action est tolérée par l'usage (voir section 2.1) ?
- *SC1, SC2 ou SC3* : le niveau de qualification requis pour effectuer l'action.

Attribut	*ordre*	*com*	*coop*	*alu*	*SC3*	*SC2*	*SC1*
Pourcentage	12.5%	37.5%	50.0%	12.5%	12.5%	37.5%	50.0%

TABLE 6.1 – Pourcentage d'actions possédant la caractéristique. À part la qualification, une action peut posséder plusieurs autres de caractéristiques d'où l'addition des pourcentages de la partie gauche du tableau n'est pas égale à 100%.

Chaque agent va donc faire $2^8 - 1 = 255$ fois le cycle pour tirer l'ensemble des combinaisons possibles d'actions. En fonction de son état et des modèles cognitifs actifs ou non, il choisit une action (opération « sélection d'action »). Nous notons à chaque fois l'action choisie et les actions candidates.

Nous avons $2^3 * 3 = 24$ états d'agent possibles (communicant ou non, pro-actif ou passif, stressé ou non et trois qualifications possibles). Pour chaque modèle cognitif,

nous pouvons distinguer les états dans lesquels la fonction d'activation renverrait *vrai* pour les fonctions-impacts du modèle cognitif considéré. Par exemple, pour le modèle de Driskell, la fonction d'activation renvoie *true* pour les fonctions-impacts lorsque la variable *stress* de l'agent est vraie. Nous notons De, Dr et ALU l'ensemble de ces états pour les modèles de Demary [Demary, 2020], Driskell [Driskell et al., 2018] et Fadier [Fadier et al., 2003] respectivement. Réciproquement, nous notons $\overline{De}$, $\overline{Dr}$ et $\overline{ALU}$ l'ensemble des états qui n'activent pas les fonctions-impacts des modèles de Demary, Driskell et Fadier respectivement.

Concrètement, le groupe d'agents De est le groupe des agents pour lesquels le comportement **devrait** être impacté quand le modèle de Demary est présent. Réciproquement, le groupe d'agents $\overline{De}$ est le groupe pour lequel le comportement ne devrait pas être impacté.

6.1.2 Mesures

Nous considérons les mesures suivantes et nous indiquons entre parenthèse le modèle cognitif qui devrait impacter la mesure :

ORD = pourcentage d'actions *« ordre »*, (De),

COM = pourcentage d'actions *« com »* (Dr),

ALU = pourcentage d'actions *« alu »* (ALU),

COOP = pourcentage d'actions *« coop »* (Dr),

QUAL = pourcentage d'actions choisies dont le niveau de qualification est inférieur ou égal à celui de l'agent (Dr).

Concrètement, pour une combinaison de modèles cognitifs donnée, $ORD(Dr) = 60\%$ signifie que dans l'ensemble des exécutions où une action *« ordre »* était proposée dans la liste des actions candidates et où la fonction d'activation aurait renvoyé *vrai* suivant les conditions du modèle de Driskell, l'agent a choisi une action *« ordre »* dans 60% des cas.

6.1.3 Résultats

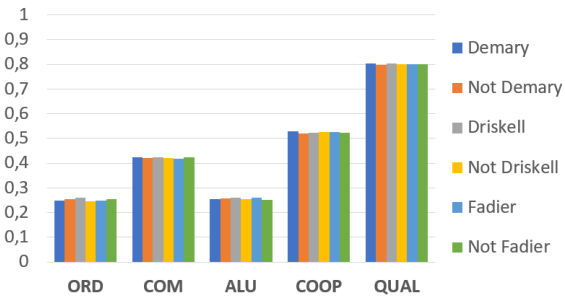


FIGURE 6.1 – Taux de sélection d'action lorsqu'aucun modèle n'est activé.

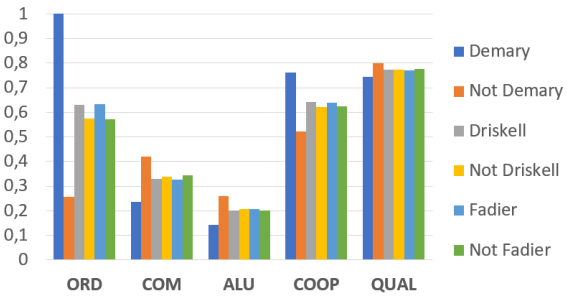


FIGURE 6.2 – Taux de sélection d'action lorsque seul le modèle de Demary est activé.

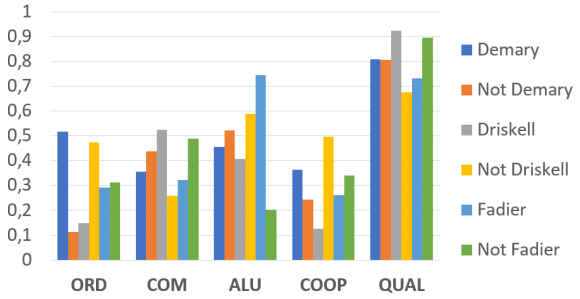


FIGURE 6.3 – Taux de sélection d'action lorsque tous les modèles sont activés.

Lecture des graphes

Chaque barre représente le taux de sélection d'une action avec le tag correspondant. La couleur représente le groupe associé avec cette barre. Le groupe bleu, par exemple, est l'ensemble des agents où la fonction d'activation retournerait $true$ pour Demary. Le groupe orange est le contraire ($false$ pour Demary).

Exemple de lecture avec la figure 6.1 Dès qu'ils pouvaient choisir une action ORD, les agents du groupe De *(en bleu)* ont choisi une action ORD dans 25% des cas, comme le groupe des agents $\overline{De}$ *(en orange)*.

Lorsque aucun modèle cognitif n'est présent, quelle que soit la mesure, aucune différence n'existe entre les groupes (puisqu'aucune fonction-impact n'est activée), comme la figure 6.1 le reporte. C'est le comportement de référence.

Lorsque Demary est le seul modèle cognitif activé et qu'une action ORD est proposée, le groupe De choisit toujours cette action ($ORD(De) = 1$) alors que le groupe $\overline{De}$ la choisira une fois sur quatre ($ORD(\overline{De}) = 0.2553$), comme dans le comportement de référence. Puisque, l'unique action ordonnée dans notre jeu de test est une action

non-communicative, respectant la réglementation (non-ALU) et coopérative, nous observons une diminution de la mesure COM et ALU et une augmentation de la mesure $COOP$ pour le groupe actif. Les autres groupes ne sont pas affectés et restent sur le comportement de référence. Ces résultats sont présentés sur la figure 6.2).

Nous obtenons le même résultat pour les autres modèles cognitifs lorsqu'ils sont activés seuls. Lorsque nous activons les trois modèles cognitifs, la tendance de chaque modèle cognitif se conserve (*cf.* figure 6.3). Par exemple, $ORD(De) = 0.51 > ORD(\overline{De}) = 0,11$, c'est-à-dire que le groupe actif de Demary continue à privilégier les ordres, contrairement au groupe non-actif. C'est le cas pour tous les modèles cognitifs. Par contre, nous observons des impacts sur les groupes à première vue non-concernés par un modèle, comme les actions peuvent correspondre à plusieurs critères.

6.1.4 Bilan

Nous avons montré que les modèles cognitifs ont un impact net sur la sélection d'action, qui est un élément important de la génération de comportements. Ces travaux ont été présentés lors de JFSMA [de Blauwe et al., 2022b]

Nous nous sommes limités, dans cette première étude, à l'opération de sélection d'action. Nous aurions aimé pouvoir aller plus loin et mesurer l'impact du modèle cognitif sur d'autres opérations, autres que la sélection d'action, comme l'exécution d'une action.

En effet, pour étendre notre évaluation et qualifier l'intelligibilité du modèle d'agent construit sur notre méta-modèle OPACK, il faudrait simuler le modèle d'agent complet et analyser les séquences d'actions produites. Nous pourrions alors demander à des experts du domaine de qualifier les comportements cognitifs observés (est-ce que l'agent était stressé, était-il communicant), pour répondre à la question : « est-ce que le changement de comportement produit par l'intégration ou non d'un modèle cognitif est intelligible » ? C'est ce que nous avons débuté lors de l'expérimentation au projet ORCHESTRAA, décrit lors de la section 6.3. Auparavant, nous avons réalisé une autre évaluation préliminaire, mais cette fois de l'intelligibilité et de la représentativité des comportements générés.

6.2 Évaluation préliminaire de l'intelligibilité et de la représentativité

Nous présentons dans cette section, une évaluation utilisateur pour tester la représentativité et l'intelligibilité, dans un contexte de la médecine d'urgence [Huguet et al., 2016]. Nous posons les hypothèses de recherches suivantes :

- **H1 - représentativité** : notre modèle permet de combiner plusieurs modèles cognitifs, tout en restant représentatifs.
- **H2 - intelligibilité** : notre modèle permet de générer des comportements où les utilisateurs sont capables d'identifier les profils de nos agents (selon les termes employés par les modèles cognitifs, *e.g.* stressé, proactif, *etc.*).

— **H3 - de l'intelligibilité vers l'explicabilité** : notre modèle permet d'extraire les causes d'un comportement pour la construction d'explications pertinentes, soit qui aident à mieux comprendre le comportement de l'agent.

Afin de vérifier ces hypothèses, nous avons mené une évaluation préliminaire en employant le protocole suivant.

6.2.1 Protocole

L'expérience est fondée sur le scénario suivant : un médecin diagnostique un patient, conclut (de manière erronée) qu'il a besoin d'une injection d'adrénaline et demande à une infirmière de préparer l'injection. L'infirmière s'aperçoit de l'erreur de diagnostic : le patient a en réalité besoin de morphine. Elle réagit donc aux ordres du médecin de manière différente selon son profil cognitif : passif ou proactif (modèle cognitif de Demary [Demary, 2020]), communicatif ou non (modèle cognitif de Demary), stressé ou non (modèle cognitif de Driskell [Driskell et al., 2018]). Elle va d'abord réagir à l'instruction du médecin (soit en mentionnant l'erreur au médecin, soit en confirmant l'ordre initial, soit en ne disant rien) puis préparer une injection (de morphine ou d'adrénaline) puis rendre compte de ses actions, de manière correcte (elle dit ce qu'elle a fait) ou erronée (elle dit le contraire de ce qu'elle a fait) ou inexistante (elle ne dit rien). À chaque étape, le choix de l'agent est déterminé par la combinaison des différents modèles cognitifs et les paramètres initiaux de l'agent : cela donne finalement huit exécutions différentes possibles.

Le scénario obtenu est traduit sous forme textuelle. En complément, une explication est proposée à l'utilisateur soit construite de manière systématique à partir des variables des modèles cognitifs, comme illustré sur la figure 6.4, soit décrivant simplement le comportement (par exemple : « l'agent est dans une situation où il est susceptible de se tromper, notamment pour des tâches relativement simples »). Cela produit seize conditions expérimentales en tout, chacune correspondant à un texte d'une vingtaine de lignes, scénario et explications compris.

Nous notons nos quatre variables X avec deux modalités $X+$ et $X-$ de la manière suivante :

C : communicant ($C+$) et non-communicant ($C-$).

P : proactif ($P+$) et passif ($P-$).

S : stressé ($S+$) et non-stressé ($S-$).

E : explications issues de modèles ($E+$) et explications naïves ($E-$).

Nous avons utilisé G*Power [Faul et al., 2007, Faul et al., 2009] pour estimer le nombre minimum de participants (128), répartis dans nos seize groupes, avec une taille d'effet moyenne ($0,25$). 174 participants ont été recrutés en ligne parmi les étudiants et le personnel du laboratoire, avec un minimum de dix participants par condition. Chaque participant a reçu de manière aléatoire un scénario parmi les huit possibles. Après avoir lu le texte, les sujets ont été invités à évaluer la crédibilité perçue du comportement de l'infirmière sur une échelle de Likert à cinq points (de -2 - pas du tout crédible - à 2

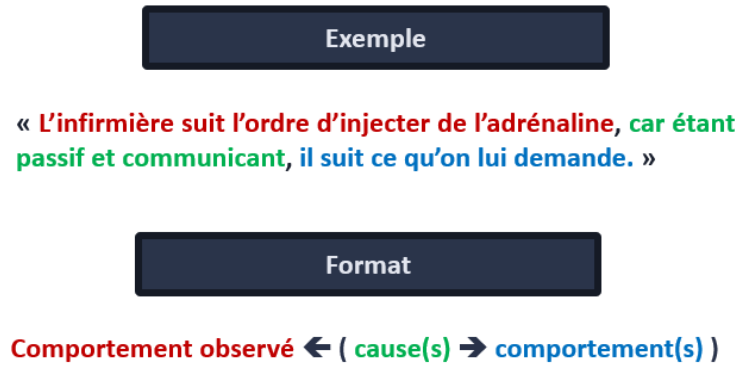


FIGURE 6.4 – Méthode de construction d'une explication à partir des modèles cognitifs. L'exemple présente une explication construite à partir du modèle cognitif de Demary [Demary, 2020].

- tout à fait crédible), noté R. Ils doivent ensuite qualifier le comportement de l'agent pour chaque variable cognitive : « Diriez-vous que l'infirmière est… » stressée ou non stressée, passive ou proactive, communicante ou non communicante. Chaque variable est aussi présentée sous la forme d'une échelle de Likert à 5 points (voir exemple du questionnaire en annexe 8.1.1). Dans un second temps, les sujets reçoivent les explications. Il leur est demandé d'évaluer la pertinence de l'explication (échelle de Likert de -2 à 2). Ensuite, ils peuvent modifier leur évaluation initiale de la crédibilité perçue de l'agent. Malheureusement, une mauvaise manipulation ne nous a pas permis de récupérer cette donnée. Enfin, un champ libre permet aux sujets de justifier chacune de leurs réponses.

En suivant nos hypothèses de recherche, nous avons formulé trois hypothèses expérimentales :

(H1) $R > 0$: Le score moyen de crédibilité perçue est positif : cela suggérerait que, lorsque plusieurs modèles cognitifs sont combinés, leurs comportements restent crédibles.

(H2) Pour chacune de nos variables C, P et S, nous allons comparer les deux modalités avec un test U de Mann-Whitney : le groupe avec la modalité $X+$ et le groupe avec la modalité $X-$. Les groupes sont aussi notés respectivement $X+$ et $X-$ par abus de langage. Si le test retourne que le groupe $X+$ est plus grand pour le groupe $X-$, cela suggérerait que chaque groupe a correctement identifié et différencié la modalité. Plus précisément :

- **(H2.1)** $C+ > C-$: Le groupe de sujets $C+$ ayant eu un scénario avec un agent communicant a tendance à évaluer le score communicant de manière plus élevée que pour le groupe $C-$ ayant eu un agent non-communicant.
- **(H2.2)** $P+ > P-$: Le groupe de sujets $P+$ ayant eu un scénario avec un agent proactif a tendance à évaluer le score proactif (2) / passif (-2) de manière plus élevée pour le groupe de sujets $P-$ ayant eu un agent passif.

(H2.3) $S+ > S-$: Le groupe de sujets $S+$ ayant eu un scénario avec un agent stressé a tendance à évaluer le score de stress de manière plus élevée que pour le groupe $S-$ ayant eu un agent non-stressé.

La validation de ces trois hypothèses suggérerait que, lorsque plusieurs modèles cognitifs sont présents, ils restent intelligibles pour les observateurs.

(H3) $E+ > E-$: Le groupe $E+$, ayant reçu des explications issues des modèles cognitifs, a tendance à évaluer de manière plus élevée le score de pertinence des explications que pour le groupe $E-$, ayant reçu une explication « naïve ». Cela suggérerait qu'une explication apportée par les modèles que nous avons utilisés, aide à mieux comprendre le comportement des agents, qu'une explication naïve.

6.2.2 Résultats

Nos mesures ne suivent pas une loi normale selon le test de Shapiro-Wilk (*cf.* table 6.2.2 et graphes de densités de la figure 6.5). C'est pourquoi nous utilisons des tests de Mann-Whitney plutôt que des tests de Student pour les hypothèses **H2** et **H3**.

TABLE 6.2 – Test de Shapiro-Wilk sur nos différentes variables.

	R	C	P	S	E
N	178	178	178	178	178
W de Shapiro-Wilk	0.829	0.743	0.860	0.892	0.838
Valeur p de Shapiro-Wilk	.001	.001	.001	.001	.001

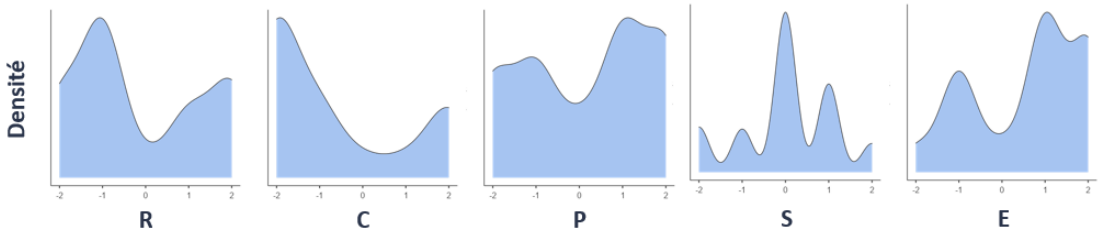


FIGURE 6.5 – Graphes de densité de nos variables R, C, P, S, E.

H1 La moyenne sur l'ensemble des groupes concernant la crédibilité du comportement de l'infirmière est de -0.152, ce qui contredit l'hypothèse **H1** (*cf.* table 6.3). Cependant, nous avons des scores positifs (1.818 et 1.125) pour les groupes correspondant à un profil communicatif et proactif (respectivement sans et avec stress), ce qui suggère que cette combinaison de comportement entraine des comportements plus crédibles (*cf.* table 6.4).

TABLE 6.3 – Statistiques descriptives de la crédibilité avec toutes les conditions.

								Shapiro-Wilk	
Statistiques descriptives	N	Manquants	Moyenne	Médiane	Écart-type	Minimum	Maximum	W	p
R	178	0	-0.152	-1.00	1.49	-2	2	0.829	0.001

TABLE 6.4 – Statistiques descriptives de la crédibilité selon les conditions

Effective C	Effective P	Effective S	N	Manquants	Moyenne	Médiane	Écart-type	Min.	Max.
Non-communicant	Passif	Non-stressé	22	0	-0.545	-1.00	1.299	-2	2
		Stressé	21	0	-0.905	-1	1.179	-2	2
	Proactif	Non-stressé	26	0	-0.731	-1.00	1.343	-2	2
		Stressé	21	0	-0.857	-1	0.655	-2	1
Communicant	Passif	Non-stressé	20	0	-0.550	-1.00	1.276	-2	2
		Stressé	22	0	-0.682	-1.00	1.287	-2	2
	Proactif	**Non-stressé**	22	0	**1.818**	2.00	0.501	0	2
		Stressé	24	0	**1.125**	2.00	1.296	-2	2

H1 - représentativité : non vérifiée

$R = -0.152 < 0$. Les comporements n'ont pas été jugés comme crédibles, sauf pour les conditions avec un profil communicatif et proactif (R > 1).

H2 Nous observons une différence nette entre chaque échantillon, hormis pour le trait « stressé » (*cf.* figure 6.6). Nous avons posé un niveau de significativité $\alpha < 0.05$.

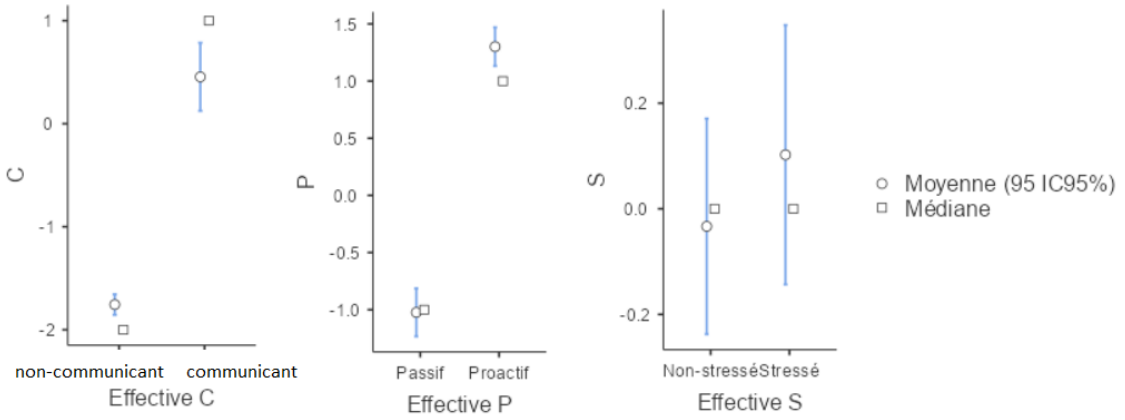


FIGURE 6.6 – Graphes descriptifs de nos trois couples de traits X et $\neg X$.

H2.1 Le groupe $C+$ est significativement supérieur au groupe $C-$ selon le test statistique U de Mann-Whitney, avec $p < 0.001 < \alpha$, ainsi que taille d'effet importante de 0.754 (*cf.* table 6.2.2). D'autres éléments de preuves sont la moyenne et la médiane qui vont dans ce sens (*cf.* table 6.2.2). Par exemple, la médiane de $C+$ (1.00) est effectivement supérieure que celle de $C-$ (-2.00), idem pour la moyenne (*cf.* table 6.2.2). L'hypothèse

H2.1 est donc validée, ce qui suggère que les sujets ont su différencier entre un comportement communicant et non-communicant.

TABLE 6.5 – Test U de Mann-Whitney : $C+ > C-$

		Statistique	p		Taille de l'effet
C	U de Mann-Whitney	976	.001	Corrélation entre rangs bisériés	0.754

TABLE 6.6 – Statistiques descriptives des groupes $C+$ et $C-$.

	Groupe	N	Moyenne	Médiane	Écart-type	Erreur standard
C	Non-communicant	90	-1.76	-2.00	0.481	0.0507
	Communicant	88	0.455	1.00	1.58	0.169

H2.1 - intelligibilité : vérifiée

Selon le test U de Mann-Whitney, $C+ > C-$ avec $p < 0.001 < \alpha = 0.05$, ce qui suggère que les sujets ont su différencier entre un comportement communicant et non-communicant.

H2.2 Le groupe $P+$ est significativement supérieur au groupe $P-$ selon le test statistique U de Mann-Whitney, avec $p < 0.001 < \alpha$, ainsi que taille d'effet importante de 0.878 (*cf.* table 6.7). D'autres éléments de preuves sont la moyenne et la médiane qui vont dans ce sens (*cf.* table 6.8). Par exemple, la médiane de $C+$ (1.00) est effectivement supérieure à celle de $C-$ (-1.00), idem pour la moyenne (*cf.* table 6.2.2). L'hypothèse **H2.2** est donc validée, ce qui suggère que les sujets ont su différencier entre un comportement passif et proactif.

TABLE 6.7 – Test U de Mann-Whitney : $P+ > P-$

		Statistique	p		Taille de l'effet
P	U de Mann-Whitney	481	.001	Corrélation entre rangs bisériés	0.878

TABLE 6.8 – Statistiques descriptives des groupes $P+$ et $P-$.

	Groupe	N	Moyenne	Médiane	Écart-type	Erreur standard
P	Passif	85	-1.02	-1.00	0.988	0.107
	Proactif	93	1.30	1.00	0.831	0.0862

H2.2 - intelligibilité : vérifiée

Selon le test U de Mann-Whitney, $P+ = 1.00 > P- = -1.00$ avec $p < 0.001 < \alpha = 0.05$, ce qui suggère que les sujets ont su différencier entre un comportement proactif et passif.

H2.3 Le groupe $S+$ n'est pas significativement supérieur au groupe $P-$ selon le test statistique U de Mann-Whitney, avec $p = 0.196 > \alpha$ (*cf.* table 6.9). La médiane et la moyenne vont aussi dans ce sens (*cf.* table 6.10). Par exemple, les médianes des deux groupes valent 0 et les moyennes sont très proches (-0.0333 pour $S-$ et 0.102 pour $S+$) (*cf.* table 6.10). L'hypothèse **H2.3** n'est donc pas validée ce qui suggère que les sujets n'ont pas su différencier entre un comportement stressé et non-stressé.

TABLE 6.9 – Test U de Mann-Whitney : $S+ > S-$

		Statistique	p		Taille de l'effet
S	U de Mann-Whitney	3539	0.196	Corrélation entre rangs bisériés	0.106

TABLE 6.10 – Statistiques descriptives des groupes $S+$ et $S-$.

	Groupe	N	Moyenne	Médiane	Écart-type	Erreur standard
S	Non-stressé	90	-0.0333	0.00	0.988	0.104
	Stressé	88	0.102	0.00	1.17	0.125

H2.3 - intelligibilité : non vérifiée

Selon le test U de Mann-Whitney, $S+$ n'est pas strictement supérieur à $S-$, avec $p = 0.196 > \alpha = 0.05$, ce qui suggère que les sujets n'ont pas su différencier entre un comportement stressé et non stressé.

H3 Le groupe $E+$ est significativement supérieur au groupe $E-$ selon le test statistique U de Mann-Whitney, avec $p = 0.048 < \alpha$ (*cf.* table 6.2.2), malgré le fait que si nous regardons la médiane, elle vaut 0 pour les deux groupes. D'autres éléments de preuve sont la moyenne et la répartition. Pour la moyenne, elle est effectivement supérieure pour le groupe $E+$ avec une valeur de 0.793 contre 0.407 pour $E-$ (*cf.* table 6.10). Si nous regardons la répartition de nos deux groupes (*cf.* figure 6.7), nous remarquons une différence nette entre le groupe $E+$ où l'avis des sujets est plus tranché que le groupe $E-$. L'hypothèse **H3** est validée ce qui suggère que les sujets ont trouvé les explications de notre modèle plus pertinentes que celles naïves, avec toutefois une taille d'effet basse (0.164).

Lecture d'un graphe violon

Un graphe violon permet de comparer visuellement la distribution sur plusieurs populations. Le fil passant à travers la boite bleu indique les quartiles (un de chaque côté de la boîte, un dans la boite bleu et le dernier sur la ligne en gras).

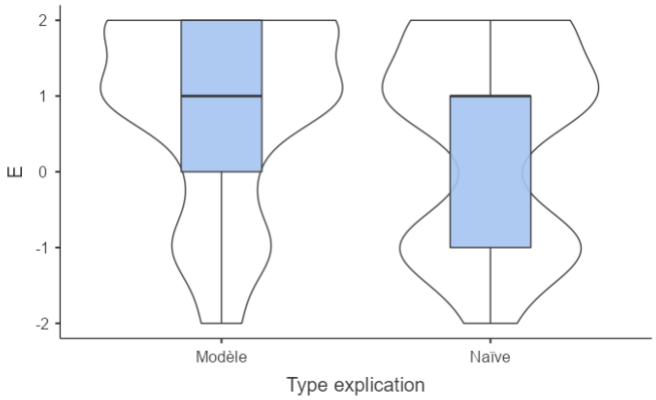


FIGURE 6.7 – Graphes violons des groupes $E+$ et $E-$.

TABLE 6.11 – Test U de Mann-Whitney : $E+ > E-$

		Statistique	p		Taille de l'effet
E	U de Mann-Whitney	3307	0.048	Corrélation entre rangs bisériés	0.164

H3 - de l'intelligibilité vers l'explicabilité : vérifiée

Selon le test U de Mann-Whiteny, $E+ > E-$ avec $p = 0.48 < \alpha = 0.05$, ce qui suggère que les sujets ont trouvé les explications de notre modèle plus pertinentes que celles naïves, avec toutefois une taille d'effet basse (0.164).

6.2.3 Discussions

La majorité des personnes n'ont pas trouvé les comportements des scénarios passifs convaincants. Aussi, les comportements stressés ont été mal perçus. Une explication serait que nous avons mal représenté l'urgence de la situation, ainsi que le contexte (tente militaire sur le front), ce qui est difficile sans EV et sans introduire de biais. Les commentaires des utilisateurs vont dans ce sens : plusieurs d'entre eux ont jugé que le comportement de l'agent était caricatural.

Malgré cela, nous avons validé deux hypothèses sur trois concernant l'intelligibilité et celle de la pertinence des explications. Ce qui est encourageant quant à l'apport de notre modèle pour la simulation en EV en termes d'intelligibilité des comportements. Ces travaux ont été présentés à la conférence CSCWD en 2022 [de Blauwe et al., 2022a].

Nous allons désormais présenter une évaluation complémentaire de la perception de l'intelligibilité et de la représentativité des comportements produits par notre modèle, dans le cadre du projet ORCHESTRAA.

TABLE 6.12 – Statistiques descriptives des groupes $E+$ et $E-$.

	Type explication	N	Manquants	Moyenne	Médiane	Ecart-type	Minimum	Maximum	Centiles 25th	50th	75th
E	Modèle	92	0	0.793	1.00	1.23	-2	2	0.00	1.00	**2.00**
	Naïve	86	0	0.407	1.00	1.31	-2	2	-1.00	1.00	1.00

6.3 Expérimentation avec le projet ORCHESTRAA : mesure de la représentativité et de l'intelligibilité

Cette expérience a pour objectif de tester la représentativité et l'intelligibilité de notre génération de comportements, appliqué au projet ORCHESTRAA. Nous posons les mêmes hypothèses de recherche, que celles de l'évaluation précédente (*cf.* section 6.2), hormis **H3** :

- **H1 - représentativité** : notre modèle permet de combiner plusieurs modèles cognitifs, tout en restant représentatifs.
- **H2 - intelligibilité** : notre modèle permet de générer des comportements où les utilisateurs sont capables d'identifier les profils de nos agents (selon les termes employés par les modèles cognitifs, *e.g.* stressé, proactif, *etc.*).

Nous avons à notre disposition dix experts. Ils sont supposés capables de lire les messages. Toutefois, il se peut que nos experts proviennent de cellules/de postes différents et/ou qu'ils n'aient jamais été dans ces postes. Autrement dit, nous ne connaissons pas leur degré de maîtrise des postes que nous allons simuler.

6.3.1 Scénario

Notre expérimentation utilise le scénario suivant :

Scénario

« Un TIC est reporté par une troupe au sol qui subit des tirs d'artillerie. L'AOCC relaye l'information et requiert un support aérien. Le CAS DO doit prendre connaissance de l'information et fournir une solution. Le SODO en tant que supérieur du CAS DO, peut intervenir pour l'aider ou le rappeler à l'ordre. Le CAS DO peine à trouver une solution mais finit par en donner une, ce qui conclut la simulation. »

Nous précisons que le CAS DO est joué par un utilisateur humain, tandis que nous simulons le comportement de l'AOCC et du SODO. Nous adoptons le comportement d'un CAS DO qui peine à fournir une solution. Le but est soit d'aggraver la situation par le comportement d'autres postes, soit au contraire d'y palier. Enfin, le scénario est fondé sur une session déjà réalisée par le CASPOA.

Hormis les performances du CAS DO toujours mauvaises, le niveau du reste de la salle varie selon le profil de l'AOCC et du SODO. Leur profil est déterminé par une

combinaison de modèles cognitifs. Ces modèles affectent le processus de décision, mais également la réalisation d'actions et la perception, comme nous avons vu lors des chapitres précédents. Quoi qu'il en soit, il suffit de retenir que nous avons au total trois caractéristiques, soit trois couples de traits antagonistes qui affectent le comportement :

- passif, proactif,
- non-communicant, communicant,
- stressé, non-stressé.

Nous pourrions avoir ainsi $2^6 = 64$ variations du scénario, en considérant nos deux agents simulés. Toutefois, nous allons uniquement considérer deux niveaux de salle (une des raisons est que nous disposons uniquement de dix sujets) :

- $S-$: Une mauvaise salle où l'AOCC et le SODO ont les traits : passif, non-communicant et stressé
- $S+$: Une bonne salle où l'AOCC et le SODO ont les traits : pro-actif, communicant et non-stressé

Selon le niveau de la salle, le temps de résolution du TIC sera plus ou moins rapide. Ce temps est déterminé par la justesse de la solution et la rapidité pour la trouver. Le temps de résolution est retourné par l'environnement virtuel.

6.3.2 Conditions

Pour éviter que les sujets passent sur plusieurs conditions et pour ne pas les comparer, nous sommes partis sur une expérimentation inter-sujet plutôt qu'intra-sujet. Nous sommes conscients que cela ne sera pas statistiquement valide avec nos dix sujets.

Dès lors, nous sommes partis sur les deux variations du scénario $S-$ et $S+$ décrites précédemment. Nous y réglons le profil du SODO et de l'AOCC aux extrêmes pour traduire une mauvaise ou une bonne salle. Ensuite, nous voulons aussi comparer les comportements avec et sans le méta-modèle. Sans le méta-modèle consiste à générer les comportements uniquement avec ACTIVITY-DL, soit sans les modèles cognitifs. Ce qui nous fait un total de 4 conditions :

1. $M-$, soit le groupe associé au scénario $S-$ avec le méta-modèle.
2. $\overline{M}-$, soit le groupe associé au scénario $S-$ sans le méta-modèle, *i.e.* sans les comportements issus des modèles cognitifs.
3. $M+$, soit le groupe associé au scénario $S+$ avec le méta-modèle.
4. $\overline{M}+$, soit le groupe associé au scénario $S+$ sans le méta-modèle, *i.e.* sans les comportements issus des modèles cognitifs.

Dans les faits, les comportements générés pour les groupes $\overline{M+}$ et $\overline{M-}$ sont identiques. Il n'y a pas de variation puisque les comportements sont désactivés. Il aurait été donc plus pertinent de considérer uniquement trois conditions plutôt que quatre. Nous aurions ainsi pu avoir trois sujets par condition plutôt que deux.

Le groupe $M+$ correspondrait donc à un scénario avec une bonne salle, le groupe $M-$ à une mauvaise salle et le groupe M a un entre deux.

6.3.3 Déroulé de l'expérience

Chacun de nos experts passe sur une unique condition. Au total, nous avons attribué deux experts par condition. Nous gardons deux experts au cas où, pour les conditions où les avis sont trop divergents entre les deux. L'expérience a lieu en ligne. Les sujets visionnent une vidéo de la RAP en temps réel. Le but est de donner des éléments aux experts pour évaluer la situation en temps réel de l'espace aérien et pour ancrer les échanges de messages. Nous avons deux variations de cette vidéo correspondant aux deux niveaux de salle. La variation observable est au niveau du temps pour dérouter un avion vers le TIC. Ensuite, les sujets ont à leur disposition la trace des messages échangés sur le JCHAT. Puis, ils doivent remplir un questionnaire disponible en annexe 8.1.2.

6.3.4 Mesures

Les sujets remplissent un questionnaire pour évaluer la représentativité du comportement, identifier le niveau de la salle et les caractéristiques des agents.

Nos variables indépendantes correspondent aux profils de nos agents, soit les deux salles, avec les scénarios $S-$ et $S+$. Nous utilisons la notation x_p pour indiquer que la mesure x concerne le poste p. Chaque mesure est une échelle de Likert comprise entre -2 et 2.

Degré de maîtrise Tout d'abords, nous mesurons le degré de maîtrise, noté M_p pour les positions SODO, AOCC et CAS DO.

Mesure du degré de maîtrise

non-familier $-2 \leq Y_p \leq 2$ familier

Niveau de salle Nous demandons ensuite le niveau de salle, noté L.

Mesure du niveau de salle de maîtrise

peu performante $-2 \leq L \leq 2$ performante

Représentativité Ensuite, le sujet note la représentativité de chaque position SODO, CAS DO et AOCC, noté R_p.

Mesure de la représentativité

non-représentatif $-2 \leq R_p \leq 2$ représentatif

Intelligibilité Enfin, le sujet évalue le profil de l'AOCC, du SODO et du CAS DO pour chaque couple antagoniste de traits :

- C_p, le score attribué au couple non-communicant(-2)/communicant(2).
- P_p, le score attribué au couple passif(-2)/proactif(2).
- S_p, le score attribué au couple stressé(-2)/non-stressé(2).

Mesures de l'intelligibilité

$$\begin{array}{rcl} \text{non-communicant} & -2 \leq C_p \leq 2 & \text{communicant} \\ \text{passif} & -2 \leq P_p \leq 2 & \text{pro-actif} \\ \text{stressé} & -2 \leq S_p \leq 2 & \text{non-stressé} \end{array}$$

La correspondance des valeurs des bornes avec les traits est faite pour que pour le scénario $S-$, les valeurs effectives correspondent à la borne négative et vice-versa pour le scénario $S+$.

6.3.5 Hypothèses opérationnelles

Pour chaque mesure x, la notation $x_p(g)$ indique la mesure x pour la position p du groupe g. Par exemple, $R_{CASDO}(M)$ indique la mesure de représentativité de la position CAS DO dans le groupe M, soit le groupe contenant les deux sous groupes $M+$ et $M-$. Nous avons formulé trois hypothèses expérimentales :

Hypothèse H1 : Représentativité

$\forall p \in \{\text{CASDO}, \text{SODO}, \text{AOCC}\}, R_p(M) > 0, R_p(\overline{M}) > 0$

Autrement dit, le score moyen de représentativité R perçu pour chaque position est positif pour tous les groupes. Plus le score est grand, plus les comportements sont perçus comme représentatifs/crédibles. Une hypothèse plus forte serait de poser $R_p(M) >= R_p(\overline{M})$ pour indiquer que l'ajout de modèles cognitifs n'a pas été détrimentale à la représentativité (scores égaux), voir au contraire positive (score supérieur).

Hypothèse H2.1 : Intelligibilité - comportement individuel

$$\forall p \in \{\text{CASDO}, \text{SODO}, \text{AOCC}\}, \forall x \in \{P, C, S\}, x_p(M+) > x_p(M-)$$

Un test applicable si nous avions suffisamment de sujets est de comparer les modalités avec un test U de Mann-Whitney : le groupe avec la modalité $X+$ et le groupe avec la modalité $X-$. Si le test retourne que le groupe $M+$ est plus grand que pour le groupe $M-$ pour une mesure x_p, cela suggérerait que chaque groupe a correctement identifié et différencié la modalité pour la position. Une hypothèse plus forte serait de tester si nous obtenons : $\forall p \in \{\text{CASDO}, \text{SODO}, \text{AOCC}\}, \forall x \in \{P, C, S\}, x_p(M+) > x_p(\overline{M}) > x_p(M-)$

Hypothèse H2.2 : Intelligibilité - niveau de salle

$$L(M+) > L(M-)$$

Nous appliquons le même raisonnement que pour **H2**, mais cette fois en regardant le niveau de la salle. Si $L(M+)$ est $L(M-)$, alors cela suggérait que les groupes ont correctement identifié le niveau de la salle et que nous avons pu impacter le niveau de la salle, à partir du profil des agents. Nous pourrions aussi poser une hypothèse plus forte : $L(M+) > L(\overline{M}) > L(M-)$.

6.3.6 Résultats

Nous n'avons pas réalisé de test statistique comme Kruskall-Walis ou Mann-Whitney. Nous n'avons pas assez de sujets. C'est pourquoi, nous nous contentons des statistiques descriptives, bien qu'il s'agisse de preuves beaucoup moins fortes.

Les mesures récoltées auprès des sujets sont disponibles dans le tableau 6.13. Le tableau 6.14 résume les statistiques descriptives du score de représentativité des positions selon le groupe. Le tableau 6.15 décrit les statistiques descriptives des mesures du niveau de salle selon le groupe. Le tableau 6.16 décrit les statistiques descriptives des mesures d'intelligibilités selon le groupe.

Condition	M(CASDO)	M(SODO)	M(AOCC)	Room level	R(CASDO)	R(SODO)	R(AOCC)	S(CASDO)	S(SODO)	S(AOCC)	C(CASDO)	C(SODO)	C(AOCC)	P(CASDO)	P(SODO)	P(AOCC)
C1 *(Use behaviours / Bad profile)*	1	1	0	0	1	1	0	-1	-1	-2	1	1	2	1	0	1
	0	0	1	-1	-1	-1	1	0	0	0	0	0	0	-1	1	1
C2 *(No behaviours / Bad profile)*	1	1	1	-1	1	1	1	-1	-2	-1	0	1	1	-1	1	0
	1	1	-1	1	1	0	0	2	-1	-1	1	1	1	-1	1	0
C3 *(Use behaviours / Good profile)*	2	2	1	1	1	1	2	1	-1	-1	1	1	1	1	1	1
	1	1	1	-1	1	-1	2	-1	1	2	1	-1	2	1	-1	2
C4 *(No behaviours / Good profile)*	1	2	1	1	1	1	1	0	0	0	1	1	1	1	1	1
	1	1	1	-2	2	-1	0	-2	-2	2	-1	-2	0	1	-1	-2

TABLE 6.13 – Résultats du questionnaire en ligne du projet ORCHESTRAA

TABLE 6.14 – Statistiques descriptives des mesures de représentativité des positions selon le groupe. Si la valeur de la moyenne d'une mesure est verte, cela indique que la valeur du groupe M est supérieure à celle du groupe $\overline{M}$, sinon elle est en rouge.

	Groupe	**N**	**Manquants**	**Moyenne**	**Médiane**	**Ecart-type**	**Minimum**	**Maximum**
R_{CASDO}	M	4	0	**0.500**	1.000	1.000	-1	1
	$\overline{M}$	4	0	1.250	1.000	0.500	1	2
R_{SODO}	M	4	0	**0.000**	0.000	1.155	-1	1
	$\overline{M}$	4	0	0.250	0.500	0.957	-1	1
R_{AOCC}	M	4	0	**1.250**	1.500	0.957	0	2
	$\overline{M}$	4	0	0.500	0.500	0.577	0	1

TABLE 6.15 – Statistiques descriptives des mesures du niveau de salle selon le groupe. Si la valeur de la moyenne du groupe $M+$ est verte, cela indique est supérieure à celle d'autres groupes $\overline{M}$, sinon elle est en rouge. Inversement, si la moyenne du groupe $M-$ est verte, cela indique qu'elle est inférieure à celle d'autres groupes, sinon elle est rouge.

	Groupe	N	Manquants	Moyenne	Médiane	Ecart-type	Minimum	Maximum
L	$M+$	2	0	**0.000**	0.000	1.414	-1	1
	$M-$	2	0	*-0.500*	-0.500	0.707	-1	0
L	$\overline{M}$	4	0	*-0.250*	0.0	1.50	-2	1

TABLE 6.16 – Statistiques descriptives des mesures liées à l'intelligibilité selon le groupe. Pour chaque mesure, nous avons trois lignes correspondant respectivement au groupe $M-$, $\overline{M}$ et $M+$. Les valeurs de la troisième ligne sont en gras. Si la valeur est verte, alors cela indique que la valeur du groupe $M+$ est supérieure aux deux autres groupes, soit $M-$ et $\overline{M}$. Les valeurs de la première ligne sont aussi colorées, mais avec une signification légèrement différente. Si la valeur est verte, cela indique que la valeur du groupe $M-$ est inférieure à celle du groupe $M+$ et $\overline{M}$, sinon elle est en rouge.

	Groupe	N	Manquants	Moyenne	Médiane	Ecart-type	Minimum	Maximum
S_{CASDO}	$M-$	2	0	-0.500	-0.500	0.707	-1	0
	$\overline{M}$	4	0	-0.250	-0.500	1.708	-2	2
	$M+$	2	0	**0.000**	0.000	1.414	-1	1
S_{SODO}	$M-$	2	0	-0.500	-0.500	0.707	-1	0
	$\overline{M}$	4	0	-1.250	-1.500	0.957	-2	0
	$M+$	2	0	**0.000**	0.000	1.414	-1	1
S_{AOCC}	$M-$	2	0	-1.000	-1.000	1.414	-2	0
	$\overline{M}$	4	0	0.000	-0.500	1.414	-1	2
	$M+$	2	0	**0.500**	0.500	2.121	-1	2
C_{CASDO}	$M-$	2	0	0.500	0.500	0.707	0	1
	$\overline{M}$	4	0	0.250	0.500	0.957	-1	1
	$M+$	2	0	**1.000**	1.000	0.000	1	1
C_{SODO}	$M-$	2	0	**0.500**	0.500	0.707	0	1
	$\overline{M}$	4	0	0.250	1.000	1.500	-2	1
	$M+$	2	0	**0.000**	0.000	1.414	-1	1
C_{AOCC}	$M-$	2	0	**1.000**	1.000	1.414	0	2
	$\overline{M}$	4	0	0.750	1.000	0.500	0	1
	$M+$	2	0	**1.500**	1.500	0.707	1	2
P_{CASDO}	$M-$	2	0	**0.000**	0.000	1.414	-1	1
	$\overline{M}$	4	0	0.000	0.000	1.155	-1	1
	$M+$	2	0	**1.000**	1.000	0.000	1	1
P_{SODO}	$M-$	2	0	**0.500**	0.500	0.707	0	1
	$\overline{M}$	4	0	0.500	1.000	1.000	-1	1
	$M+$	2	0	**0.000**	0.000	1.414	-1	1
P_{AOCC}	$M-$	2	0	**1.000**	1.000	0.000	1	1
	$\overline{M}$	4	0	-0.250	0.000	1.258	-2	1
	$M+$	2	0	**1.500**	1.500	0.707	1	2

6.3.7 Discussions

En raison du nombre de sujets, nous ne pouvons pas conclure de manière significative sur nos hypothèses. L'utilisation de la moyenne est une preuve faible comparée à un test statistique. Quoi qu'il en soit, voici notre interprétation des résultats.

H1 Nous remarquons que l'ensemble des moyennes sont supérieures à 0, strictement sauf pour $R_{SODO}(M)$ qui vaut 0. Cela suggère que l'hypothèse **H1** serait vérifiée. L'hypothèse forte $L(M) > L(\overline{M})$ serait vérifiée uniquement pour la position AOCC. Quoi qu'il en soit, il est difficile de conclure réellement avec aussi peu de sujet.

H2.1 Si nous regardons la moyenne, nous constatons que pour l'ensemble des mesures $x \in \{P, C, S\}$ et des positions $p \in \{CASDO, SODO, AOCC\}$, nous avons $x_p(M+)$ supérieure à $x_p(M-)$, sauf dans deux cas : C_{SODO} et P_{SODO}. Cela suggérait que l'hypothèse **H2.1** est plutôt vérifiée pour la position AOCC mais pas pour le SODO. Si nous excluons le CAS DO, quatre mesures sur six respectent l'hypothèse $x_p(M+) > x_p(M-)$. Si nous regardons pour l'hypothèse forte $x_p(M+) > x_p(\overline{M}) > x_p(M-)$, uniquement les mesures S_{CASDO} et S_{AOCC} la vérifie.

H2.2 Enfin, si nous regardons encore la moyenne, nous avons bien $L(M+) > L(\overline{M}) > L(M-)$. Cela voudrait dire que nous avons plutôt bien réussi à représenter les trois niveaux de salles : de performante à peu performante.

Résumé D'après nos résultats préliminaires, l'hypothèse faible de **H1** semble vérifiée, l'hypothèse faible de **H2.1** aussi, mais uniquement pour la position AOCC, mais pas du SODO et l'hypothèse forte de **H2.2** semble vérifiée aussi. Toutefois, une analyse statistique fondée sur la moyenne et avec aussi peu de sujets a peu de valeur.

Nous avons eu l'occasion de récolter le retour des sujets grâce au questionnaire, mais également après une évaluation en présentiel. En effet, une autre évaluation a été menée par les partenaires du projet ORCHESTRAA, mais sur des problématiques hors de notre thèse, notamment sur l'utilisabilité de l'environnement. Les mêmes sujets sont passés. Ils ont aussi pu interagir avec nos agents, mais uniquement avec le scénario $S+$. Quoi qu'il en soit, nous avons récolté le retour à chaud.

Ces retours mettent en lumière plusieurs points. Tout d'abord, les avis des experts divergent. Certains experts nous ont affirmé que le CAS DO n'avait rien à demander aux autres positions et qu'il devait se débrouiller tout seul. D'autres experts nous ont fait part de l'avis opposé : ils s'attendaient à ce que le SODO résume les options aux CAS DO. En discutant avec un expert, une des raisons est qu'il aurait fallu mettre en place un scénario plus complexe.

Aussi, une erreur de notre part est d'avoir fondé le comportement de nos agents sur des documents trop anciens et non sur les plus récents. Des modifications de protocoles

ont eu lieu entre-temps ; la représentativité des comportements générés en a pâti. Globalement, nous avons eu beaucoup de peine à établir un scénario et des comportements où nos personnages pouvaient générer divers comportements.

6.4 Protocole expérimental : sensibilité

Dans cette section, nous décrivons un protocole expérimental pour évaluer la sensibilité des comportements. Il s'agit d'une généralisation du protocole présenté au début de ce chapitre, section 6.1. Le but est de pouvoir l'appliquer sur n'importe quelle application.

Une génération de comportement est dite sensible, si les divers comportements induits par les modèles cognitifs opérationnalisés, se retrouvent dans le comportement de l'agent. Notre difficulté est que notre contrainte de généricité et de modularité, nous demande de pouvoir considérer n'importe quelles combinaisons de modèles cognitifs. Nous souhaitons ainsi évaluer, dans le cadre d'une application, si le méta-modèle propose une génération sensible. Par faute de temps, nous n'avons pas eu le temps d'appliquer ce protocole au projet ORCHESTRAA.

Pour établir ce protocole, nous présentons tout d'abord quelques éléments de formalisations. Ensuite, nous présentons ce que nous allons mesurer. Puis, nous présentons comment interpréter les résultats.

Formalisation

Soit C_i le comportement d'un agent i et Mc un modèle cognitif quelconque. C_i est une séquence d'actions. La notation $C_i[start, stop]$ retourne la séquence d'actions $[start, stop[$. Par exemple, $C_i[0, 2]$ retourne les deux premières actions de la séquence.

Nous supposons que l'agent i possède uniquement un effecteur, soit la possibilité de réaliser une action à la fois. D'où C_i est une séquence d'actions. Autrement, nous aurions une séquence d'actions par effecteur.

La fonction $impact(Mc, C_i)$ évalue l'impact d'un modèle Mc sur le comportement C_i de l'agent i. C'est-à-dire que pour un C_i donné et un Mc quelconque, si $impact(Mc, C_i)$ retourne :

- 0 : alors Mc n'a pas impacté le comportement de l'agent.
- 1 : alors Mc a toujours impacté le comportement de l'agent.

C'est-à-dire que pour un Mc donné, alors la valeur $impact(Mc, C_i)$ est meilleure après l'intégration de Mc. Si nous intégrons un autre Mc', il se peut que $impact(Mc, C_i)$ soit désormais inférieur.

Il se peut tout à fait que même sans l'intégration de Mc, $impact(Mc, C_i)$ renvoie une valeur non nulle.

Voici le pseudo-algorithme de $impact(Mc, C_i)$:

Algorithm 3: Pseudo-algorithme pour calculer l'impact d'un modèle cognitif sur un comportement.

```
Data: C_i
Result: times/total
times ← 0;
total ← 0;
for a_j ∈ C_i do
    if could_impact(Mc, i, C_i[0, j]) then
        if had_impact(Mc, i, C_i[0, j]) then
            times ← times + 1;
    total ← times + 1;
```

Les fonctions had_impact et $could\_impact$ sont dépendantes pour chaque Mc.

Protocole

Soit C_i le comportement de l'agent i, généré avec un ensemble de modèles cognitifs Mc quelconque. Le protocole suivi est le suivant : pour chaque modèle cognitif Mc_j, nous comparons $impact(Mc_j, C_i)$ avec et sans $Mc_j \in Mc$. Nous regardons ensuite si la valeur est plus grande / moins grande.

Nous supposons que si nous enlevions un modèle cognitif Mc_j dans Mc, alors $impact(Mc_j, C_i)$ diminue. Autrement dit, notre hypothèse est que $impact(Mc_j, C_i)$ est plus grande avec $Mc_j \in Mc$ que sans.

- Si oui pour l'ensemble des Mc_j, alors la génération de comportements est sensible **pour cet ensemble de modèles cognitifs**.
- Si cela est vérifié pour certains des Mc_j, alors nous dirons que Mc_j **est déterminant dans cet ensemble** Mc.
- Sinon, pour l'ensemble des Mc_j, alors la génération de comportements n'est pas sensible **pour cet ensemble de modèles cognitifs**.

6.5 Évaluation des performances

Dans cette section, nous comparons les performances de notre outil par rapport à d'autres outils de la communauté SMA : GAMA [Taillandier et al., 2019] et NETLOGO [Wilensky, 1999].

Nous aurions aimé comparer avec d'autres outils comme REPAST [Dubitzky et al., 2012] et MASON [Luke, 2011]. Par faute de temps, nous les ajoutons dans les perspectives / travaux restants.

Nous les comparons sur $4 \backsim 5$ scénarios :

S1 création de N agents dits « vides »,

S2 création de N agents dits « simples »,

S3 itération d'un cycle avec N agents dits « vides »,

S4 itération d'un cycle avec N agents dits « simples ».

Une itération est le passage d'un cycle de simulation, comprenant l'ensemble des cycles des modèles d'agents. Un agent dit « vide » est l'agent le plus minimaliste fourni par chaque outil. Autrement dit, nous n'ajoutons aucune donnée et comportement. Un agent dit « simple » est un agent avec quelques données : un entier, un float, un booléen et trois strings. Le booléen a une chance sur deux d'être vrai. Nous avons deux comportements possibles pour ces agents, ce qui nous donne deux variations du scénario **S4** :

S4.1 incrémentation de leur entier à chaque cycle,

S4.2 incrémentation de leur entier à chaque cycle, si leur booléen est vrai.

La raison de cette variation est pour illustrer un des principes DOD, présenté lors du chapitre 4.

Deux types d'agents Nous voulons comparer la différence de vitesse d'itération plus la complexité d'un agent augmente. Nous simulons cette augmentation en considérant deux types d'agents : l'agent le plus minimaliste fourni par chaque outil et un agent plus complet avec quelques variables en plus. Même si la complexité algorithmique reste inchangée, l'utilisation de la mémoire est significative Complexity in software [1] . Pour mettre cela en évidence, nous allons nous appuyer sur différents scénarios.

Deux types de scénarios Nous avons deux types de scénarios : un de création d'agents et un autre d'itération. Lors d'une simulation interactive, nous sommes sûrement amenés à créer des agents. Nous aimerions nous assurer que cette création soit peu coûteuse. Mais, son importance est très moindre que notre second scénario, soit la vitesse d'itération pour réaliser un cycle. Quel que soit le scénario, plus le temps est bas, mieux c'est.

1. https://www.dataorienteddesign.com/dodbook/node4.html

6.5.1 Outils comparés

Nous allons comparer des outils, présentés lors de l'état, en particulier :

— NetLogo [2] - langage de programmation et un environnement de modélisation pour le développement de système multi-agents.
— GAMA [3] - un environnement de modélisation et de simulation pour construire des simulations fondées sur agents spatialisés.

La raison est qu'il s'agit des plate-formes populaires pour la simulation d'agents, accessible et qui plus est synchrone.

À titre illustratif, nous ajoutons aussi les outils ci-dessous. Nous les comparons uniquement avec le scénario **S1**, car ils sont asynchrones : le scénario **S2** a donc moins de sens.

— CAF [4] - un framework C++ de programmation par agent.
— Jade [5] - un framework Java de programmation par agent respectant les normes FIPA.

6.5.2 Valeur seuil

Lors du chapitre 4, nous avions mentionné que lors d'une simulation interactive, des contraintes de performances sont imposées afin de proposer une expérience fluide à l'utilisateur. Dès lors, nous comparons aussi les performances vis-à-vis d'une valeur seuil.

Dans l'industrie du jeu vidéo, les simulations interactives réalisent énormément de calcul pour afficher une image. Le standard est d'afficher 30 à 60 images par seconde [Claypool and Claypool, 2009]. Nous considérons ici la valeur basse soit 30 images par secondes.

Toutefois, la majorité du temps d'un cycle est dédié au rendu de l'image. Il n'existe pas de standard, mais la communauté s'accorde à dire que 90% du temps d'un cycle est alloué au rendu, et à 10% à l'intelligence artificielle.

Autrement dit, nous fixons comme objectif que les calculs de notre modèle restent en dessous des 10%, soit $3,334$ ms.

6.5.3 Méthodes

Nous avons deux paramètres :

— Le nombre d'agents : $\{1, 2^3\,(8), 2^6\,(64), 2^9\,(512), 2^{12}\,(4096), 2^{15}\,(32768), 2^{18}\,262144, 2^{20}\,(1048576)\}$

2. http://www.netlogoweb.org/
3. https://gama-platform.org/
4. https://www.actor-framework.org/
5. https://jade-project.gitlab.io/

— Trois types d'agents : $\{empty, simple, simple\_if\}$

Une répétition est l'exécution d'un scénario sur une combinaison de paramètres, par exemple : 8 agents $simple$.

Mesures Lors de chaque répétition, nous mesurons le temps en nano-secondes lorsque possible, sinon en milli-secondes. Nous faisons ensuite la moyenne sur 5 répétitions pour chaque scénario. Hormis pour les tests de GAMA avec $2^{20} = 1048576$ agents, où nous avons réalisé uniquement 2 répétitions.

Axes graphiques Les axes de tous nos tracés graphiques (horizontales comme verticales) utilisent des échelles logarithmiques. Les résultats dans nos tableaux seront tous en nano-secondes. Dans le cas, où la précision d'un outil n'est pas à la nano-secondes, mais en milli-secondes, nous le convertissons alors en nano-secondes.

Données manquantes Certains outils cessent de fonctionner à partir d'un certain nombre d'agents ou retournent une mesure de zéro milli-secondes, du fait de la précision insuffisante. Nous omettons alors le résultat des tracés graphiques et des tableaux. C'est la raison de cellules vide dans un tableau ou une ligne avec des points manquants.

Matériels

Tous les benchmarks ont eu lieu sur la même machine avec les spécifications suivantes (*cf.* figure 6.8 obtenue grâce à lstopo [6]) :

— Windows 10 Famille 22H2
— $8 * 2808$ MHz CPU's (Intel(R) Core(TM) i7-7700HQ CPU)
— CPU Caches :
  — L1 Data $32KiB$ (x4)
  — L1 Instruction $32KiB$ (x4)
  — L2 Unified $256KiB$ (x4)
  — L3 Unified $6144KiB$ (x1)

Avant-propos

Tout d'abord, voici des précisions sur les versions et l'utilisation de chaque outil (*cf.* table 6.17) :

Dans la colonne « Outils », « Manuel » signifie que nous n'avons pas pu/su utiliser des outils de benchmarks. Dans le cas de NetLogo, il s'agit d'une plate-forme qui ne nous

6. https://www.open-mpi.org/projects/hwloc/)

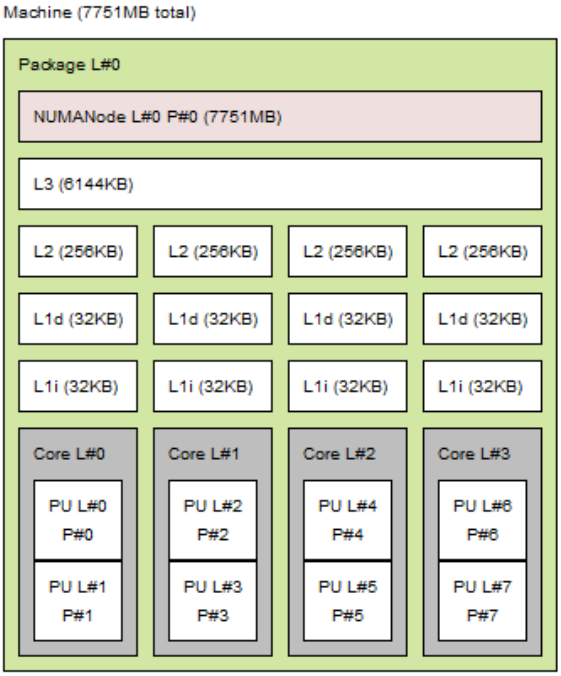


FIGURE 6.8 – Topologie du CPU utilisé pour les benchmarks obtenue grâce à l'outil lstopo.

laisse pas à notre connaissance et selon les exemples [7] fournis, mesurer le temps d'exécution, autrement que manuellement (c'est-à-dire en débutant un timer avant et après l'opération mesurée). Idéalement, nous utilisons des outils comme google benchmark [8] ou Java Microbenchmark Harness [9] .

Concernant GAMA, nous utilisons des expérimentations de type « "batch" » pour explorer les différents paramètres, en veillant à ce que les simulations tournent bien en séquentiel, plutôt qu'en parallèle (par défaut) en ajustant l'option *« Make experiments run simulations in parallel turned off »* à $true$ (soit pas de parallélisme).

De plus, comme nous avons mentionné, JADE et CAF ne sont pas strictement comparables avec les autres outils, car leurs agents fonctionnent de manière asynchrone. Aucune notion de cycles n'existe comme pour les trois autres. Ainsi, nous n'avons donc pas pris la peine de faire des cycles avec ces agents. Ils ne seront donc pas présents lors des scénarios **S3** et **S4**.

Ensuite, les résultats ci-dessous sont à considérer avec précaution. Tout d'abord, nous ne maîtrisons pas les autres outils. Nous avons tout de même essayé de trouver la manière la plus adéquate, selon l'outil, de réaliser les différents scénarios. Ensuite, chaque outil possède des ambitions et des ensembles de fonctionnalités différents. Ici, nous nous sommes contentés de comparer les outils sur des scénarios simples, notamment pour leur faisabilité. De plus, ces scénarios ont aussi été choisis pour illustrer ou non les gains

7. https://ccl.northwestern.edu/netlogo/models/models/test/benchmarks/
8. https://github.com/google/benchmark
9. https://openjdk.org/projects/code-tools/jmh/

TABLE 6.17 – Description des outils comparés

Nom	Version	Langage	Outils	Précision	Notes
OPACK	*0.5*	C++	Google benchmark	ns	
GAMA	*1.8.2*	Java	Manuel avec des expériences de type "batch"	ms	*Option : "Make experiments run simulations in parallel turned off". Changemement du heap de 4096 Mb à 6192 Mb*
Jade	*4.5.3*	Java	Java Micro-benchmark Harness	ns	*Ne supporte pas plus de 32768 agents*
Netlogo	*6.2*	Java	Manuel	ms	*Option "view updates turned off"*
CAF	*18.6*	C++	Manuel	ns	

d'une approche DOD, selon les principes décrits lors du chapitre 4.

Enfin, nous ne montrerons dans le texte que le code évalué de OPACK, notamment parce qu'il est le plus concis, grâce à l'utilisation de GOOGLE BENCHMARK. Toutefois, le code pour chaque outil utilisé lors de ces benchmarks est disponible en annexe 8.2. Le code 22 explique le squelette du code nécessaire pour google benchmark, qui facilite le benchmark d'un bout de codes sur plusieurs itérations et combinaisons de paramètres.

Code 22: Boilerplate des benchmarks avec google benchmark.

```
void BM_name(benchmark::State& state) { // Google Benchmark boilerplate
    // Setup
    for (auto _ : state) // Repeat each benchmark multiples times for every
parameter combinaisons
    {
        // ... benchmarked code
    }
}
```

6.5.4 Résultats

Voici l'ensemble des résultats menés sur nos différents scénarios. Nous commentons les résultats lors de la prochaine sous-section.

Scénario 1 : création d'agents « vides »

Code 23: Code-source d'OPACK pour la création de N agents « vides »

```
void BM_t_spawn_n_empty_agents(benchmark::State& state){
    auto world = opack::create_world();
    for (auto _ : state) {
        opack::spawn_n<Agent>(state.range(0));
    }
}
```

```
// ...
```

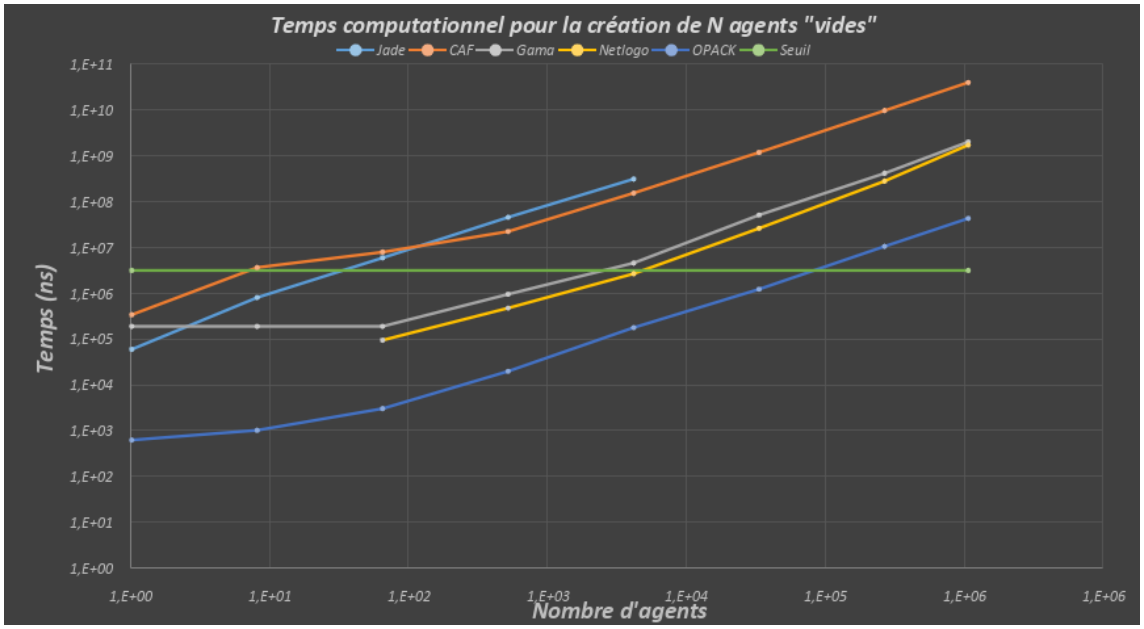


FIGURE 6.9 – Comparaison de la vitesse de création d'agents « vides ».

TABLE 6.18 – Temps computationnel pour la création de N agents « vides » (en ns)

N	1	8	64	512	4096	32768	262144	1048576
Jade	63023	848349	6213064	47487518	324039940			
CAF	357912	3830000	8290000	23303800	160032000	1230820000	10011800000	41079200000
Gama	200000	200000	200000	1000000	4800000	53200000	431000000	2081500000
Net-logo			100000	500000	2800000	27300000	291300000	1779300000
OPACK	667	1090	3230	20999	187640	1288602	11038434	44900000

TABLE 6.19 – Tableau des facteurs multiplicatifs du temps mis par l'outil vis-à-vis d'OPACK lors de créations de N agents « vides ». Un résultat en vert indique que OPACK est X fois plus rapide que l'outil comparé.

N	1	8	64	512	4096	32768	262144	1048576
Jade	**94**	**778**	**1924**	**2261**	**1727**			
CAF	**537**	**3514**	**2567**	**1110**	**853**	**955**	**907**	**915**
Gama	**300**	**183**	**62**	**48**	**26**	**41**	**39**	**46**
Netlogo			**31**	**24**	**15**	**21**	**26**	**40**

Scénario 2 : création d'agents "simples"

Code 24: Benchmark d'OPACK pour la création de N agents « simples ».

```
void BM_t_spawn_n_simple_agents(benchmark::State& state) {
    auto world = opack::create_world();
    auto prefab = opack::init<SimpleAgent>(world)
        .override<Counter>()
        .override<Value1>()
        .override<Tag>()
        .override<String2>()
        .override<String3>()
        .override<String4>()
        ;
    for (auto _ : state) {
        opack::spawn_n(prefab, state.range(0));
    }
}
```

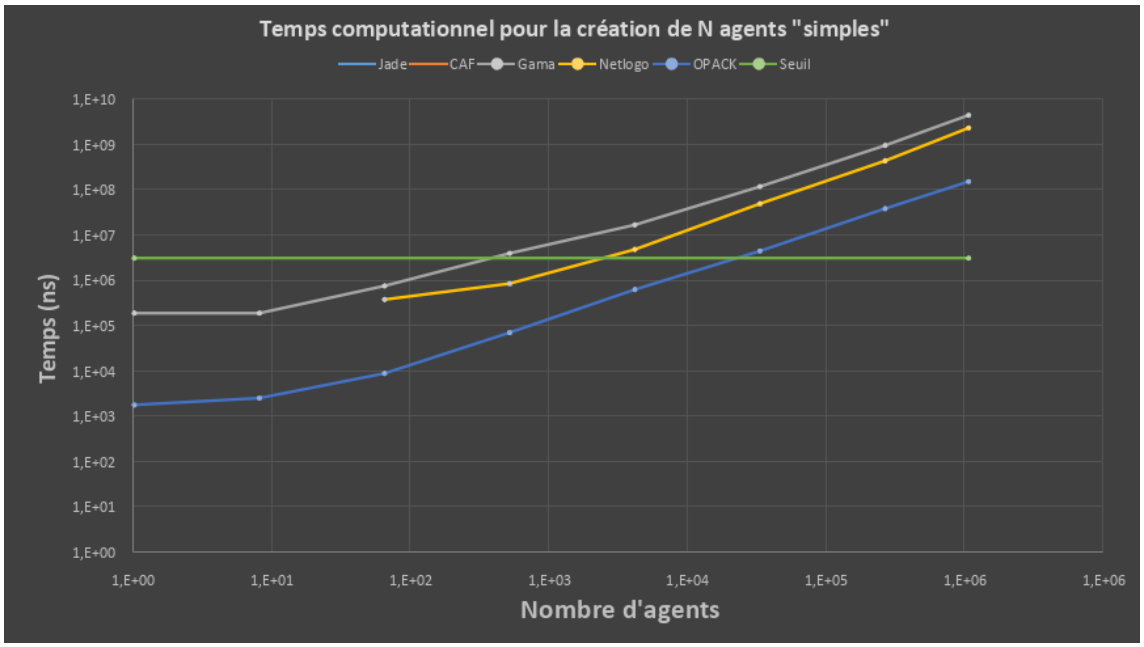


FIGURE 6.10 – Comparaison de la vitesse de création d'agents « simples ».

TABLE 6.20 – Temps computationnel pour la création de N agents « simples » (en ns)

N	1	8	64	512	4096	32768	262144	1048576
Gama	200000	200000	800000	4200000	17800000	124800000	1011600000	4703000000
Net-logo			400000	900000	5100000	52300000	462800000	2456400000
OPACK	1883	2665	9367	74559	665523	4710074	40778422	160747700

TABLE 6.21 – Tableau des facteurs multiplicatifs du temps mis par l'outil vis-à-vis d'OPACK lors de la création de N agents « simples ». Un résultat en vert indique que OPACK est X fois plus rapide que l'outil comparé.

N	1	8	64	512	4096	32768	262144	1048576
Gama	**106**	**75**	**85**	**56**	**27**	**26**	**25**	**29**
Netlogo			**43**	**12**	**8**	**11**	**11**	**15**

Scénario 3 : itération d'agents « vides »

Code 25: Benchmark d'OPACK pour l'itération de N agents « vides ».

```
void BM_t_loop_with_n_empty_agents(benchmark::State& state){
    auto world = opack::create_world();
    auto prefab = opack::entity<opack::Agent>(world);
    opack::spawn_n<opack::Agent>(world, static_cast<size_t>(state.range(0)));
    world.system().term(flecs::IsA, prefab).each([](flecs::entity e){ });
    for ([[maybe_unused]] auto _ : state){
        opack::step(world);
    }
}
```

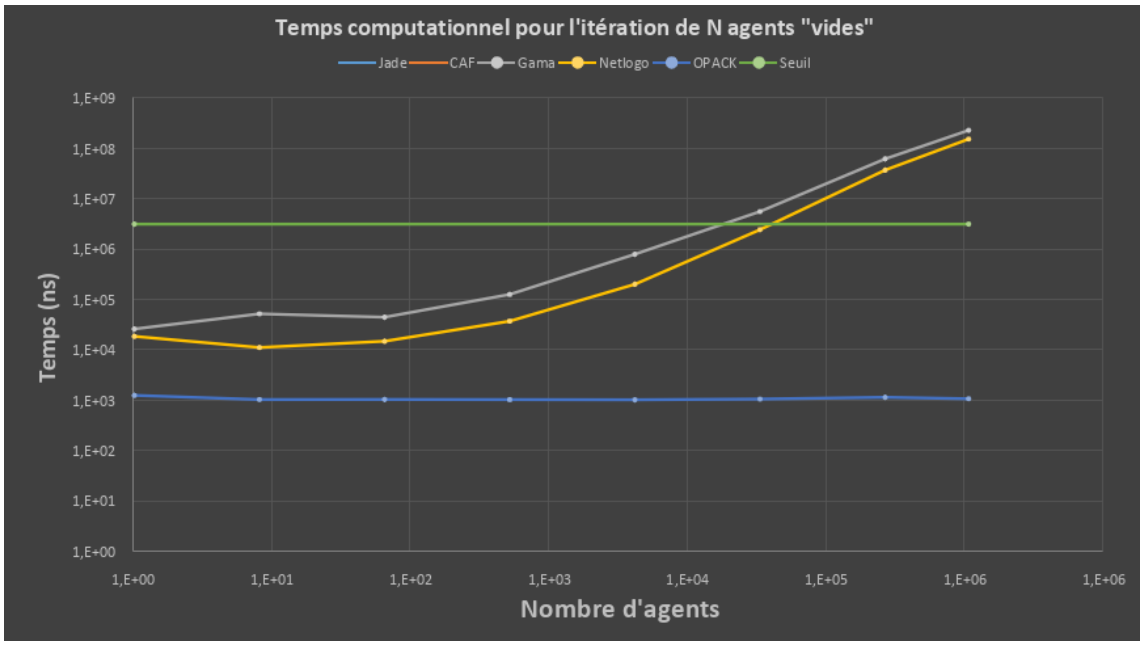


FIGURE 6.11 – Comparaison de la vitesse d'itération avec n agents « vides ».

TABLE 6.22 – Temps computationnel pour l'itération de N agents « vides » (en ns)

N	1	8	64	512	4096	32768	262144	1048576
Gama	27343.75	54687.5	46875	132812.5	832031.25	5867187.5	65085937.5	240335937.5
Net-logo	19531.25	11718.75	15625	39062.5	210937.50	2558593.75	39039062.5	160964843.75
OPACK	1316	1085	1091	1084	1075	1111	1204	1127

TABLE 6.23 – Tableau des facteurs multiplicatifs du temps mis par l'outil vis-à-vis d'OPACK lors de l'itération de N agents « vides ». Un résultat en vert indique que OPACK est X fois plus rapide que l'outil comparé.

N	1	8	64	512	4096	32768	262144	1048576
Gama	**21**	**50**	**43**	**123**	**774**	**5281**	**54058**	**213253**
Netlogo	**15**	**11**	**14**	**36**	**196**	**2303**	**32424**	**142826**

Scénario 4.1 : itération d'agents « simples » sans condition

Code 26: Benchmark d'OPACK pour l'itération de N agents « simples ».

```
void BM_t_loop_with_n_simple_agents(benchmark::State& state) {
    auto world = opack::create_world();
    const auto prefab = opack::init<SimpleAgent>(world)
        .override<Counter>()
        .override<Value1>()
        .override<Tag>()
        .override<String2>()
        .override<String3>()
        .override<String4>()
        ;
    world.system<Counter>().each([](flecs::entity e, Counter& counter)
        { counter.value++; });
    opack::spawn_n<SimpleAgent>(world, static_cast<size_t>(state.range(0)));
    for ([[maybe_unused]] auto _ : state){
        opack::step(world);
    }
}
```

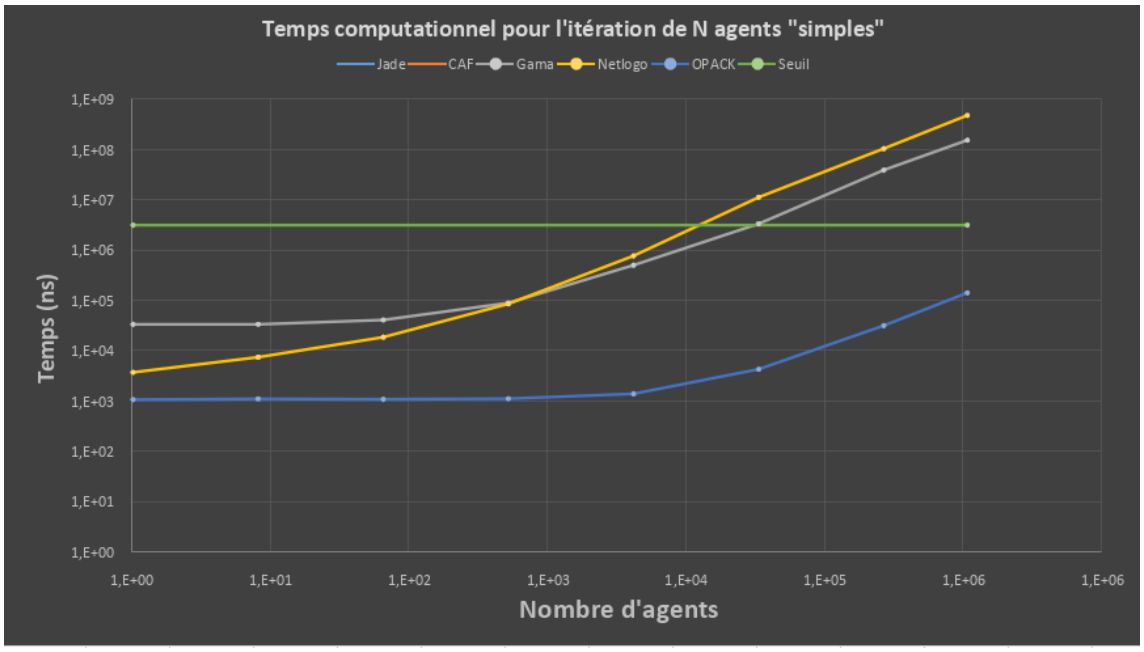


FIGURE 6.12 – Comparaison de la vitesse d'itération avec n agents « simples » sans condition.

TABLE 6.24 – Temps computationnel pour l'itération de N agents « simples » (en ns)

N	1	8	64	512	4096	32768	262144	1048576
Gama	35156.25	35156.25	42968.75	93750	527343.75	3511718.75	41125000	161449218.75
Net-logo	3906.25	7812.5	19531.25	89843.75	808593.7499999	11796875	109722656.2	501132812.50
OPACK	1121	1162	1142	1170	1463	4508	33197	148888

TABLE 6.25 – Tableau des facteurs multiplicatifs du temps mis par l'outil vis-à-vis d'OPACK lors de l'itération de N agents « simples ». Un résultat en vert indique que OPACK est X fois plus rapide que l'outil comparé.

N	1	8	64	512	4096	32768	262144	1048576
Gama	**31**	**30**	**38**	**80**	**360**	**779**	**1239**	**1084**
Netlogo	**3**	**7**	**17**	**77**	**553**	**2617**	**3305**	**3366**

Scénario 4.2 : itération d'agents « simples » avec une condition

Code 27: Benchmark d'OPACK pour l'itération de N agents « simples » avec un if.

```
void BM_t_loop_with_simple_agents_and_if(benchmark::State& state) {
    auto world = opack::create_world();
    const auto prefab = opack::init<SimpleAgent>(world)
        .override<Counter>()
        .override<Value1>()
        .override<Tag>()
        .override<String2>()
        .override<String3>()
        .override<String4>()
        ;
    opack::spawn_n(prefab, static_cast<size_t>(state.range(0) / 2));
    world.filter<Tag>().each([](flecs::entity e, Tag tag){e.remove<Tag>();});
    opack::spawn_n(prefab, static_cast<size_t>(state.range(0) / 2));
    world.system<Counter, Tag>().each([](flecs::entity e, Counter& counter, Tag tag)
        { counter.value++; }
    );
    for ([[maybe_unused]] auto _ : state){
        opack::step(world);
    }
}
```

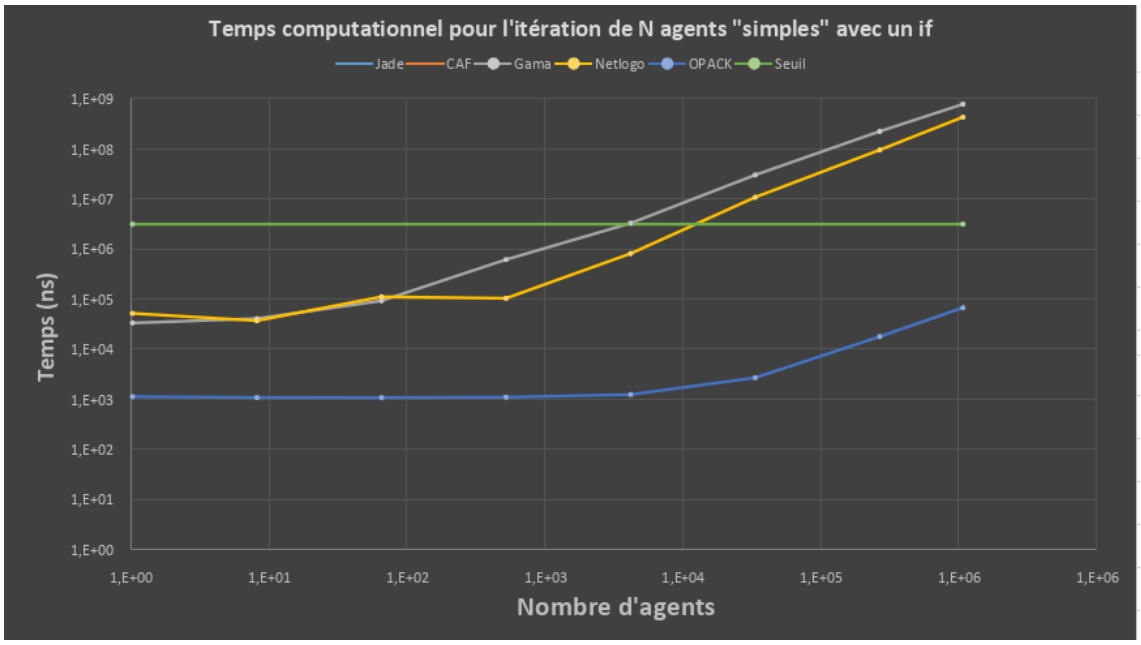


FIGURE 6.13 – Comparaison de la vitesse d'itération avec N agents « simples » avec une condition.

TABLE 6.26 – Temps computationnel pour l'itération de N agents « simples » avec un if (en ns).

N	1	8	64	512	4096	32768	262144	1048576
Gama	35156.25	42968.75	97656.25	648437.5	3457031.25	31707031.25	235648437.5	820167968.75
Net-logo	54687.5	39062.5	117187.50	109375	855468.75	11398437.5	100339843.7!	452882812.5
OPACK	1189	1142	1138	1157	1309	2832	18783	70873

TABLE 6.27 – Tableau des facteurs multiplicatifs du temps mis par l'outil vis-à-vis d'OPACK lors de l'itération de N agents « simples » avec un if. Un résultat en vert indique que OPACK est X fois plus rapide que l'outil comparé.

N	1	8	64	512	4096	32768	262144	1048576
Gama	**30**	**38**	**86**	**560**	**2641**	**11196**	**12546**	**11572**
Netlogo	**46**	**34**	**103**	**95**	**654**	**4025**	**5342**	**6390**

6.5.5 Discussions

Sur tous les scénarios, OPACK est plus rapide que les autres outils et parfois de plusieurs ordres de grandeur. Nous rappelons que ces résultats sont à prendre avec précaution. Chaque plate-forme procure différentes fonctionnalités. Quant à OPACK, l'outil est loin d'avoir la maturité des autres outils.

Scénario de créations d'agents

Si nous regardons tout d'abord, les scénarios de création d'agents, sans compter OPACK, l'outil le plus rapide est NetLogo. Si nous comparons OPACK avec NetLogo lors de la création de 2^{20} agents « vides », alors nous avons :

1. **Scénario 1** : 40x plus rapide que NetLogo.
2. **Scénario 2** : 15x plus rapide que NetLogo.

Nous remarquons qu'OPACK est constamment plus rapide que les autres, avec un ordre de grandeur de différence avec NetLogo. Une des raisons est que nous utilisons une technique de Batching pour pré-allouer l'ensemble de la mémoire. Toutefois, l'intérêt de ce scénario est limité. La création d'agents est une étape ponctuelle. Nous sommes plus concernés par les performances lors de l'itération.

Scénario d'itérations avec des agents « vides »

Lors de ce scénario, nous mesurons la vitesse d'itération de la simulation, soit le passage d'un cycle, en fonction du nombre d'agents « vides ». Nous remettons les courbes obtenues lors de l'itération de N agents « vides » :

Dans ce scénario, avec 2^{20} agents, OPACK est 213 253 fois plus rapide que Gama. Pour mettre les choses en perspectives, s'il faut 3 jours pour Gama pour aller sur la lune, il ne faut que 1.22 secondes pour OPACK.

Les raisons derrière cette performance sont multiples. Une des explications est une propriété du méta-modèle OPACK, suivi par notre implémentation : nous posons le moins d'hypothèses sur le modèle d'agent. Donc, nous n'imposons aucun modèle d'agent. Dès lors, les agents sont, par défaut, juste une information sans donnée, distinguables entre eux par leur identifiant. Autrement dit, si nous réemployons le terme introduit par [Picault and Sicard, 2020], alors nos agents sont juste une information. C'est au modélisateur de décider du paradigme ou du modèle d'agent à appliquer. Une autre raison est que l'utilisation de ECS améliore la localité (*cf.* section sur les principes DOD du chapitre 4).

Ce scénario illustre, pour les autres approches, que juste l'utilisation de la notion d'agents induit un coût computationnel non négligeable, même lorsque l'agent ne fait rien. Contrairement à notre approche, un agent impacte l'itération d'un cycle surtout lorsqu'il doit réaliser quelque chose.

Dans la section suivante, nous illustrons comment notre implémentation permet aussi de gagner des performances dans le cas de certaines dynamiques.

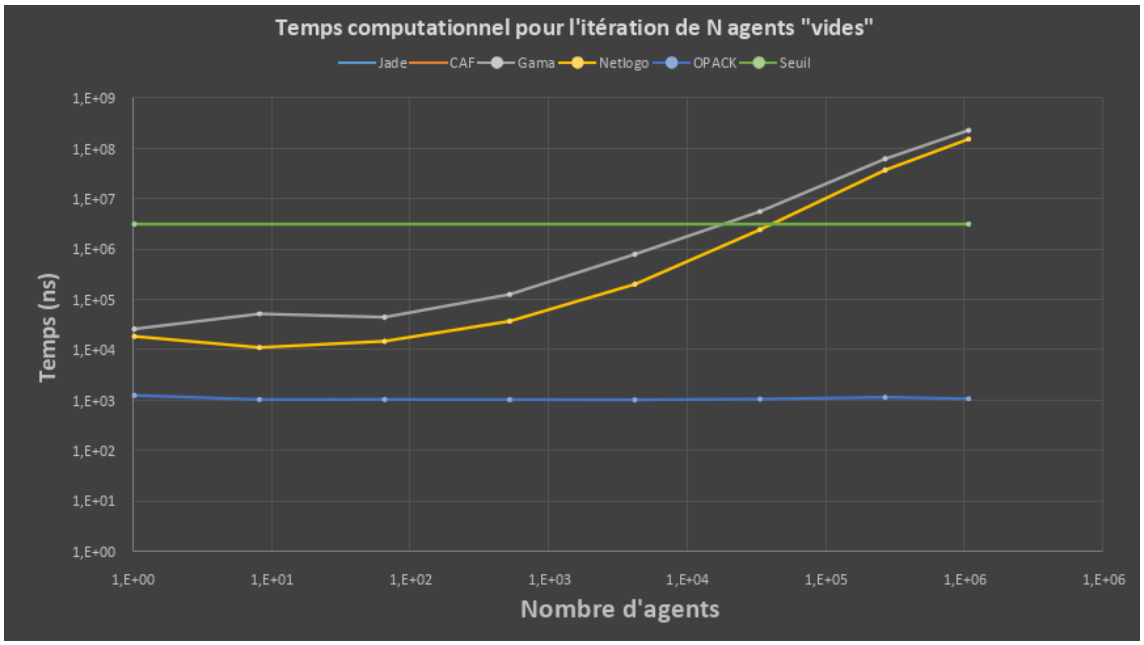


FIGURE 6.14 – Comparaison de la vitesse d'itération avec n agents « vides ».

Scénarios d'itérations avec des agents « simples »

Lors du chapitre 4 et de la présentation de DOD, nous avions introduit quelques principes pour gagner en performance. L'un de ces principes mentionnait l'impact des « ratés » lorsqu'un cache prédit la mauvaise branche du flux d'exécution d'un programme, avec un if par exemple. Ici, nous illustrons comment l'application de ce principe permet de gagner des performances. Nous avons deux sous-scénarios : un où tous les agents simples incrémentent un compteur, un autre où tous les agents le font, uniquement si une condition est vérifiée.

Lors du premier scénario, soit celui où tous les agents « simples » incrémentent leur compteur, nous obtenons, avec 2^20 agents, les résultats suivants :

— OPACK est 1084 fois plus rapide que GAMA ;
— OPACK est 3366 fois plus rapide que NETLOGO ;

Ce scénario va nous servir de comparaison pour le second scénario. Commençons par comparer pour chaque approche, le coût de l'ajout de la condition lors de l'incrémentation, avec 2^20 agents :

— GAMA est 500% plus lent ;
— NETLOGO est 10% plus rapide.

NETLOGO est légèrement plus rapide. Toutefois, GAMA a été fortement affecté par l'ajout de condition. Toutefois, ce qui nous intéresse est de comparer à quel point notre implémentation a gagné en performance, comparé aux autres approches :

— OPACK est désormais 11572 fois plus rapide que GAMA ;
— OPACK est désormais 6390 fois plus rapide que NETLOGO.

Nous constatons que juste en ajoutant une condition, les écarts de performances s'élargissent. Comparé à NETLOGO, l'écart a été augmenté de 1.90 fois. Tandis que pour GAME, cet écart s'est agrandi 10.68 fois. Une explication possible de ce gain est que nous itérons uniquement sur les agents qui possède la condition $vrai$, comme nous l'avons vu lors du paragraphe 4.3.2 sur le traitement existentiel.

Enfin, comparé aux autres approches, notre implémentation reste toujours en dessous de la valeur de seuil lors des scénarios d'itérations, même à 2^{20} agents, soit plus d'un million d'agents. Tandis que les autres approches passent aux dessus de la valeur seuil aux alentours de $2^{15} = 32768$ agents.

À propos du multi-threading Jusqu'ici, nous n'avons pas mentionné la notion de multi-threading. Les résultats de notre implémentation ne font pas appel à du multi-threading. Cependant, un apport de ECS est de faciliter sa mise en place. Grâce à l'utilisation de la bibliothèque FLECS [10] , nous pouvons choisir quels bouts de logique, soit quelles opérations dans le méta-modèle OPACK, utilisent ou non du multi-threading. Par exemple, lors du scénario **S4.1**, nous pouvons indiquer que l'incrémentation employe du multi-threading. Pour cela, il suffit de rajouter dans notre code ces deux éléments :

Code 28: Benchmark d'OPACK pour l'itération de N agents "simples" avec du multi-threading.

```
    void BM_t_loop_with_n_simple_agents_mt(benchmark::State& state) {
    auto world = opack::create_world();
    world.set_threads(8) // Here we indicate how many threads can be used.
    const auto prefab = opack::init<SimpleAgent>(world)
        .override<Counter>()
        .override<Value1>()
        .override<Tag>()
        .override<String2>()
        .override<String3>()
        .override<String4>()
        ;
    world.system<Counter>()
        .multi_threaded() // Here we indicate that this system can be run with
multiple threads.
        .each([](flecs::entity e, Counter& counter)
        { counter.value++; });
    opack::spawn_n<SimpleAgent>(world, static_cast<size_t>(state.range(0)));
    for ([[maybe_unused]] auto _ : state){
        opack::step(world);
    }
}
```

Cependant les performances obtenues sont moins bonnes que sans. Plus le nombre d'agents augments, moins l'écart est important. La raison est que le multi-threading a du sens uniquement lorsque nous sommes bornées par la vitesse du CPU et non par la mémoire. Pouvoir indiquer quel processus peut utiliser du multi-threading permet de l'appliquer judicieusement.

10. https://github.com/SanderMertens/flecs

Chapitre 7

Conclusions et perspectives

Contents

Dans ce chapitre, nous présentons les conclusions de nos travaux, ainsi que des perspectives à court, moyen et long terme.

7.1 Conclusions

Cette thèse s'inscrit dans le domaine des systèmes multi-agents et la génération de comportements. En particulier, nous avons travaillé sur la génération de comportements induits par des modèles cognitifs issus de la littérature des SHS. Nous avons proposé une approche modulaire, pour faciliter l'intégration de tels modèles au sein d'un système multi-agents. Notre approche est indépendante du modèle d'agent choisi, ce qui permet une plus grande généricité et l'intégration modulaire des modèles cognitifs. Nous avons ensuite proposé un mécanisme de génération de comportements à partir des modèles cognitifs, de telle sorte que l'ajout d'un modèle impacte le comportement généré selon ses propriétés. Les comportements ainsi obtenus sont intelligibles, c'est-à-dire qu'un observateur externe peut les identifier à partir des actions effectuées par les agents. Il est également possible d'extraire du système les causes qui ont amené à générer ce comportement pour produire des explications.

La conception et la mise en œuvre de notre approche a nécessité d'aborder différentes questions scientifiques.

État de l'art L'étude de la littérature nous a permis de montrer que les approches les plus pertinentes vis-à-vis de nos objectifs étaient PMFServ [Silverman et al., 2006b, Silverman et al., 2006a], BEN [Bourgais et al., 2021] et le méta-modèle de Bernon [Bernon et al., 2005]. Pour PMFServ, un aspect contradictoire avec nos objectifs est l'expertise nécessaire pour intégrer un modèle cognitif : le couplage fort des modèles entre eux complexifie la tâche. BEN est à notre connaissance l'architecture la plus récente, avec le plus grand nombre de dimensions (social, affection, normes). Les différents modèles cognitifs sont liés entre eux grâce au modèle de personnalité OCEAN, ce qui permet de réduire le nombre de paramètres à régler pour le modélisateur. Cette architecture est aussi modulaire où de nombreux processus sont optionnels. Toutefois, son utilisation suppose l'utilisation d'un modèle d'agent fondée sur BDI [Bratman, 1987]. Ensuite, les travaux de Bernon [Bernon et al., 2005] consistent à proposer un méta-modèle, recouvrant les concepts employés couramment dans différents modèles d'agents. Cependant, leur méta-modèle se focalise uniquement sur la partie descriptive d'un modèle d'agent et des concepts employés. La partie dynamique était omise, car jugée peu pertinente : la dynamique est trop dépendante de l'application. Nous avons ainsi conclu qu'il n'était pas pertinent de proposer un modèle d'agent « "One size fits all" ». Ce sentiment est partagé par Bernon et al. [Bernon et al., 2005] et Silverman [Silverman et al., 2006b].

Contribution 1 : Méta-modèle OPACK C'est pourquoi dans cette thèse, nous proposons un méta-modèle, nommé OPACK, pour intégrer des modèles cognitifs au sein de modèles d'agents, tout en faisant un minimum d'hypothèses sur leur nature. OPACK abstrait un modèle d'agent autour de cinq concepts : **O**peration, **P**erception, **A**ction, **C**aractéristique et **K**nowledge (connaissances en français). Nous dissocions la dynamique du modèle d'agent représenté par **O**, de son état représenté par **P**, **A**, **C** et **K**. Ces cinq concepts ont été identifiés comme les cinq aspects d'un modèle d'agent qu'un modèle cognitif peut impacter. Ils sont tous optionnels, mais aussi composables, pour permettre la modularité, généricité et donc simplifier l'intégration de modèles cognitifs.

La dynamique du modèle d'agent est décrite par un enchaînement d'opérations, sous forme d'un graphe acyclique d'opérations, désigné comme flux d'opérations. Chaque opération est une fonction avec des entrées et des sorties. Les liens entre les opérations sont soit des liens de précédences, soit le passage d'une valeur d'une sortie d'une opération vers l'entrée d'une autre opération. Ensuite, chaque opération est libre de modifier l'état de l'agent. Les flux d'opérations sont modulaires et peuvent être attribués de manière dynamique. C'est-à-dire que les flux d'opérations d'un agent peuvent être modifiés lors de la simulation pour s'adapter à la situation.

Contribution 2 : stratégie de choix par graphes d'influences et de préférences D'autre part, le fonctionnement des opérations est aussi modifiable modulairement. Une stratégie indique comment calculer les valeurs de sorties de l'opération, selon un ensemble de fonctions-impacts, issues de modèles cognitifs. Par exemple, nous proposons une stratégie de choix, appliquée à la sélection d'action : les graphes d'influences et de préférences. Les fonctions-impacts des modèles cognitifs sont des fonctions qui retournent des influences positives ou négatives ou nulles, vers les actions envisageables

(en entrée de l'opération). Si les influences ne permettent pas de déterminer un vainqueur sans indécision (plusieurs actions avec le score le plus élevé), nous introduisons aussi un mécanisme de préférences. Le modélisateur peut indiquer des préférences entre les modèles cognitifs. Parmi les actions candidates, nous sélectionnons l'action la plus préférée. Cette stratégie a été proposée comme une méthode intelligible et sensible pour la génération de comportements.

Contribution 3 : implémentation ECS des SMA Enfin, nous avons proposé une implémentation de notre méta-modèle, soit un outil de simulation d'agents virtuels. Contrairement aux autres approches similaires qui emploient une approche de POO, notre implémentation est une bibliothèque (et non une plate-forme), qui applique les principes DOD, notamment avec l'utilisation de ECS.

Évaluation informatique de la sensibilité Nous avons ensuite réalisé plusieurs évaluations. La première est une évaluation informatique de notre approche pour vérifier que les comportements obtenus sont correctement impactés par les modèles cognitifs. Nous souhaitons donc tester la sensibilité de notre génération de comportements avec un ensemble de modèles cognitifs. Nous avons montré que les modèles cognitifs ont un impact net sur la sélection d'action.

Évaluation utilisateur de la représentativité et de l'intelligibilité La seconde est une évaluation utilisateur pour évaluer l'intelligibilité et la représentativité des comportements générés dans le cadre du projet VICTEAMS. Nous simulions des médecins secouristes en situation de crise. Les comportements ont été jugés peu représentatifs : le stress, impliqué par la situation, a été mal retransmis dans notre scénario. Toutefois, les sujets ont bien su différencier les comportements de nos agents (hormis ceux liés au stress).

Évaluation utilisateur de la représentativité et de l'intelligibilité dans le cadre du projet ORCHESTRAA Ensuite, nous avons appliqué nos travaux au projet ORCHESTRAA afin de simuler les opérateurs d'un centre de commandement aérien. Nous avons donc mené une autre évaluation utilisateur, semblable à la précédente, mais cette fois dans le cadre de ce projet. Les résultats préliminaires, qui devront être confirmés par une étude avec plus de sujets, suggèrent que les comportements sont plutôt représentatifs, et que les sujets ont pu identifier correctement le profil de nos agents.

Évaluation de performances de notre implémentation Enfin, nous avons réalisé une évaluation de performances de notre approche. Nous comparons les résultats à d'autres approches de la communauté SMA : NetLogo [Wilensky, 1999] et Gama [Taillandier et al., 2016]. Nous constatons que notre outil est constamment plus rapide que les autres approches et parfois de plusieurs ordres de grandeur, dans certains scénarios. Toutefois, ces résultats sont à prendre avec précaution : notre bibliothèque n'a pas la maturité et les capacités des autres outils. Qui plus est, le domaine visé n'est pas le

même : au lieu de proposer une plate-forme, nous proposons une bibliothèque pour être intégrée dans des applications de simulations interactives.

7.2 Limites

Nos travaux ont permis de poser les bases d'un méta-modèle pour intégrer des modèles d'agents et des modèles cognitifs, notamment le raisonnement au travers des flux d'opérations. Toutefois, nous identifions plusieurs limites :

Le modélisateur doit définir beaucoup de choses

À lui seul, le méta-modèle ne génère aucun comportement, mais fournit des outils au modélisateur pour composer son agent et son/ses modèles, puis intégrer modulairement d'autres modèles cognitifs.

Le désavantage est que c'est à la charge du modélisateur de définir correctement ses modèles d'agents et cognitifs pour obtenir les comportements souhaités avec les outils à disposition.

Manque de capacités/d'outils

Une autre limite est justement que le méta-modèle n'offre pas suffisamment d'outils pour permettre au modélisateur de composer rapidement ses agents. Pour le moment, uniquement les concepts de base sont proposés. La difficulté est de rajouter des concepts tout en respectant les propriétés du méta-modèle.

Un aspect que nous aimerions développer est de pouvoir mettre à jour une valeur continue au sein d'une opération, tout en respectant nos propriétés : quelle stratégie serait envisageable ? Nous pensons qu'il est possible d'adapter la stratégie de choix par graphes d'influences et de préférences pour choisir une tendance de l'évolution de la valeur, selon l'ensemble des fonctions-impacts.

Manque d'expressivité des flux d'opérations

Actuellement, les flux d'opérations sont représentés par des graphes acycliques d'opérations. Lorsque nous avons construit des flux, nous sommes retrouvés à déplorer la présence d'opérations pour contrôler le flux d'opérations. Aussi, nous aimerions nous rapprocher plus des propriétés de la programmation par flux de Morrison [Morrison, 1994], en rajoutant notamment la possibilité que certaines opérations puissent fonctionner de manière asynchrone.

7.3 Perspectives

Dès lors, nous envisageons plusieurs améliorations, d'abord pour circonvenir aux limites actuelles, mais aussi à des cas auxquels nous n'avons pas forcément apporté de solutions. Nous présentons d'abord des perspectives à court terme, à moyen terme, puis à long terme.

7.3.1 Perspectives à court terme

Perception

Lors de la perception, nous avons défini que les sens indiquaient les caractéristiques perceptibles d'une entité perçue. Une subtilité est présente lorsqu'une caractéristique possède une arité supérieure à zéro : peut-on percevoir les arguments des arités ?

Illustrons cette question avec l'exemple suivant. Nous avons :

- trois agents $i, j, k \in Agt^3$;
- une caractéristique d'arité 0, nommée $Stress \in C$;
- une caractéristique d'arité 1, nommée $Friend \in C$;
- un sens $s \in Sen$ quelconque capable de percevoir les deux caractéristiques précédentes.

Supposons désormais que j est ami avec k. Nous avons donc $Friend(j, k)$. Est-ce que percevoir j avec le sens quelconque s permet de percevoir $Friend(j, k)$ ou $Friend(j, *)$?

Actuellement, nous supposons qu'uniquement les identifiants directement perçus sont identifiables comme arguments. La figure 7.1 illustre ce fonctionnement.

Toutefois, nous pouvons considérer que certains sens peuvent percevoir les arguments, même si nous sommes dans le cas 2, plutôt que 3. Autrement dit, nous pourrions ne pas imposer un unique mode de fonctionnement pour la perceptibilité des arguments.

Dès lors, une solution envisageable est d'employer encore une fois la notion de caractéristiques. Nous pourrions ajouter une caractéristique d'arité 0, nommée $Transitive$ par exemple, pour indiquer si les arguments d'une arité sont perceptibles.

- Si cette caractéristique est appliquée à un sens, alors le sens serait capable de percevoir tous les arguments.
- Ou alors uniquement l'ensemble des arguments possédant cette caractéristique seraient perceptibles.

Par exemple, si l'agent k possède la caractéristique $Transitive$, alors tous les sens qui perçoivent une caractéristique avec k comme argument le verront aussi. Cependant, cela n'est peut-être pas voulu pour tous les sens. Nous pourrions alors envisager de transformer la caractéristique $Transitive$ d'arité 0 à une arité 1 pour spécifier pour quels sens cela fonctionne. Inversement, nous pouvons aussi ce raisonnement si la caractéristique $Transitive$ est appliquée à un sens.

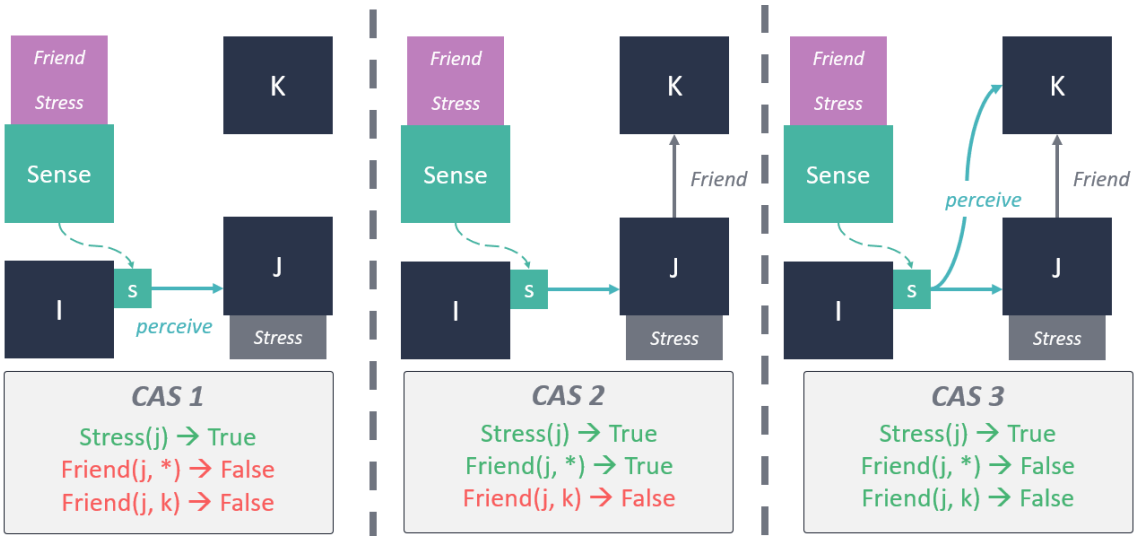


FIGURE 7.1 – Dans le cas 1, i perçoit j et j possède uniquement une caractéristique de stress. Donc i perçoit, de j, uniquement $Stress(j)$. $Friend(j, *)$ et $Friend(j, k)$ sont faux, car j, dans le cas 1, n'a pas d'ami. Ensuite, si nous considérons une caractéristique d'arité supérieure à 0, comme la relation $Friend$. Dans le cas 2, j a désormais une relation $Friend$ avec k. i perçoit désormais la relation $Friend(j, *)$, mais pas $Friend(j, k)$, car k n'est pas perçu par i. Dans le cas 3, i perçoit désormais k. Donc, il peut percevoir $Friend(j, k)$, ce qui n'était pas le cas avant.

Action

Nous pourrions aussi faciliter la mise en œuvre de modèles d'agents en complétant les fonctionnalités du méta-modèle vis-à-vis des actions.

Mise en pause Notamment, si nous considérons le cas d'étude de HPTS++ [Lamarche and Donikian, 2002] : un individu fume, boit et lit un livre par alternance. Actuellement, notre méta-modèle ne considère pas qu'une action puisse être mise en pause. Elle est uniquement interruptible. L'ajout de la mise en pause permettrait de garder le contexte d'une action, au fur et à mesure des reprises. Dans le cas d'études ci-dessus, cela pourrait faciliter la reprise du livre ou du verre d'eau : nous n'aurions pas besoin de spécifier les paramètres à nouveau.

Plusieurs effecteurs Ensuite, toujous en considérant le cas d'étude de HPTS++, nous remarquons que l'individu tient le livre avec parfois une ou deux mains. L'une est enlevée pour réaliser d'autres actions comme la prise d'un verre d'eau. Dès lors, les mains sont considérées comme deux effecteurs.

Cependant, notre méta-modèle permet actuellement d'indiquer un effecteur par action. Nous pourrions assez facilement l'étendre pour permettre de requérir plusieurs effecteurs pour réaliser une action. Ou au contraire, indiquer que certains effecteurs sont

optionnels : « il faut au moins une main pour tenir le livre, mais je peux le tenir avec deux mains. »

Opération

Toujours dans l'optique de proposer plus d'outils et de capacités au modélisateur, nous aimerions aussi améliorer le concept d'opérations.

Tout d'abord, pour faciliter la réutilisation, nous souhaiterions permettre qu'un flux soit composé d'autres flux. Ensuite, nous aimerions ajouter la possibilité de déterminer automatiquement les opérations à ajouter dans un flux par un chaînage arrière. Par exemple, si dans un flux, nous ajoutons une opération nécessitant des actions, nous pourrions automatiquement trouver les opérations permettant de satisfaire ces contraintes d'entrées, soit des actions.

Ensuite, nous disposons de peu de stratégies pour le moment. Pour le moment, nous n'avons réellement que la stratégie de choix par graphes d'influences et de préférences, appliquée à la sélection d'action. Nous pourrions notamment réemployer cette stratégie pour d'autres opérations, comme la mise à jour d'une valeur : tel modèle indique que la nouvelle valeur devrait être x, *etc.* . Quoi qu'il en soit, nous aimerions proposer plus de stratégies pour offrir plus de possibilités au modélisateur.

Enfin, nous aimerions augmenter les capacités du flux en ajoutant notamment de nouveaux rôles pour contrôler le flux. Surtout, nous aimerions proposer plusieurs flux ou opérations, prêts à être utilisés. Par exemple, nous pourrions disposer d'un ensemble d'opérations pour mettre en place un modèle BDI [Bratman, 1987] que le modélisateur pourrait réemployer. Idéalement, nous aimerions procurer un ensemble de briques à disposition des modélisateurs.

Implémentation

Enfin, nous envisageons plusieurs perspectives au niveau de la bibliothèque et de l'implémentation du méta-modèle.

Interopérabilité Selon le dictionnaire du Larousse, l'interopérabilité en informatique signifie la chose suivante :

Citation 18 - Interopérabilité

« Capacité de matériels, de logiciels ou de protocoles différents à fonctionner ensemble et à partager des informations »

Éditions Larousse. Dictionnaire de français Larousse — larousse.fr. https://www.larousse.fr

Autrement dit, l'interopérabilité appliquée ici serait de pouvoir décrire un méta-modèle sans imposer le langage informatique sous-jacent. Plusieurs pistes sont envisageables.

La première serait d'utiliser le langage C au lieu du C++. Ce langage est beaucoup plus adapté pour réaliser des liaisons (ou *bindings* en anglais) vers d'autres langages. Le modélisateur pourrait alors choisir la *liaison* correspondant à son langage de prédilection.

Une autre piste serait de proposer un formalisme informatique pour décrire le méta-modèle. C'est ce que nous avons exploré avec l'utilisation de la bibliothèque Flecs [1] : il est possible de décrire les entités et les composants. Comme notre méta-modèle repose majoritairement sur ces notions, alors nous pouvons écrire un fichier descriptif. Par exemple, le fichier 29 est un exemple du formalisme de la bibliothèque pour décrire l'état du monde, en réemployant l'exemple du chapitre 4.

Code 29: Exemple d'un fichier plecs.

```
Bathtub : opack.Artefact
Patient : opack.Artefact
Doctor : opack.Agent
{
    - Introvert
    - Stress {1.0}
    simple__Sense
    {
        - Patient
    }
}
```

Idéalement, nous aimerions ajouter deux choses :

— Premièrement, la possibilité de déporter la description du flux, des stratégies et des fonctions-impacts dans ce type de fichier. Cela est en partie possible, mais ce type de fichier est uniquement descriptif. Il n'y a pas de capacités de programmation ce qui est important pour la mise en place du flux. Toutefois, des solutions uniquement descriptives sont envisageables. Par exemple, pour la stratégie de sélection d'actions, nous pourrions décrire les fonctions-impacts sous forme de *pattern* appliqué sur les actions.

— Deuxièmement, utiliser un format interopérable qui n'est pas dépendant de Flecs, utilisable par n'importe quelle bibliothèque ECS. C'est le but du projet Ecsact [2] qui est en cours. Cependant, la notation de relation est encore très nouvelle et risque de ne pas être prise en compte, car très peu de bibliothèques ECS possèdent cette notion (Flecs est pour le moment l'unique à notre connaissance).

7.3.2 Perspectives à moyen terme

À moyen terme, il serait intéressant d'appliquer nos travaux dans différents contextes et cas de simulation multi-agents.

1. https://github.com/SanderMertens/flecs
2. https://ecsact.dev/

Adaptation de modèles d'agents avec le méta-modèle

Jusqu'ici dans nos travaux, nous avons uniquement adapter, à notre méta-modèle, le modèle d'agent Replicants [Huguet et al., 2018] et ses modèles cognitifs. Cela nous a permis ensuite d'intégrer partiellement le modèle cognitif de Driskell [Driskell et al., 2018].

Nous pourrions envisager d'aller plus loin, en adaptant d'autres modèles d'agents, comme BEN [Bourgais et al., 2021], tout en gardant nos propriétés : sensibilité, intelligibilité et représentativité. Nous pourrions ainsi intégrer d'autres modèles cognitifs.

La première étape consisterait à vérifier si le méta-modèle supporte l'intégration des principes BDI [Bratman, 1987]. Si des fonctionnalités sont manquantes, comme des stratégies d'opérations, alors il convient de les définir. La difficulté est de respecter au maximum nos propriétés. Cette étape est à répéter pour l'ensemble des modèles cognitifs, comme OCEAN [Ortony et al., 2022]. Puis la seconde, serait d'adapter le fonctionnement de BEN, avec notre méta-modèle.

Une évaluation serait tout d'abord de vérifier dans le cas d'étude de la boîte de nuit qui a pris feu, si nous pouvons générer les mêmes comportements. Puis, si en intégrant un nouveau modèle cognitif, nous retrouvons les comportements induits.

Intégration d'un moteur scénaristique

Actuellement, nos travaux ne considèrent aucunement la présence d'un moteur scénaristique pour guider la simulation selon des objectifs haut-niveaux, comme des objectifs pédagogiques pour la formation [Lacaze-Labadie, 2019]. Conceptuellement, les deux approches sont à l'opposé : dans une approche scénaristique, le comportement des agents est dicté par le scénario, tandis que dans une approche « agents », les agents sont autonomes.

Nous pourrions envisager une approche hybride où les deux approches cohabitent. Le moteur scénaristique guide la simulation selon ses objectifs, tout en laissant de l'autonomie aux agents. La difficulté consiste donc à relier ces deux approches opposées.

Nous pensons que notre approche est une compatible avec de telles approches. Tout d'abord, de la même manière que les modèles cognitifs, nous pouvons intégrer les impacts d'un moteur scénaristique dans le modèle d'agents. Par exemple, au sein de la sélection d'action, le moteur scénaristique peut privilégier des actions plutôt que d'autres.

Aussi, le moteur scénaristique pourrait venir influencer le comportement des agents, avec notamment la notion de « late commitment » [Lacaze-Labadie, 2019] : certaines caractéristiques des agents ne sont pas déterminées tant que nous n'en avons pas besoin, ce qui permet au moteur scénaristique de les fixer selon ce qui l'arrange.

7.3.3 Perspectives à long terme

Selon ce que nous avons compris des travaux de Sébastien Picault et Vianney Sicard [Picault and Sicard, 2020], une simulation multi-paradigmes consiste à ne pas toujours simuler des agents et à la place, employer des paradigmes issus d'autres disciplines scientifiques (physique, biologie, statistique, *etc.*). « Les agents doivent être mobilisés lorsqu'ils apportent une plus-value clairement identifiable » [Picault and Sicard, 2020]. Cette direction s'applique notamment aux simulations multi-niveaux [Sicard et al., 2021], de co-simulation [Siebert et al., 2009] et de couplages de modèles [Huynh et al., 2017].

Ensuite, ce même article argumente les intérêts de cette approche : robustesse, validation, ouverture disciplinaire, *etc.* . Il soulève aussi des difficultés et les questions de recherches à se poser pour la communauté. Nous interprétons notamment la nécessité d'une approche modulaire qui facilite la réutilisation de paradigmes et une gestion efficace des ressources computationnelles.

Ces discussions font échos avec ce que nous avons proposé dans OPACK : 1 modèle d'agent avec N modèles cognitifs. Le multi-paradigmes suggère quant à lui qu'il faudrait des méta-modèles capables de supporter N modèles d'agents différents. Toutefois, nous pensons que nos travaux sont un pas dans cette nouvelle direction : la généricité, modularité et composabilité de notre approche permettraient d'établir plusieurs paradigmes, y compris hors modèle d'agent, pour nos agents. Nous pourrions même envisager de changer dynamiquement de paradigme lors de la simulation, grâce à la dynamicité des flux d'opérations et la composabilité. Qui plus est, notre implémentation et l'application des principes DOD suggèrent des gains potentiels de ressources.

Dualité « agent/information »

D'ailleurs, nous suggérons, qu'avec ce principe, nos agents peuvent alterner entre la notion d'« agent » (*i.e.* fonctionnant selon un modèle d'agent) et d'« information ».
Lors de l'évaluation de performances du chapitre 6 et du scénario d'itérations avec des agents minimalistes, nous avons illustré que nos agents n'impactent pas la vitesse d'itération de la simulation lorsqu'ils ne font rien, à l'instar des autres approches NetLogo [Wilensky, 1999] et Gama [Taillandier et al., 2016].
D'une certaine manière, c'est le paradigme employé qui induit s'il s'agit d'un agent ou d'une information. Mais comme la composabilité et la dynamicité permettent de changer le paradigme, alors selon les besoins à un instant donné de la simulation, nous pourrions adapter le paradigme pour passer de la notion d'agent à information et vice/versa.

Il conviendrait tout d'abord réaliser un état de l'art, notamment vis-à-vis des approches de simulations multi-niveaux, de co-simulations et de couplages de modèles pour vérifier les apports, limites et manques de nos travaux.

Ensuite, une première étape serait d'abord d'appliquer nos travaux dans le contexte d'une simulation multi-niveaux. Il serait intéressant d'étudier l'adéquation de la dynamicité de paradigmes pour agréger ou désagréger des agents selon le niveau considéré.

Ensuite, si la littérature des SMA propose un important nombre de modèles d'agent, c'est que les besoins sont nombreux : nous ne cherchons pas tous les mêmes comportements tout le temps. Dès lors, il semble intéressant de pouvoir combiner les modèles, selon les besoins de la simulation. Une seconde étape serait donc de considérer des co-simulations et des couplages de modèles où plusieurs modèles d'agent (ou paradigmes) cohabitent simultanément. Nous pourrions ainsi combiner le modèle de BEN pour tous les comportements haut-niveau et un modèle réactif pour les comportements réactifs. L'objectif serait donc de travailler sur la réutilisation de modèles comme le suggère [Picault and Sicard, 2020], ainsi que sur leurs combinaisons pour s'attaquer aux « problèmes difficiles » des SMA.

Table des figures

Bibliographie

[noa, a] APA Dictionary of Psychology.

[noa, b] The Sims - AIGPG Wiki.

[see, 2012] (2012). Cognition. In Seel, N. M., editor, *Encyclopedia of the Sciences of Learning*, pages 559–559. Springer US, Boston, MA.

[Abaci et al., 2005] Abaci, T., Cíger, J., and Thalmann, D. (2005). Planning with Smart Objects. In *WSCG*.

[Adam and Gaudou, 2016] Adam, C. and Gaudou, B. (2016). BDI agents in social simulations : a survey. *The Knowledge Engineering Review*, 31(3) :207–238.

[Algava et al., 2011] Algava, E., Chouanière, D., Christine, C., Dubré, J.-Y., Kittel, F., Leclerc, A., Le Moal, M., Loriol, M., Moisan, M.-P., Niedhammer, I., Pezet-Langevin, V., Sermet, C., Sultan-Taieb, H., and Weibel, L. (2011). Stress au travail et santé : situation chez les indépendants. Research report, Institut national de la santé et de la recherche médicale (INSERM).

[Anderson, 1996] Anderson, J. R. (1996). ACT : A simple theory of complex cognition. *American psychologist*, 51(4) :355. Publisher : American Psychological Association.

[Baber, 2018] Baber, C. (2018). Designing Smart Objects to Support Affording Situations : Exploiting Affordance Through an Understanding of Forms of Engagement. *Frontiers in Psychology*, 9.

[Balaji and Srinivasan, 2010] Balaji, P. G. and Srinivasan, D. (2010). An Introduction to Multi-Agent Systems. In Srinivasan, D. and Jain, L. C., editors, *Innovations in Multi-Agent Systems and Applications - 1*, Studies in Computational Intelligence, pages 1–27. Springer, Berlin, Heidelberg.

[Barot, 2014] Barot, C. (2014). *Scénarisation d'environnements virtuels : vers un équilibre entre contrôle, cohérence et adaptabilité.* These de doctorat, Compiègne.

[Barot et al., 2013] Barot, C., Lourdeaux, D., Burkhardt, J.-M., Amokrane, K., and Lenne, D. (2013). V3S : A virtual environment for risk-management training based on human-activity models. *Presence : Teleoperators and Virtual Environments*, 22(1) :1–19.

[Bernon et al., 2005] Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., and Zambonelli, F. (2005). A Study of Some Multi-agent Meta-models. In Odell, J., Giorgini, P., and Müller, J. P., editors, *Agent-Oriented Software Engineering V*, Lecture Notes in Computer Science, pages 62–77, Berlin, Heidelberg. Springer.

[Bloch et al., 1992] Bloch, H., Chemama, R., Gallo, A., Leconte, P., Le Ny, J.-F., Postel, J., Moscovici, S., Reuchlin, M., and Vurpillot, E. (1992). Grand dictionnaire de la psychologie. Publisher : Larousse.

[Bourgais, 2018] Bourgais, M. (2018). *Vers des agents cognitifs, affectifs et sociaux dans la simulation.* phdthesis, Normandie Université.

[Bourgais et al., 2021] Bourgais, M., Taillandier, P., and Vercouter, L. (2021). Using BEN for the simulation of cognitive, affective and social agents. In Amblard, F., Chapuis, K., Drogoul, A., Gaudou, B., Longin, D., and Verstaevel, N., editors, *1st conference GAMA Days 2021*, Toulouse (Online), France. Frédéric Amblard and Kevin Chapuis and Alexis Drogoul and Benoit Gaudou and Dominique Longin and Nicolas Verstaevel.

[Bratman, 1987] Bratman, M. (1987). *Intention, Plans, and Practical Reason.* Center for the Study of Language and Information.

[Carvalho, 2002] Carvalho, C. (2002). The gap between processor and memory speeds.

[Chakraborti et al., 2019] Chakraborti, T., Kulkarni, A., Sreedharan, S., Smith, D. E., and Kambhampati, S. (2019). Explicability ? Legibility ? Predictability ? Transparency ? Privacy ? Security ? The Emerging Landscape of Interpretable Agent Behavior. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29 :86–96.

[Charousset et al., 2016] Charousset, D., Hiesgen, R., and Schmidt, T. C. (2016). Revisiting Actor Programming in C++. *Computer Languages, Systems & Structures*, 45 :105–131.

[Chiva et al., 2003] Chiva, E., Devade, J., Donnard, J., and Maruéjouls, S. (2003). Motivation graphs : A new architecture for complex behavior simulation. *AI Game Programming Wisdom*, 2.

[Chong et al., 2007] Chong, H.-Q., Tan, A.-H., and Ng, G.-W. (2007). Integrated cognitive architectures : A survey. *Artificial Intelligence Review*, 28 :103–130.

[Claypool and Claypool, 2009] Claypool, M. and Claypool, K. (2009). Perspectives, frame rates and resolutions : It's all in the game. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, page 42–49, New York, NY, USA. Association for Computing Machinery.

[Colledanchise and Ögren, 2018] Colledanchise, M. and Ögren, P. (2018). Behavior Trees in Robotics and AI : An Introduction. *arXiv :1709.00084 [cs].* arXiv : 1709.00084.

[Costa Jr. and McCrae, 2008] Costa Jr., P. T. and McCrae, R. R. (2008). The Revised NEO Personality Inventory (NEO-PI-R). In *The SAGE handbook of personality theory and assessment, Vol 2 : Personality measurement and testing*, pages 179–198. Sage Publications, Inc, Thousand Oaks, CA, US.

[de Blauwe et al., 2022a] de Blauwe, T., Lourdeaux, D., and Sabouret, N. (2022a). Design of modular generation of human behaviour in a collaborative context. In *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 269–274, Hangzhou, China. IEEE.

[de Blauwe et al., 2022b] de Blauwe, T., Sabouret, N., and Lourdeaux, D. (2022b). Métamodèle d'agent pour la génération de comportements variables induits par des modèles cognitifs modulaires. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2022)*, Saint-Etienne, France.

[Demary, 2020] Demary, G. (2020). Évaluation cognitive du leader dans une dyade hiérarchique : des comportements non verbaux du suiveur aux comportements de leadership. *Bulletin de psychologie*, Numéro 569(5) :271–275.

[Driskell et al., 2015] Driskell, T., Driskell, J. E., and Salas, E. (2015). Mitigating Stress Effects on Team Cohesion. In *Team Cohesion : Advances in Psychological Theory, Methods and Practice*, volume 17 of *Research on Managing Groups and Teams*, pages 247–270. Emerald Group Publishing Limited.

[Driskell et al., 2018] Driskell, T., Salas, E., and Driskell, J. E. (2018). Teams in extreme environments : Alterations in team development and teamwork. *Human Resource Management Review*, 28(4) :434–449.

[Dubitzky et al., 2012] Dubitzky, W., Kurowski, K., and Schott, B. (2012). *Repast HPC : A Platform for Large-Scale Agent-Based Modeling*, pages 81–109.

[Erol et al., 1994] Erol, K., Hendler, J., and Nau, D. (1994). HTN Planning : Complexity and Expressivity. *Proceedings of the National Conference on Artificial Intelligence*, 2.

[Fabian, 2018] Fabian, R. (2018). Data-oriented design. *framework*, 21 :1–7.

[Fadier et al., 2003] Fadier, E., De La Garza, C., and Didelot, A. (2003). Safe design and human activity : construction of a theoretical framework from an analysis of a printing sector. *Safety Science*, 41(9) :759–789.

[Faul et al., 2009] Faul, F., Erdfelder, E., Buchner, A., and Lang, A.-G. (2009). Statistical power analyses using G*Power 3.1 : Tests for correlation and regression analyses. *Behavior Research Methods*, 41(4) :1149–1160.

[Faul et al., 2007] Faul, F., Erdfelder, E., Lang, A.-G., and Buchner, A. (2007). G*Power 3 : A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior Research Methods*, 39(2) :175–191.

[Faur, 2016] Faur, C. (2016). *Approche computationnelle du regulatory focus pour des agents interactifs : un pas vers une personnalité artificielle.* These de doctorat, Université Paris-Saclay (ComUE).

[Fitrianie et al., 2020] Fitrianie, S., Bruijnes, M., Richards, D., Bönsch, A., and Brinkman, W.-P. (2020). The 19 Unifying Questionnaire Constructs of Artificial Social Agents : An IVA Community Analysis. In *Proceedings of the 20th ACM International Conference on Intelligent Virtual Agents*, IVA '20, pages 1–8, New York, NY, USA. Association for Computing Machinery.

[Gaver, 1991] Gaver, W. W. (1991). Technology affordances. In *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91*, pages 79–84, New Orleans, Louisiana, United States. ACM Press.

[Gibson, 1977] Gibson, J. J. (1977). The theory of affordances. *Hilldale, USA*, 1(2).

[Hanlon and Watts, 2017] Hanlon, S. and Watts, C. (2017). Dragon Age Inquisition's Utility Scoring Architecture. *2017*, page 9.

[Härkönen, 2019] Härkönen, T. (2019). Advantages and implementation of entity-component-systems.

[Herzig et al., 2016] Herzig, A., Lorini, E., Perrussel, L., and Xiao, Z. (2016). BDI Logics for BDI Architectures : Old Problems, New Perspectives. *KI - Künstliche Intelligenz*.

[Huguet, 2018] Huguet, L. (2018). *Génération de comportements pour les PVAs*. thesis, Compiègne.

[Huguet et al., 2016] Huguet, L., Lourdeaux, D., and Sabouret, N. (2016). Présentation du projet VICTEAMS. In *Workshop Affect Compagnon Artificiel Interaction (WACAI 2016)*, Brest, France.

[Huguet et al., 2018] Huguet, L., Lourdeaux, D., and Sabouret, N. (2018). Moteur de sélection de tâches pour des personnages virtuels autonomes non omniscients. In *Workshop affects, compagnons artificiels et interaction (WACAI 2018)*, Porquerolles, France.

[Huynh et al., 2017] Huynh, N. Q., en Huu, T. N., ud Grignard, A., Huynh, H. X., and oul, A. D. (2017). Coupling equation based models and agent-based models : example of a multi-strains and switch sir toy model. *EAI Endorsed Transactions on Context-aware Systems and Applications*, 4(11).

[Jiang et al., 2007] Jiang, H., Vidal, J. M., and Huhns, M. N. (2007). EBDI : an architecture for emotional agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*, page 1, Honolulu, Hawaii. ACM Press.

[Lacaze-Labadie, 2019] Lacaze-Labadie, R. (2019). *Planification et modèle graphique pour la génération dynamique de scénarios en environnements virtuels*. These de doctorat, Compiègne.

[Laird, 2022] Laird, J. E. (2022). Introduction to soar.

[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.

[Lamarche and Donikian, 2002] Lamarche, F. and Donikian, S. (2002). Automatic orchestration of behaviours through the management of resources and priority levels. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 3*, AAMAS '02, pages 1309–1316, Bologna, Italy. Association for Computing Machinery.

[Lambert and Lazarus, 1970] Lambert, W. W. and Lazarus, R. S. (1970). Psychological Stress and the Coping Process. *The American Journal of Psychology*, 83(4) :634.

[Larsen, 2019] Larsen, J. B. (2019). Going beyond BDI for agent-based simulation. *Journal of Information and Telecommunication*, 3(4) :446–464. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/24751839.2019.1620024.

[Lazarus and Folkman, 1984] Lazarus, R. S. and Folkman, S. (1984). *Stress, appraisal, and coping*. Springer publishing company.

[Llopis, 2009] Llopis, N. (2009). Data-oriented design (or why you might be shooting yourself in the foot with oop). *Game Developer Magazine*, 16(8).

[Luke, 2011] Luke, S. (2011). Multiagent simulation and the mason library. *George Mason University*, 1.

[Macal et al., 2018] Macal, C. M., Collier, N. T., Ozik, J., Tatara, E. R., and Murphy, J. T. (2018). CHISIM : AN AGENT-BASED SIMULATION MODEL OF SOCIAL INTERACTIONS IN a LARGE URBAN AREA. In *2018 Winter Simulation Conference (WSC)*. IEEE.

[Martínez-Cruz et al., 2011] Martínez-Cruz, C., Blanco, I., and Vila, M. (2011). Ontologies versus relational databases : Are they so different ? A comparison. *Artificial Intelligence Review - AIR*, 38.

[Mayer, 2005] Mayer, J. D. (2005). A Tale of Two Visions : Can a New View of Personality Help Integrate Psychology ? *American Psychologist*, 60(4) :294–307.

[Mayer, 2015] Mayer, J. D. (2015). The personality systems framework : Current theory and development. *Journal of Research in Personality*, 56 :4–14.

[Millington, 2019] Millington, I. (2019). *AI for Games*. CRC Press, Boca Raton, 3 edition.

[Morrison, 1994] Morrison, J. P. (1994). Flow-based programming. In *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*, pages 25–29.

[North et al., 2013] North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1).

[Ochs et al., 2009] Ochs, M., Sabouret, N., and Corruble, V. (2009). Simulation de la dynamique des émotions et des relations sociales de personnages virtuels. *Revue des Sciences et Technologies de l'Information - Série RIA : Revue d'Intelligence Artificielle*, 23(2-3) :327.

[Okreša Đurić et al., 2019] Okreša Đurić, B., Rincon, J., Carrascosa, C., Schatten, M., and Julian, V. (2019). MAMbO5 : a new ontology approach for modelling and managing intelligent virtual environments based on multi-agent systems. *Journal of Ambient Intelligence and Humanized Computing*, 10(9) :3629–3641.

[Omicini et al., 2008] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3) :432–456.

[Ortony et al., 1988] Ortony, A., Clore, G. L., and Collins, A. (1988). *The Cognitive Structure of Emotions*. Cambridge University Press, Cambridge.

[Ortony et al., 2022] Ortony, A., Clore, G. L., and Collins, A. (2022). *The cognitive structure of emotions*. Cambridge university press.

[PhD and PhD, 1984] PhD, R. S. L. and PhD, S. F. (1984). *Stress, Appraisal, and Coping*. Springer Publishing Company.

[Picault and Sicard, 2020] Picault, S. and Sicard, V. (2020). Les meilleurs agents sont ceux qu'on ne simule pas : vers des architectures de simulation multi-paradigmes ? In *28e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2020)*. Cépaduès.

[Plaisant et al., 2010] Plaisant, O., Guertault, J., Courtois, R., Réveillère, C., Mendelsohn, G., and John, O. (2010). Histoire des « Big Five » : OCEAN des cinq grands facteurs de la personnalité. Introduction du Big Five Inventory français ou BFI-Fr. *Annales Médico-psychologiques, revue psychiatrique*, 168(7) :481–486.

[Russell and Norvig, 2021] Russell, S. J. and Norvig, P. (2021). *Artificial intelligence : a modern approach*. Pearson series in artificial intelligence. Pearson, Hoboken, fourth edition edition.

[Sansonnet and Bouchet, 2014] Sansonnet, J.-P. and Bouchet, F. (2014). A Framework Covering the Influence of ffm/neo pi-r Traits over the Dialogical Process of Rational Agents. In Filipe, J. and Fred, A., editors, *Agents and Artificial Intelligence*, volume 449, pages 62–79. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Sicard et al., 2021] Sicard, V., Andraud, M., and Picault, S. (2021). L'organisation comme Design Pattern dans les systèmes multi-agents multi-niveaux. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*, Journées Francophones sur les Systèmes Multi-Agents, pages 73–82, Bordeaux, France.

[Siebert et al., 2009] Siebert, J., Ciarletta, L., and Chevrier, V. (2009). De l'intérêt du couplage de modèles pour appréhender les interactions utilisateurs-réseaux dynamiques. *Revue des Sciences et Technologies de l'Information - Série RIA : Revue d'Intelligence Artificielle*, pages 749–776.

[Silverman et al., 2006a] Silverman, B. G., Bharathy, G., O'Brien, K., and Cornwell, J. (2006a). Human Behavior Models for Agents in Simulators and Games : Part II : Gamebot Engineering with PMFserv. *Presence : Teleoperators and Virtual Environments*, 15(2) :163–185.

[Silverman et al., 2006b] Silverman, B. G., Johns, M., Cornwell, J., and O'Brien, K. (2006b). Human Behavior Models for Agents in Simulators and Games : Part I : Enabling Science with PMFserv. *Presence : Teleoperators and Virtual Environments*, 15(2) :139–162.

[Silverman et al., 2012] Silverman, B. G., Pietrocola, D., Nye, B., Weyer, N., Osin, O., Johnson, D., and Weaver, R. (2012). Rich socio-cognitive agents for immersive training environments : case of NonKin Village. *Autonomous Agents and Multi-Agent Systems*, 24(2) :312–343.

[Slotin, wip] Slotin, S. (2022-wip)). Algorithms for modern hardware. *Open-access online book*.

[Sun, 2006] Sun, R. (2006). *Cognition and Multi-Agent Interaction : From Cognitive Modeling to Social Simulation*. Cambridge University Press. Google-Books-ID : V1RyhTamPkgC.

[Sánchez et al., 2019] Sánchez, Y., Coma, T., Aguelo, A., and Cerezo, E. (2019). ABC-EBDI : An affective framework for BDI agents. *Cognitive Systems Research*, 58 :195–216.

[Taillandier et al., 2016] Taillandier, P., Bourgais, M., Caillou, P., Adam, C., and Gaudou, B. (2016). A situated BDI agent architecture for the GAMA modelling and simulation platform. In *17th International Workshop on Multi-Agent-Based Simulation (MABS 2016) at AAMAS 2016*, number 10399, pages pp. 3–23, Singapore, Singapore.

[Taillandier et al., 2019] Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.-N., Marilleau, N., Caillou, P., Philippon, D., and Drogoul, A. (2019). Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica*, 23(2) :299–322.

[Weiss, 2013] Weiss, G. (2013). *Multiagent Systems*. MIT Press. Google-Books-ID : WY36AQAAQBAJ.

[Weyns et al., 2006] Weyns, D., Omicini, A., and Odell, J. (2006). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*.

[Wilensky, 1999] Wilensky, U. (1999). NetLogo. *Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL*.

[Éditions Larousse,] Éditions Larousse. Dictionnaire de français Larousse — larousse.fr. https://www.larousse.fr.

Chapitre 8

Annexes

8.1 Questionnaires

Cette section liste les deux questionnaires employés lors de nos différentes évaluations.

8.1.1 Premier questionnaire (VICTEAMS)

Explications naïves

	Communicant	Non-Communicant
Proactif	« L'infirmier a tendance à discuter les instructions avec ses supérieurs »	« L'infirmier préfère prendre des initiatives personnelles »
Passif	« L'infirmier a besoin de suivre des ordres précis »	« L'infirmier évite de faire face à ses supérieurs »

Stressé	Non-Stressé
« L'infirmier a tendance à être maladroit »	« L'infirmier est rigoureux dans son travail »

Questionnaire

Texte (scénario) *(exemple : A1 B1 C1, Communicant, Passif, Stressé, Compétent)*

Un médecin possède une équipe d'infirmiers compétents sous ses ordres. Il doit collaborer avec eux afin de soigner un patient.

Après analyse, le médecin diagnostique que le patient a besoin d'une injection d'adrénaline. Il ordonne à un des infirmiers d'exécuter cette action. Cependant, l'infirmier réalise que le patient n'a pas besoin d'adrénaline mais de morphine.

L'infirmier répète l'ordre du médecin, puis le suit sans discuter ; l'infirmier se prépare donc à injecter de l'adrénaline.

Il injecte cependant de la morphine, et réalise son erreur.

Il annonce alors quelle injection il a faite au médecin.

Questions

Le comportement de l'infirmier vous semble-t-il rationnel ?

Pas du tout | Pas vraiment | Sans avis | Un peu | Tout à fait

Justifiez votre choix (optionnel) :

_____

_____

Diriez-vous que l'infirmier est :

Non-Communicant	Peu Communicant	Sans avis	Plutôt Communicant	Communicant
Passif	Plutôt Passif	Sans avis	Plutôt Proactif	Proactif
Non-Stressé	Peu Stressé	Sans avis	Plutôt Stressé	Stressé

Justifiez vos choix (optionnel) :

_____

_____

Texte (explication) (à partir des modèles)

Il a été déterminé que l'infirmier est Communicant, Passif et Stressé, et donc que :

- *Il demande quoi faire et fait ce qu'on lui dit*
- *Il est dans une situation où il peut se tromper, notamment pour des tâches relativement simples*

Questions

Ces explications vous semblent-elles pertinentes ?

Pas du tout	Pas vraiment	Sans avis	Un peu	Tout à fait

Justifiez votre choix (optionnel) :

_____

_____

→ ***(Si le participant a répondu que le comportement ne semblait pas rationnel)*** Avec ces explications, le comportement de l'infirmier vous semble-t-il plus rationnel ?

Pas du tout	Pas vraiment	Sans avis	Un peu	Tout à fait

8.1.2 Deuxième questionnaire (ORCHESTRAA)

Questionnaire-orchestraa-pvas

Ce questionnaire fait partie d'une expérimentation menée dans le cadre du projet **ORCHESTRAA** par le laboratoire Heudiasyc. Dans cette expérimentation, nous souhaitons évaluer la qualité du comportement de quelques opérateurs d'une salle de conduite : le **SODO**, l'**AOCC** et le **CAS DO**. Vous allez évaluer le comportement de ces postes lors de la gestion d'un **TIC** (Troop In Contact). Voici le scénario :

> *" Un TIC est reporté par une troupe au sol (indicatif: PLATOON 39) avec un FAC (indicatif: KARABAL 10). Cette troupe subit des tirs d'artillerie. L'AOCC relaye l'information et requiert un support aérien rapidement. Le CAS DO doit prendre connaissance de l'information et fournir une solution. Le SODO en tant que supérieur du CAS DO peut intervenir pour l'aider ou le rappeler à l'ordre. Le CAS DO peine à trouver une solution mais finit par en donner une. "*

L'expérience va se dérouler de la manière suivante : Dans le mail que nous vous avons envoyé, vous avez dû recevoir :

- Un numéro que nous vous demandrons de bien préciser dans le champ prévu (au tout début du questionnaire).
- Une vidéo de la RAP décrivant le résultat des actions menées par la salle. Cette vidéo permet aussi de vous donner une idée de la situation. Vous pourrez notamment y observer les avions proches du TIC, ainsi que des informations sur leur état.
- Un extrait des messages échangés dans l'outil JCHAT par les trois postes considérés, jusqu'à l'envoi de la solution.

Après avoir visionné et lu les messages, merci de remplir le questionnaire !

Notez votre numéro *

Merci de bien noter le numéro qui vous a été fourni avec le mail !

Maîtrisez-vous les postes suivants ? *

	Pas du tout	Peu	Sans avis	Plutôt	Tout à fait
CAS DO *					

	Pas du tout	Peu	Sans avis	Plutôt	Tout à fait
SODO *					
AOCC *					

A quel point connaissez-vous ce poste, ses responsabilités, les interactions à avoir, etc.

Évaluation de la salle

Comment jugez-vous la performance de la salle ? *

- Mauvaise
- Plutôt mauvaise
- Sans avis
- Plutôt bonne
- Bonne

Selon vous, au vu des échanges de messages, du temps de réaction, de la pertinence de la solution et de la rapidité à la fournir, diriez-vous que la salle est plutôt performante ou non ?

[OPTIONEL] - Justifiez votre choix

N'hésitez pas si vous avez des remarques sur le timing, la cohérence/pertinence de certains messages ou l'absence de messages.

Évaluation des postes

Diriez-vous que le comportement de la personne au poste considéré est représentatif de ce que vous observez pour ce type de poste? *

	Non-représentatif	Peu représentatif	Sans avis	Plutôt représentatif	Représentatif
CAS DO *					
SODO *					
AOCC *					

Diriez-vous que la personne au poste considéré est stressé ? *

	Non-stressé	Peu stressé	Sans avis	Plutôt stressé	Stressé
CAS DO *					
SODO *					
AOCC *					

Diriez-vous que la personne au poste considéré est communicante ? *

	Non-communicante	Peu communicante	Sans avis	Plutôt communicante	Communicante
CAS DO *					
SODO *					
AOCC *					

Diriez-vous que la personne au poste considéré est pro-active ? *

	Passif	Plutôt passif	Sans avis	Plutôt proactive	Proactive
CAS DO *					
SODO *					
AOCC *					

[OPTIONEL] - Justifiez vos choix

N'hésitez pas à remarquer si des champs étaient difficiles à juger.

Ne communiquez aucun mot de passe via Framaforms.

8.2 Code-source

Cette section liste les codes-sources des différents outils lors de l'évaluation de performances.

8.2.1 GAMA

```
/**
* Name: benchmark
* Based on the internal test template.
* Author: Tristan
* Tags:
*/

model benchmark_creation


global
{
  string agents_type <- "empty";
  int nb_agents <-1 ;

  float start_time <- 0.0;
  float elapsed_time <- 0.0;

  init
  {
    write "Launching experiment with " + nb_agents + " " + agents_type + ".";
    if(agents_type = "empty")
    {
        start_time <- machine_time;
      create empty_agent number:nb_agents;
    }else if(agents_type = "simple")
    {
      start_time <- machine_time;
      create simple_agent number:nb_agents;
    }else if(agents_type = "simple_if")
    {
       start_time <- machine_time;
      create simple_agent_if number:nb_agents;
    }
    }

    reflex on_end when: cycle = 0
  {
      elapsed_time <- machine_time - start_time;
      write "Ending experiment with " + nb_agents + " " + agents_type + " in " +
    elapsed_time;
      if(agents_type = "empty")
    {
      ask empty_agent {do die;}
    }else if(agents_type = "simple")
    {
      ask simple_agent {do die;}
    }else if(agents_type = "simple_if")
    {
      ask simple_agent_if {do die;}
    }
    }
}

species empty_agent
{
  reflex {}
```

```
57 }
58
59 species simple_agent
60 {
61   int counter <- 0;
62   float value_1 <- 0.0;
63   bool tag;
64   string value3;
65   string value4;
66   string value5;
67
68   init {
69     counter <- 0;
70     value_1 <- 0.0;
71     tag <- flip(0.5);
72     value3 <- "";
73     value4 <- "";
74     value5 <- "";
75   }
76
77   action increment_counter
78   {
79         counter <- counter + 1;
80   }
81 }
82
83 species simple_agent_no_if parent: simple_agent
84 {
85
86   reflex act
87   {
88     do increment_counter;
89   }
90 }
91
92 species simple_agent_if parent: simple_agent
93 {
94
95   reflex act when: tag
96   {
97     do increment_counter;
98   }
99 }
100
101 experiment Benchmark_Creation type: batch repeat: 2 until: (cycle=1)
102 {
103   parameter 'Agents_type:' var: agents_type among: ["empty", "simple", "simple_if"];
104   parameter 'Number of agents:' var: nb_agents among: [1048576];
105
106   init
107   {
108     float mean_ms <- 0.0;
109     float mean_ns <- 0.0;
110     save [nb_agents, agents_type, mean_ms, mean_ns] to: "benchmark_creation.txt" type: "
      csv" rewrite:true;
111   }
112
113   reflex on_end
114   {
115     float mean_ms <- simulations mean_of each.elapsed_time;
116     float mean_ns <- mean_ms * 1000000;
117     write "ms";
118     write replace(string(mean_ms), ".", ",");
119     write "ns";
120     write replace(string(mean_ns), ".", ",");
121     save [nb_agents, agents_type, mean_ms, mean_ns] to: "benchmark_creation.txt" type: "
      csv" rewrite:false;
122     ask simulations { do die; }
123   }
```

```
124 }
```

Listing 8.1 – Benchmark GAMA pour la création d'agents

```
1  /**
2  * Name: benchmark
3  * Based on the internal test template.
4  * Author: Tristan
5  * Tags:
6  */
7
8  model benchmark_iteration
9
10
11 global
12 {
13   string agents_type <- "empty";
14   int max_cycles <- 256;
15   int nb_agents <-1 ;
16
17   float start_time <- 0.0;
18   float elapsed_time <- 0.0;
19   float mean_cycle <- 0.0;
20
21   init
22   {
23     write "Launching experiment with " + nb_agents + " " + agents_type + ".";
24     if(agents_type = "empty")
25     {
26       create empty_agent number:nb_agents;
27     }else if(agents_type = "simple")
28     {
29       create simple_agent number:nb_agents;
30     }else if(agents_type = "simple_if")
31     {
32       create simple_agent_if number:nb_agents;
33     }
34
35       start_time <- machine_time;
36     }
37
38     reflex on_end when: cycle = max_cycles
39   {
40       elapsed_time <- machine_time - start_time;
41       mean_cycle <- elapsed_time / cycle;
42       write "Ending experiment with " + nb_agents + " " + agents_type + ". " + cycle + "
       iterations in " + elapsed_time + " ms so with a mean of " + mean_cycle + " ms per
      cycle";
43       if(agents_type = "empty")
44     {
45       ask empty_agent {do die;}
46     }else if(agents_type = "simple")
47     {
48       ask simple_agent {do die;}
49     }else if(agents_type = "simple_if")
50     {
51       ask simple_agent_if {do die;}
52     }
53     }
54 }
55
56 species empty_agent
57 {
58   reflex {}
59 }
60
61 species simple_agent
62 {
63   int counter <- 0;
```

```
64   float value_1 <- 0.0;
65   bool tag;
66   string value3;
67   string value4;
68   string value5;
69
70   init {
71     counter <- 0;
72     value_1 <- 0.0;
73     tag <- flip(0.5);
74     value3 <- "";
75     value4 <- "";
76     value5 <- "";
77   }
78
79   action increment_counter
80   {
81         counter <- counter + 1;
82   }
83 }
84
85 species simple_agent_no_if parent: simple_agent
86 {
87
88   reflex act
89   {
90     do increment_counter;
91   }
92 }
93
94 species simple_agent_if parent: simple_agent
95 {
96
97   reflex act when: tag
98   {
99     do increment_counter;
100  }
101 }
102
103 experiment Benchmark_Iteration type: batch repeat: 2 until: (cycle=257)
104 {
105   parameter 'Agents_type:' var: agents_type among: ["empty", "simple", "simple_if"];
106   parameter 'Number of agents:' var: nb_agents among: [1048576];
107
108   init
109   {
110     float mean_total_time <- 0.0;
111     float mean_ms <- 0.0;
112     float mean_ns <- 0.0;
113     save [nb_agents, agents_type, mean_total_time, mean_ms, mean_ns] to: "
        benchmark_iteration.txt" type: "csv" rewrite:true;
114   }
115
116   reflex on_end
117   {
118     float mean_total_time <- simulations mean_of each.elapsed_time;
119     float mean_ms <- mean_of(simulations, mean_cycle);
120     float mean_ns <- mean_ms * 1000000;
121     write "ms";
122     write replace(string(mean_ms), ".", ",");
123     write "ns";
124     write replace(string(mean_ns), ".", ",");
125     save [nb_agents, agents_type, mean_total_time, mean_ms, mean_ns] to: "
        benchmark_iteration.txt" type: "csv" rewrite:false;
126     ask simulations { do die; }
127   }
```

```
128 }
```

Listing 8.2 – Benchmark GAMA pour l'itération d'agents

8.2.2 CAF

```
1  #include <string>
2  #include <iostream>
3  #include <chrono>
4
5  #include "caf/actor_ostream.hpp"
6  #include "caf/actor_system.hpp"
7  #include "caf/caf_main.hpp"
8  #include "caf/event_based_actor.hpp"
9
10 using namespace caf;
11
12 static std::chrono::duration<double, std::nano> total_time;
13 static std::vector<int> params {1, 8, 64, 512, 4096, 32768, 262144, 1048576};
14 static int nb_agents;
15
16 void caf_actor_creation(actor_system& sys) {
17     auto start = std::chrono::steady_clock::now();
18     for(int i{0}; i < nb_agents; i++)
19     {
20         auto actor = sys.spawn<event_based_actor>();
21     }
22     auto end = std::chrono::steady_clock::now();
23     total_time += end-start;
24 }
25
26 void caf_actor_iteration(actor_system& sys) {
27     for(int i{0}; i < nb_agents; i++)
28     {
29         auto actor = sys.spawn<event_based_actor>();
30     }
31     auto start = std::chrono::steady_clock::now();
32     sys.
33     auto end = std::chrono::steady_clock::now();
34     total_time += end-start;
35 }
36
37 // creates a main function for us that calls our caf_main
38 int main(int argc, char** argv) {
39     int repeat {10};
40     auto start = std::chrono::steady_clock::now();
41     for(int n : params)
42     {
43         nb_agents = n;
44     for(int i{0}; i < repeat; i++)
45     {
46       caf::exec_main_init_meta_objects<>();
47       caf::core::init_global_meta_objects();
48       caf::exec_main<>(caf_actor_creation, argc, argv);
49     }
50         std::cout << "Total time elapsed for " << nb_agents << " agents : " <<
       total_time.count() / 8 << " ns [Repeat = " << repeat << "]\n";
51     }
52     return 1;
53 }
```

Listing 8.3 – Code-source CAF

8.2.3 NetLogo

```
1  extensions [csv]
2  breed [empty-agents empty-agent]
3  breed [simple-agents simple-agent]
4  simple-agents-own [counter value1 tag value3 value4 value5]
5
6  to reset-all
7    clear-all
8    reset-ticks
9    reset-timer
10 end
11
12 to create_empty_agent [n]
13   create-empty-agents n
14 end
15
16 to create_simple_agent [n]
17   create-simple-agents n [
18     set counter 0
19     set value1 0
20     set tag false
21     set value3 ""
22     set value4 ""
23     set value5 ""
24   ]
25 end
26
27 to benchmark_empty_agent_creation
28   let repeat_bench 10
29   let results []
30   foreach [1 8 64 512 4096 32768 262144 1048576]
31   [ nb_agents ->
32     let time_sum 0
33     repeat repeat_bench
34     [
35       reset-all
36       create_empty_agent nb_agents
37       set time_sum time_sum + timer
38     ]
39     let result time_sum / repeat_bench
40     set results lput result results
41     print (word "Nb agents " nb_agents  " in " result)
42   ]
43   show "In seconds :"
44   show csv:to-row results
45
46   show "In nano-seconds :"
47   let s-to-ns 1E-9
48   let ns-results map [ i -> i / s-to-ns ] results
49   show csv:to-row ns-results
50 end
51
52 to benchmark_empty_agent_iteration
53   let repeat_bench 1
54   let results []
55   foreach [1 8 64 512 4096 32768 262144 1048576]
56   [ nb_agents ->
57     let time_sum 0
58     repeat repeat_bench
59     [
60       reset-all
61       create_empty_agent nb_agents
62       reset-timer
63       repeat 256
64       [
65         ask empty-agents [ ]
66         tick
67       ]
68       set time_sum (time_sum + timer) / 256
69     ]
```

```
70      let result time_sum / repeat_bench
71      set results lput result results
72      print (word "Nb agents " nb_agents  " in " result)
73    ]
74    show "In seconds :"
75    show csv:to-row results
76
77    show "In nano-seconds :"
78    let s-to-ns 1E-9
79    let ns-results map [ i -> i / s-to-ns ] results
80    show csv:to-row ns-results
81  end
82
83  to benchmark_simple_agent_creation
84    let repeat_bench 10
85    let results []
86    foreach [1 8 64 512 4096 32768 262144 1048576]
87    [ nb_agents ->
88      let time_sum 0
89      repeat repeat_bench
90      [
91        reset-all
92        create_simple_agent nb_agents
93        set time_sum time_sum + timer
94      ]
95      let result time_sum / repeat_bench
96      set results lput result results
97      print (word "Nb agents " nb_agents  " in " result)
98    ]
99    show "In seconds :"
100   show csv:to-row results
101
102   show "In nano-seconds :"
103   let s-to-ns 1E-9
104   let ns-results map [ i -> i / s-to-ns ] results
105   show csv:to-row ns-results
106 end
107
108 to benchmark_simple_agent_iteration
109   let repeat_bench 1
110   let results []
111   foreach [1 8 64 512 4096 32768 262144 1048576]
112   [ nb_agents ->
113     let time_sum 0
114     repeat repeat_bench
115     [
116       reset-all
117       create_simple_agent nb_agents
118       reset-timer
119       repeat 256
120       [
121         ask simple-agents [set counter counter + 1]
122         tick
123       ]
124       set time_sum (time_sum + timer) / 256
125     ]
126     let result time_sum / repeat_bench
127     set results lput result results
128     print (word "Nb agents " nb_agents  " in " result)
129   ]
130   show "In seconds :"
131   show csv:to-row results
132
133   show "In nano-seconds :"
134   let s-to-ns 1E-9
135   let ns-results map [ i -> i / s-to-ns ] results
136   show csv:to-row ns-results
137 end
138
```

```
to benchmark_simple_agent_iteration_if
  let repeat_bench 1
  let results []
  foreach [1 8 64 512 4096 32768 262144 1048576]
  [ nb_agents ->
    let time_sum 0
    repeat repeat_bench
    [
      reset-all
      create_simple_agent nb_agents
      ask n-of (nb_agents / 2) simple-agents [ set tag true ]
      reset-timer
      repeat 256
      [
        ask simple-agents [ if tag [set counter counter + 1]]
        tick
      ]
      set time_sum (time_sum + timer) / 256
    ]
    let result time_sum / repeat_bench
    set results lput result results
    print (word "Nb agents " nb_agents  " in " result)
  ]
  show "In seconds :"
  show csv:to-row results

  show "In nano-seconds :"
  let s-to-ns 1E-9
  let ns-results map [ i -> i / s-to-ns ] results
  show csv:to-row ns-results
end
```

Listing 8.4 – Code-source Netlogo

Python

8.2.4 Jade

```
package fundamentals.jadeBehaviours.composite.sequential;
import jade.core.Agent;
public class Empty extends Agent {
    private static final long serialVersionUID = 7494115848353017052L;
    protected void setup() {}
}
```

Listing 8.5 – Code-source de la classe d'agent « vide »

```
package benchmarks;

import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;
import org.openjdk.jmh.annotations.*;
import princ.Principal;

import java.util.concurrent.TimeUnit;

@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Warmup(iterations = 3, time = 3, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
@Fork(1)
@State(Scope.Benchmark)
public class benchmark {

    Principal principal;
    //@Param({"1", "8", "64", "512", "4096", "32768", "262144", "1048576"})
```

```
21      @Param({"1", "8", "64", "512", "4096", "32768"})
22      int nb_agents;
23
24      public static void main(String[] args) throws Exception {
25          org.openjdk.jmh.Main.main(args);
26      }
27
28      @Setup
29      public void setup()
30      {
31          principal = new Principal();
32          principal.setup();
33      }
34
35      @Benchmark
36      public void create_agents()
37      {
38          for(int i = 0; i<nb_agents; i++)
39          {
40              create_agent();
41          }
42      }
43
44      private void create_agent()
45      {
46          Object[] params = new Object[]{};
47          ContainerController container = principal.containerList.get("Mycontainer1");
48          try {
49              AgentController ag=  container.createNewAgent(java.util.UUID.randomUUID().
        toString(), fundamentals.jadeBehaviours.composite.sequential.Empty.class.getName(),
        params);
50          } catch (StaleProxyException e) {
51              e.printStackTrace();
52          }
53      }
54  }
```

Listing 8.6 – Code-source de la création d'agent VIDE avec JADE et JMH.