Formalisation et implémentation des interactions pour la simulation centrée individu

Yoann Kubera — Philippe Mathieu — Sébastien Picault

LIFL CNRS UMR 8022
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
Cité Scientifique - 59655 Villeneuve d'Ascq Cedex
{yoann.kubera,philippe.mathieu,sebastien.picault}@lifl.fr

RÉSUMÉ. *De nombreuses architectures logicielles ont été proposées dans le cadre de la simulation centrée individu, notamment à travers des plates-formes parfois très abouties. Néanmoins, il n'y a pas dans ces plates-formes de découpage logiciel entre les agents, leurs comportements, et le processus de sélection d'action. Nous proposons au contraire une architecture où les interactions sont réifiées indépendamment des agents qui les utilisent. Un agent peut alors subir ou effectuer un ensemble quelconque d'interactions qui n'ont pas été spécifiquement développées pour lui. Outre la réutilisabilité des interactions, cette architecture logicielle a l'avantage, à l'instar des systèmes experts, de séparer le déclaratif du procédural, donc de faciliter à la fois la conception et la montée à l'échelle dans la richesse des comportements.*

ABSTRACT. *This article deals with the software architecture for individual-centered simulations, whithin which many entities interact one with another. Many software architecture have been developped in this context, especially many advanced – but domain specific – frameworks. Yet those frameworks lead to tight software dependencies between agents, behaviors and action selection mechanisms. We propose an architecture where interactions are reified regardless of agents using them. An agent may perform or undergo a set of interactions which are not specifically developped for it. Thus most interactions can be re-used in many contexts. In addition, this architecture clearly separates data from processing, and thus makes the design of simulations easier. Moreover, this new approach enables programmers to build simulations with a large number of different behaviors at the same time.*

MOTS-CLÉS : *agents, simulation, interaction, plate-forme, modèle, méthodologie, architecture*

KEYWORDS: *agents, simulation, interaction, framework, model, methodology, architecture*

RSTI - L'objet – 14/2008. Architectures logicielles, pages 9 à 33

1. Introduction

Depuis quelques années, la simulation à base d'agents a pris une place prépondérante dans les outils de simulation du vivant, que ce soit pour en comprendre les mécanismes ou pour en reproduire les caractéristiques à des fins ludiques (jeux vidéo, animation cinématographique, etc.), ce que ne permet pas la simulation analytique (*i.e.* par équations). Ce domaine établit un lien étroit entre experts d'un domaine particulier (biologie, sociologie, etc.) et experts informaticiens, et son aspect pluridisciplinaire a donné naissance à tout un ensemble de plates-formes informatiques plus ou moins spécifiques au domaine simulé, dont aucun métamodèle n'est à ce jour spécifié.

Un grand nombre de ces dernières, ouvertes, laissent au concepteur une totale liberté d'implémentation des agents, de leur comportement et de l'environnement. Elles sont caractérisées par un plus ou moins grand degré de sophistication logicielle : tous les raffinements du génie logiciel sont envisageables (réutilisation, généricité, *design patterns*, composants). C'est le cas notamment de Swarm (Burkhart, 1994), Madkit (Gutknecht *et al.*, 2001) ou Magique (Bensaid *et al.*, 1997, Mathieu *et al.*, 2001), qui ne sont d'ailleurs pas spécifiquement dédiées à la réalisation de simulations, mais peuvent également être employées pour la résolution distribuée de problèmes. D'autres outils, comme Netlogo (Wilenski, 1999), sont au contraire conçus pour pouvoir être utilisés par des non informaticiens, et reposent donc sur une forme très simple de programmation. L'ouverture de ces plates-formes et leur généricité sont malheureusement obtenues au détriment d'un cadre précis qui guiderait la conception des comportements : le déclaratif et le procédural sont mélangés dans les agents, ce qui implique la réécriture complète du fonctionnement de l'agent en cas d'ajout ou de retrait de compétences.

A l'inverse, divers formalismes permettent de guider fortement la conception de comportements, au détriment de la réutilisabilité de ces derniers dans d'autres formalismes. C'est le cas, par exemple, des réseaux de Pétri, réseaux neuronaux, bases de règles, subsomption, etc. L'une des rares architectures permettant une généricité des comportements quel que soit le domaine d'application est le modèle *Belief-Desire-Intention* ou ses variantes (Wooldridge, 2000, Pokahr *et al.*, 2007), dans la mesure où il sépare les connaissances de leur traitement ; mais il est souvent inadéquat pour la réalisation de simulations. D'ailleurs, peu d'applications pratiques dérivées de ce modèle existent.

Nous proposons dans cet article une méthodologie de conception intuitive de comportements génériques et réutilisables, nommée IODA[1] (Mathieu *et al.*, 2001, Mathieu *et al.*, 2007, Kubera *et al.*, 2008), permettant d'abstraire de l'implémentation de l'agent tout ce qui peut être décrit de manière générique dans son comportement, et de réifier ces notions indépendamment de l'agent ou de son comportement global. Nous décrivons de plus la plate-forme JEDI[2] qui présente une réalisation possible de la méthodologie IODA en java pour des simulations à agents réactifs et situés.

---

1. Pour : *Interaction Oriented Design of Agent simulations*.
2. Pour : *Java Environnement for the Design of agent Interactions*.

La section 2 décrit certaines limites des méthodologies et plates-formes de simulation existantes, la section 3 présente la méthodologie IODA et ses divers avantages, tels que la séparation du déclaratif et du procédural, la définition d'ontologies d'interactions réutilisables ou le passage au large échelle. Enfin la section 4 présente la plateforme JEDI qui constitue une implémentation des concepts présentés dans IODA, et illustre leur généricité au travers d'un paramétrage aisé et fin du moteur de simulation.

2. Décompositions existantes de systèmes multi-agents

Selon un point de vue systémique (von Bertalanffy, 1976), un système multi-agent est un système composé d'un ensemble d'entités (les agents) pouvant communiquer entre elles, similaires à des boîtes noires. Un agent est alors caractérisé par :

– un comportement à ses bornes, *i.e.* les échanges de messages impliquant l'agent et son environnement (en son sens systèmique « tout ce qui est extérieur à l'agent ») ;

– une structure interne, *i.e.* son état interne et son fonctionnement intrinsèque.

Nous proposons de considérer la recherche concernant la structure des agents selon ces deux points de vue.

2.1. *La communication entre agents*

De nombreux formalismes permettent de décrire les communications entre agents, à l'instar des nombreux langages de coordination recensés dans (Papadopoulos *et al.*, 1998). Dans de tels langages, une interaction correspond à un ensemble structuré de messages envoyés entre des agents, permettant d'atteindre un but particulier. Par exemple, dans le cas où un agent source demande à un agent cible une ressource, l'interaction suit le schéma proposé dans la figure 1.

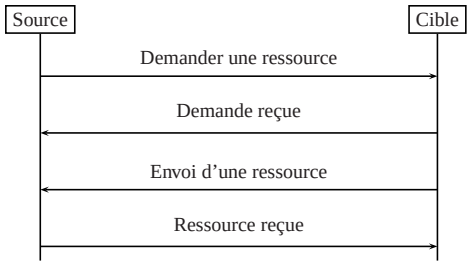

Figure 1. *Protocole simple suivi par une interaction où un agent source demande une ressource à un agent cible. Les flèches correspondent aux messages envoyés*

Dans le pire des cas ce protocole est dispersé dans le code des agents y participant, posant ainsi des problèmes de maintenabilité. Des approches comme (Doi *et al.*, 2005) proposent de réifier les interactions en des entités autonomes pour éviter leur dispersion, mais semblent renverser une partie du problème : il est nécessaire de spécifier

dans l'interaction réifiée une partie du traitement faite chez les agents (par exemple, dans la figure 1, ce serait la façon dont la cible choisit la ressource à envoyer). Ce genre de traitement est dédié à l'agent sous la forme de l'appel à une méthode propre à l'agent, car il dépend fortement de la structure interne de l'agent, et ne peut donc être exprimé de manière générique pour l'interaction. Une telle approche sépare clairement ce qui est spécifique aux agents et ce qui est spécifique aux interactions, permettant une grande maintenabilité des interactions en limitant leur dispersion.

Ces solutions sont suffisantes dans le cas d'applications réparties où le système se décrit en termes de services, mais pas dans le cas de simulations multi-agents, où le comportement des agents ne se limite pas à des actes de langage. En effet, ils peuvent aussi agir sans communiquer avec d'autres agents (par exemple *mourir*, *se déplacer*), et leurs actions ne se déclenchent en général qu'à certaines conditions, notion absente dans les interactions des langages de coordination. Ainsi, les langages de coordination ne sont pas suffisants pour exprimer la totalité du comportement des agents de manière générique et unifiée dans une simulation.

2.2. *La structure interne des agents*

L'aspect pluridisciplinaire de la simulation multi-agent a fourni un grand nombre de méthodologies et de plates-formes, associées à différentes façons de représenter les connaissances d'un système multi-agent de manière générique. Notre objectif n'est ni de les décrire en détail, ni de les classifier exhaustivement, mais de donner un aperçu de certaines limites des approches existantes telles que Swarm (Burkhart, 1994), Madkit (Gutknecht *et al.*, 2001), Magique (Bensaid *et al.*, 1997), VOLCANO (Ricordel *et al.*, 2002) en partie déjà dépeintes dans des articles tels que (Briot *et al.*, 2006).

La plupart des approches dédiées à la simulation ne font pas la distinction entre le modèle de la simulation et son implémentation sur la plate-forme. Cette absence de distinction rend la simulation totalement dépendante de la plate-forme et du langage de programmation utilisés, et ne permet pas de formuler le modèle utilisé (défini implicitement par l'implémentation) sur une autre plate-forme de simulation. Pourtant, la réutilisation du comportement des agents est primordiale dans la simulation informatique, où un certain nombre de variantes d'une même simulation doivent être développées avant d'atteindre les résultats escomptés.

Certaines approches proposent de développer la simulation, et plus particulièrement la structure interne des agents, dans un contexte logiciel générique, permettant leur réutilisabilité des simulations. C'est la cas, par exemple, du modèle de composants hierarchiques MALEVA (Briot *et al.*, 2006), fortement inspiré de l'architecture de subsomption proposée par (Brooks, 1986).

Cette approche centre le développement d'une simulation sur les actions pouvant être adoptées par les agents et sur les relations qu'elles entretiennent : les actions y sont représentées par des composants indépendants des agents, dont le nombre de bornes d'entrée et de sortie dépend des états lus et modifiés par cette action.

Définition 2.1 *Une **action** est une séquence de **primitives d'action** modifiant certains éléments de l'état global de la simulation, qui ne peut être entreprise que si ses conditions d'activation sont vérifiées. On appelle **état global de la simulation** (noté $\mathbb{E}$) l'union de l'ensemble des états de l'environnement, de l'ensemble des agents présents dans la simulation (noté $\mathbb{A}$) et de l'état de ces derniers.*

Définition 2.2 *On appelle **conditions d'activation** d'une action l'ensemble des conditions portant sur l'état global de la simulation devant être vérifiées pour qu'un agent puisse exécuter cette action.*

Définition 2.3 *La **sélection d'actions** est un mécanisme permettant à un agent de prendre une décision sur l'action à adopter en fonction de l'état global de la simulation au moment de la décision.*

La sélection d'action de l'agent repose sur l'interconnexion ces composants : dans la figure 2, les sélections d'action (a) et (b) imposent un ordre séquentiel d'exécution des actions, alors que la sélection d'action (c) n'impose rien, du moment que les trois actions sont exécutées. Cette représentation ne définit pas explicitement les conditions d'activation liées à une action : *a priori*, l'action *NewBaby* ne peut pas avoir lieu si l'un des deux agents participant à l'action est déjà en période de gestation ou est stérile. Les structures de contrôle externes aux composants proposées remédient en partie à ce problème (branchements, itérations, etc.), mais séparent de ce fait deux entités (l'action et ses conditions d'activation) étant souvent étroitement liées.

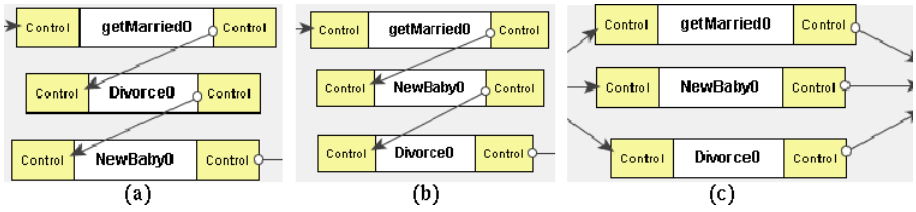

Figure 2. *Trois exemples de sélection d'action au sein d'un agent dans MALEVA portant sur trois actions* getMarried, newBaby *et* Divorce

De plus, la logique de connexion des composants devient rapidement complexe (et ce malgré les possibilité de hiérarchisation et d'encapsulation), et donc difficilement utilisable dans des agents faisant intervenir un grand nombre de comportements.

Enfin, même si l'approche présentée ci-avant permet une certaine dynamicité dans l'architecture des agents, elle n'est pas adaptée à des simulations où les agents utilisent des variantes d'actions (actions fortement similaires mais dont une partie dépend des agents y participant). En effet, une action exprime en général un ensemble structuré de primitives d'actions ayant un sens (*Divorce* propose une suite de primitives d'action effectuant la séparation de deux agents mariés, par exemple *répartition des biens communs* puis *déménagement des deux partis*) mais dont les modalités d'exécution

peuvent changer selon les agents (par exemple un agent rancunier tente de s'accaparer d'un maximum de biens lors de la *répartition des biens communs*, alors qu'un agent naïf recherche un partage équitable). Dissocier complètement les comportements des agents revient alors à dissocier deux comportements fortement similaires en deux entités disjointes, et nécessite de créer, pour chaque variante possible d'un comportement, un nouveau composant.

Ainsi, une telle approche facilite la réutilisation des actions, mais les actions réalisées conjointement avec d'autres agents n'y sont pas explicites, et cette méthode n'est pas adaptée aux simulations mettant en jeu un grand nombre de variantes d'action.

2.3. *Vers une généricité encore plus grande dans les simulations*

Nous décrivons dans cet article la plate-forme de simulation JEDI, issue de la méthodologie IODA (déjà développée d'une manière générale dans (Kubera *et al.*, 2008)), permettant de réifier et d'unifier la notion d'interaction pour les simulations multi-agents, et dont la vocation principale est la conception de simulations large échelle en termes de représentation des connaissances. Nous insistons notamment sur les choix réalisés pour JEDI et l'influence de ces choix pour la réalisation de simulations efficaces et conformes aux attentes de l'expérimentateur.

Définition 2.4 *On appelle par la suite* ***simulation large échelle*** *une simulation dans laquelle l'environnement contient un grand nombre d'agents (on parle alors de* ***large échelle en termes de calculs****) ou une simulation pour laquelle le nombre de familles d'agents et/ou le nombre de comportements par famille d'agents est important (on parle alors de* ***large échelle en termes de représentation des connaissances****).*

Il est à noter que l'un n'implique pas l'autre :

– une simulation large échelle en termes de calculs ne l'est pas forcément en termes de représentation des connaissances. C'est le cas avec 100 000 particules qui effectuent toutes la même interaction ;

– une simulation large échelle en termes de représentation des connaissances ne l'est pas forcément en termes de calculs. C'est le cas de jeux comme Age of Empires, où 15 familles d'agents d'environ 20 instances chacune interagissent de manière variée et qui requiert bien plus de calcul pour le rendu graphique que pour l'IA.

Cette méthodologie générale, ainsi que la plate-forme qui en est issue, réifient les comportements sous la forme d'entités logicielles appelées interactions (unifiant en un même terme tout types d'actions auxquelles peut participer un agent, cf. section 3), et séparent le déclaratif du procédural tout en fournissant les moyens analytiques et algorithmiques de conserver cette séparation, lorsque l'on passe du problème concret à l'implémentation. Nous montrons qu'elle est de plus adaptée au large échelle en termes de représentation des connaissances par son caractère intuitif, et qu'elle peut être exploitée comme support pour créer de manière générique des simulations.

3. La méthodologie IODA

La plate-forme de simulation JEDI implémente de manière fidèle les notions proposées dans la méthodologie de conception de simulation IODA (Mathieu *et al.*, 2007), en particulier les interactions entre agents, en les réifiant sous la forme d'entités disjointes de ces derniers. Avant de décrire une telle architecture, nous présentons la méthodologie sous-jacente, exprimant les divers éléments du modèle utilisé lors de la simulation.

3.1. *Approche centrée interactions*

Dans cette approche, un agent dispose d'un agrégat de primitives de base, définissant le niveau de granularité initial dans le comportement d'un agent. Ces primitives sont divisées en deux catégories :

– des primitives de perception, qui permettent à l'agent d'accéder à des informations relatives à l'état global de la simulation : l'état de l'environnement (par exemple la présence d'un agent dans l'environnement), la perception d'un état de l'agent, la perception de communications provenant d'autres agents (communications explicites par messages, ou implicites *via* une modification de l'environnement) ;

– des primitives d'action, qui permettent à l'agent de modifier l'état de la simulation, et donc son propre état (augmenter son énergie, ajouter un objet à son inventaire, etc.), l'état de l'environnement (déplacement d'un agent, création d'un agent, dépôt d'une phéromone, etc.), l'état d'autres agents (communication, échange d'objets, etc.).

Nous ne détaillons pas ici les critères de choix de ces primitives, dans la mesure où elles sont intrinsèquement dépendantes du problème traité.

Nous définissons les interactions comme des ensembles de primitives de base qui impliquent simultanément plusieurs agents et qui constituent un bloc sémantique dans une simulation donnée. Par exemple *manger* ou *ouvrir* ne sont pas de simples actions atomiques, mais correspondent à des ensembles structurés d'actions mettant en jeu deux agents différents et qui ne peuvent être effectuées qu'à certaines conditions, peu dépendantes des spécificités des agents concernés.

Définition 3.1 *Une **interaction** est une séquence d'actions primitives s'appliquant à plusieurs agents et soumise à des conditions d'activation. Ces primitives peuvent être réalisées selon des modalités variables par les agents, mais leur enchaînement logique est décrit de façon générale par leur structuration sous la forme d'une interaction.*

L'expression de la séquence d'actions primitives n'est contrainte par aucun formalisme, et il est *a priori* possible d'utiliser un workflow, un langage de coordination ou même une structure issue de MALEVA pour les exprimer, sous réserve que ce formalisme soit adéquat. En effet, un langage de coordination est par exemple inutile dans le cas d'une interaction dégénérée (voir la remarque de la section 3.3).

Définition 3.2 *Chaque agent participant à cette interaction ne joue pas forcément le même rôle. On effectue la distinction entre les agents **source** déterminant les agents qui peuvent effectuer l'interaction, et les agents **cible** qui peuvent la subir. Pour avoir lieu, une interaction doit mettre en relation des agents sources et des agents cibles.*

Propriété 3.1 *Les primitives d'action et de perception devant être implémentées par un agent pour qu'il puisse participer à une interaction dépendent du rôle (i.e. source ou cible) qu'il joue dans cette interaction.*

Ainsi formulée, cette notion d'interaction apparaît fortement dissociée des agents qui les utiliseront. Une interaction peut alors être réutilisée dans différentes simulations et appliquée à des arrangements de familles d'agents différentes.

Il faut souligner que les interactions donnent lieu à l'écriture d'une classe indépendante des agents lors de l'implémentation. On constitue donc, au fil des simulations, des bibliothèques d'interactions et des bibliothèques d'agents qui peuvent être réutilisées dans des variantes d'une même simulation, ou dans des simulations différentes.

3.2. *Caractéristiques des agents*

Dans IODA, les agents se réduisent à des spécifications simples, et sont représentés de manière homogène, ce qui permet d'intégrer n'importe quel agent à notre modèle de simulation centré interactions : un agent est une entité autonome instanciée à partir d'une famille d'agents particulière (*i.e.* une famille d'agents correspond au métamodèle d'un ensemble d'agents).

Notation 3.1 *Soit $\mathbb{F}$ l'ensemble des familles d'agents d'une simulation donnée, $\mathbb{A}$ l'ensemble des agents, et $\forall \mathcal{F} \in \mathbb{F}, \forall x \in \mathbb{A}, x \prec \mathcal{F} \Leftrightarrow x$ est un agent instance de la famille d'agents $\mathcal{F}$.*

Définition 3.3 *Une **famille d'agents** $\mathcal{F}$ (aussi appelée **classe d'équivalence d'agents** ou **classe d'agents**) est l'abstraction d'un ensemble d'agents partageant tout ou partie de leurs primitives de perception ou d'action ou de leur comportement. Les caractéristiques minimales d'une famille d'agents sont les suivantes :*

*– elle dispose d'une primitive d'initialisation exécutée lors de la création d'un agent $x \prec \mathcal{F}$ (notée **init()**) ;*

*– elle dispose d'une primitive d'évolution automatique de son état (notée **update()**), exécutée avant d'effectuer la sélection d'actions ;*

*– elle dispose de **primitives** de perception et d'action ;*

*– la perception de l'état global de la simulation se restreint à un **halo** dont le calcul est propre à la famille d'agents ;*

– elle se voit assigner l'ensemble des interactions auxquelles tout agent $x \prec \mathcal{F}$ peut participer en tant que source ou cible.

Cette définition est indépendante du langage où les familles sont implémentées : elles peuvent être implémentées dans un langage objet par des classes concrètes ou abstraites, par des structures de données dans des langages de programmation prcédurale, par des composants, etc.

REMARQUE. — On peut bien évidemment définir une famille singleton : $\mathcal{F} \in \mathbb{F} / \exists! x \in \mathbb{A} / x \prec \mathcal{F}$.

Perception et action ne s'étendent pas à l'ensemble de l'état global de la simulation. Pour qu'ils puissent interagir entre eux, les agents doivent passer par une étape de perception au cours de laquelle ils construisent leur halo de perception de l'état global de la simulation ainsi que leur voisinage, *i.e.* le sous-ensemble des agents avec lesquels ils peuvent agir.

Définition 3.4 *On appelle **halo de perception** (ou **représentation interne de l'état de la simulation**) d'un agent $x \prec \mathcal{F}$ le sous-ensemble de l'état global de la simulation qu'il manipule au travers de ses primitives. Il est noté $\mathcal{H}_{\mathcal{F}}(x) \subseteq \mathbb{E}$.*

*Le **voisinage** $\mathcal{N}(x)$ d'un agent x est l'ensemble des agents présents dans son halo de perception : $\mathcal{N}(x) = \mathbb{A} \backslash \{x\} \cap \mathcal{H}_{\mathcal{F}}(x)$.*

Dans notre architecture, la notion de halo est réifiée et peut être définie librement par le concepteur. Bien que libre, l'expression du halo va dépendre de la topologie de l'environnement utilisée : une surface pour un environnement plan, un volume pour un environnement en 3 dimensions, les accointances situées en deçà d'une distance maximale pour des réseaux d'accointances, etc. Néanmoins, nous n'émettons aucune contrainte sur ces topologies, du moment qu'il est possible d'y exprimer une distance. Ainsi, un voisinage dans IODA est similaire au voisinage dans (Giavitto, 2003) : il peut aussi bien être considéré dans un espace que dans une structure de données, ce qui permet de considérer des voisinages au sens structurel (*i.e.* degré de similitude des agents).

La construction du halo mise à part, la sélection des actions est indépendante de la topologie de l'environnement : seule la notion de distance au sein du voisinage y est importante (voir section 3.7).

3.3. *Cardinalité des interactions*

D'après la définition que nous avons donnée de l'interaction, un cas trivial vient immédiatement à l'esprit : celui qui fait appel à un agent source et un agent cible. Toutefois, la complexité des problèmes à simuler nécessite de prendre en compte d'autres situations, comme les actions d'un agent sur lui-même (par exemple *dormir*) ou sur l'environnement (par exemple *se déplacer*, *mourir*), mais aussi des interactions qui font intervenir plusieurs agents cibles (par exemple *exploser* dont la déflagration

touche plusieurs agents), voire plusieurs agents sources (par exemple *porter* un objet lourd à plusieurs). Cela nous amène à unifier ces situations *via* la notion de cardinalité.

Définition 3.5 *On appelle **cardinalité** d'une interaction I le couple composé du nombre d'agents sources (noté $card_S(I)$) et du nombre d'agents cibles (noté $card_T(I)$) nécessaires pour la réalisation de l'interaction.*

REMARQUE. — Le cas particulier des interactions réflexives (d'un agent sur lui-même) et d'un agent sur l'environnement, où la cible est implicite, sont appelées **interactions dégénérées**. Une interaction dégénérée est de cardinalité *(1,0)*.

L'interaction simultanée entre plusieurs sources et plusieurs cibles (cardinalité *(n,p)*) est ramenée systématiquement sous la forme *(1,(n+p-1))*, ou à plusieurs interactions *(1,n)*. Cette solution, bien que critiquable puisqu'elle traite des agents source comme s'ils étaient des cibles, donne satisfaction dans tous les cas que nous avons traités. Cela reste néanmoins un problème dur (Mathieu *et al.*, 2007).

Définition 3.6 *Une interaction I est écrite **sous forme normale** lorsqu'elle fait intervenir exactement une source, c'est-à-dire si $card_S(I) = 1$*

Propriété 3.2 *Toute interaction peut s'écrire sous forme normale :*

– les interactions de cardinalité (1,0), (1,1) *ou* (1,n) *sont déjà sous forme normale ;*

– une interaction de cardinalité (n,1) *peut s'exprimer à la voix passive (i.e. en permutant les rôles source et cibles) pour se ramener à une cardinalité* (1,n) *;*

– dans une interaction de cardinalité (n,p)*, on peut choisir comme source unique l'un des n agents sources et reléguer les autres parmi les cibles, et ainsi se ramener à une cardinalité* (1,n+p-1)*.*

IODA est un cadre méthodologique permettant de représenter et d'exploiter des interactions de tout type de cardinalité. Dans la suite de cet article, nous considérons que toutes les interactions sont écrites sous forme normale.

3.4. *L'analyse d'un problème*

Il ne s'agit pas ici d'ajouter une énième méthodologie au corpus volumineux des méthodes d'analyse et de conception de modèles informatiques d'un système simulé qui se cantonnent au niveau des spécifications. IODA fournit certes un cadre de modélisation, mais aussi des algorithmes qui permettent d'aller entièrement et de façon univoque de l'analyse à l'implémentation. Nous avons d'ailleurs réalisé un générateur automatique de code permettant de construire un modèle d'une simulation selon la méthodologie IODA, et de générer le code correspondant pour la plate-forme JEDI. La réutilisation du modèle n'est pas limitée à JEDI, et peut faire l'objet d'une génération de code destinée à une autre plate-forme de simulation (par les mêmes procédés nous pouvons générer du code Netlogo).

La méthodologie IODA propose cinq étapes pour la conception d'une simulation centrée interactions (une application est proposée en section 4.5) :

1) identifier d'abord les *familles d'agents*, ainsi que les différentes *interactions* participant à la simulation. Cela conduit à construire une matrice entre sources et cibles potentielles dans laquelle on fait apparaître ces différentes entités. On parle alors d'assignation des interactions à des agents sources et des agents cibles ;

2) écrire les *conditions d'activation* et la *séquence d'actions* primitives de chaque interactions ;

3) identifier les *primitives d'action et de perception* que les agents doivent implémenter en fonction de leur rôle dans les interactions auxquelles ils participent ;

4) spécifier, pour toute interaction assignée à des agents sources et cibles, la *priorité* relative de cette interaction et sa *garde de distance*. Cela conduit à raffiner la matrice précédente ;

5) déterminer la dynamique des familles d'agents, c'est-à-dire la façon dont, au fil des interactions et de la simulation, évolue cette matrice, et par conséquent les possibilités d'interaction des agents ;

6) déterminer les spécificités requises par le modèle portant sur la gestion du temps, de l'espace, etc. Le modèle ne pourra être implémenté que sur des plates-formes conformes à ses spécificités, sans quoi une ambiguïté persiste et risque de biaiser les résulats de la simulation. Ce point ne sera pas détaillé ici, mais (Michel *et al.*, 2003, Kubera *et al.*, 2007) le traitent plus en détail.

La matrice d'interactions a l'avantage d'être une représentation intuitive du comportement des agents, et donc facilement accessible pour les experts du domaine. Elle est préférable à une spécification en UML où la connectivité devient rapidement illisible si beaucoup d'interactions sont possibles entre deux familles d'agents.

3.5. *La matrice d'assignation des interactions aux agents*

L'assignation des interactions à des agents sources et des agents cibles définit ce que peuvent faire les agents. Pour permettre leur utilisation dans une simulation, il reste à préciser deux contraintes :

– *la garde de distance*. En effet, les agents n'interagissent potentiellement qu'avec des agents voisins suffisamment « proches ». Cette distance est indépendante du caractère situé ou non de la simulation, puisqu'elle peut aussi représenter une distance sociale, une distance dans un réseau d'accointances, ou de toute autre mesure de proximité dans un espace d'états ;

– *la priorité* que prend une interaction lorsqu'elle est assignée à un agent, par rapport aux autres interactions. Elle permet de construire une hiérarchie entre interactions au sein de l'agent. Elle n'est pas forcément constante et peut être réévaluée au cours de la simulation.

Définition 3.7 *On appelle **assignation** (notée $a_{\mathcal{S}/\mathbb{T}}$) d'un ensemble d'interactions $(I_k)_{k\in[1,n]}$ entre une famille d'agents sources $\mathcal{S} \in \mathbb{F}$ et un q-uplet de familles d'agents cibles $\mathbb{T} = (\mathcal{T}_j)_{j\in[1,q]} \in \mathbb{F}^q$, un ensemble d'**éléments d'assignation** de la forme (I_k, p_k, d_k), $k \in [1, n]$, avec :*

– I_k : l'interaction pouvant être effectuée par tout $x \prec \mathcal{S}$ et subie par le t-uplet d'agents $\{t_j, j \in [1, q]/t_j \prec \mathcal{T}_j \wedge (\forall l \in [1, q], t_j = t_l \Rightarrow j = l)\}$;

– $card_T(I_k) = q$;

*– p_k : la **priorité** donnée à l'interaction I_k ;*

*– d_k : la **garde de distance** en deçà de laquelle l'interaction est **spatialement possible** (voir définition 3.10).*

REMARQUE. — Dans le cas des interactions dégénérées ($\mathbb{T} = \emptyset$), une assignation $a_{\mathcal{S}/\emptyset}$ est définie par des couples de la forme (I_k, p_k).

Définition 3.8 *On appelle **matrice d'interaction** la matrice $\mathcal{M} = (a_{\mathcal{S}/\mathbb{T}})_{\mathcal{S}\in\mathbb{F},\mathbb{T}\subseteq\mathbb{F}^{\mathbb{N}}}$ de toutes les assignations entre source $\mathcal{S}$ et cibles $\mathbb{T}$ possibles. La source et et les cibles peuvent aussi bien être des familles d'agents individuelles que des catégories abstraites (groupes, équipes, etc.).*

Par conséquent, pour toute simulation qui comporte des interactions dégénérées, il existe dans la matrice d'interaction une colonne $(a_{i/\emptyset})$. Par soucis de lisibilité, on ne représente ici que des exemples de cardinalité *(1,0)* et *(1,1)*. La forme générale de cette matrice est donnée sur un exemple dans les tableaux 1, 2 et 4.

3.6. *Les ontologies d'agents*

Des agents de familles différentes peuvent avoir des comportements proches. Par exemple, les agents *Chèvre* et *Loup* sont tous deux une sorte d'*Animal*, ce genre étant lui même défini par la capacité de se déplacer ou de mourir. Une famille d'agents $\mathcal{X} \in \mathbb{F}$ peut ainsi représenter un sous-ensemble particulier d'une autre famille d'agents $\mathcal{Y} \in \mathbb{F}$. On dit alors que $\mathcal{X}$ est une spécialisation de $\mathcal{Y}$.

Quatre éléments relatifs à la matrice d'interaction définissent cette relation de spécialisation et le moyen de spécifier les éléments d'assignation d'une famille d'agents.

Définition 3.9 *Les éléments définissant la relation de spécialisation sont :*

*– la **spécialisation d'une famille d'agents** $\mathcal{X} \in \mathbb{F}$ par une famille d'agents $\mathcal{Y} \in \mathbb{F}$ est définie par : $\forall y \in \mathbb{A}, y \prec \mathcal{Y} \Rightarrow y \prec \mathcal{X}$. On la note $\mathcal{Y} : \mathcal{X}$. Cette relation est transitive, irréflexive, non symétrique et non antisymétique. Sauf mention contraire, on a : $(\mathcal{Y} : \mathcal{X}) \Rightarrow (\forall \mathbb{T} \in \mathbb{F}^{\mathbb{N}}, a_{\mathcal{X}/\mathbb{T}} \subseteq a_{\mathcal{Y}/\mathbb{T}})$;*

*– l'**ajout d'un élément d'assignation** e ayant pour source $\mathcal{X} \in \mathbb{F}$ et pour cibles $\mathbb{T} \in \mathbb{F}^{\mathbb{N}}$ se fait au travers de l'opérateur "+" : $+(a_{\mathcal{X}/\mathbb{T}}, e) \Rightarrow e \in a_{\mathcal{X}/\mathbb{T}}$;*

*– le **retrait d'un élément d'assignation** e ayant pour source $\mathcal{X} \in \mathbb{F}$ et pour cibles $\mathbb{T} \in \mathbb{F}^{\mathbb{N}}$ se fait au travers de l'opérateur "-" :* $-(a_{\mathcal{X}/\mathbb{T}}, e) \Rightarrow \neg\big(e \in a_{\mathcal{X}/\mathbb{T}}\big)$ *;*

*– la **modification d'un élément d'assignation** $e = (I, p, d)$ ayant pour source $\mathcal{X} \in \mathbb{F}$ et pour cibles $\mathbb{T} \in \mathbb{F}^{\mathbb{N}}$ se fait au travers de l'opérateur "*" :* $*(e, p', d') \Rightarrow (I, p', d') \in a_{\mathcal{X}/\mathbb{T}} \wedge (I, p, d) \notin a_{\mathcal{X}/\mathbb{T}}$.

REMARQUE. — Si $e = (I, p)$, $*(e, p') \Rightarrow (I, p) \notin a_{\mathcal{X}/\mathbb{T}} \wedge (I, p') \in a_{\mathcal{X}/\mathbb{T}}$

source \ cible	$\emptyset$	Herbe	Mouton	Chèvre	Loup
Herbe	+(Se reproduire ;0)				
Animal	+(Mourir ;3) +(Se déplacer ;0)				
Herbivore		+(Manger ;2 ;0)			
Mouton :Animal,Herbivore			+(Se reproduire ;1 ;1)		
Chèvre :Animal,Herbivore				+(Se reproduire ;1 ;1)	
Loup :Animal	*((Mourir ;3),4)		+(Manger ;2 ;0)	+(Manger ;3 ;0)	+(Se reproduire ;1 ;1)

Tableau 1. *Matrice d'interaction faisant apparaître les liens de spécialisation entre familles d'agents. Les éléments de cette matrice sont des opérateurs +,- ou * portant sur les éléments d'assignation*

Puisque la case de la matrice d'interaction qui définit $a_{S/T}$ se situe à l'intersection de la ligne de famille d'agents sources S et de la colonne de famille d'agents cibles T (ou $\emptyset$), les opérateurs présents dans la matrice ne précisent pas l'assignation qu'ils modifient (elle est définie implicitement par la case où est ajouté l'opérateur). Si aucun opérateur n'est spécifié, l'opérateur "+" est implicitement utilisé.

Dans le cas d'une simulation d'un écosystème à 4 espèces (voir section 3.4), dont la matrice d'interaction est présentée sur le tableau 4, la matrice équivalente utilisant les opérateur cités ci-avant (plus abstraite mais faisant apparaître les éléments propres à chaque famille d'agents) est présentée sur le tableau 1.

Un tel formalisme est indépendant de la plate-forme ou du langage de programmation utilisé. C'est lors de la génération du code associé que la relation de spécialisation prendra une forme différente selon la plate-forme de simulation ou le langage de programmation utilisé. Dans un langage objet, ce sera l'héritage, dans un langage de frames, ce sera '*sorte de*', etc.

3.7. *Notions utilisées par tout moteur de simulation utilisant IODA*

Pour déterminer si des agents peuvent interagir, nous définissons deux critères portant sur les éléments de la matrice d'interaction : le critère d'*éligibilité*, qui correspond à une vérification syntaxique de la possibilité d'interaction entre les agents, et le critère de *réalisabilité*, qui correspond à une vérification sémantique.

Notation 3.2 *Soit $x, y \in \mathbb{A}$. La fonction **dist(x,y)** permet de connaître la distance séparant les agents x et y dans la topologie utilisée par la simulation.*

Définition 3.10 *Un élément d'assignation* (I_k, p_k, d_k) *est dit* ***spatialement possible*** *pour un agent source* $s \in \mathbb{A}$ *et un q-uplet d'agents cibles* $(t_j)_{j\in[1,q]} \in \mathbb{A}^q$ *si et seulement si :*

– $card_T(I_k) = 0$, *ou*

– $card_T(I_k) \neq 0 \wedge (\forall j \in [1,q], t_j \in \mathcal{N}(x) \wedge dist(x,t_j) \leq d_k)$.

Définition 3.11 *Un élément d'assignation* $e = (I_k, p_k, c_k, d_k)$ *est dit* ***éligible*** *pour un agent source* $x \in \mathbb{A}$ *et un q-uplet d'agents cibles* $(t_j)_{j\in[1,q]} \in \mathbb{A}^q$ *(ce qu'on note* $eligible(e, x, (t_j)_{j\in[1,q]})$*) si et seulement si :*

– $\exists \mathcal{S} \in \mathbb{F}, \exists \mathbb{T} = (\mathcal{T}_j)_{j\in[1,q]} \in \mathbb{F}^q / x \prec \mathcal{S} \wedge \forall j \in [1,q], t_j \prec \mathcal{T}_j \wedge e \in a_{\mathcal{S}/\mathbb{T}}$. *Cela spécifie que l'agent* x *peut jouer le rôle de source, et que les agents de* $(t_j)_{j\in[1,q]}$ *peuvent jouer le rôle de cible ;*

– $\forall j, k \in [1,q], t_j = t_k \Rightarrow j = k$ *;*

– I_k *est spatialement possible pour* x *et* $(t_j)_{j\in[1,q]}$.

L'éligibilité porte sur des critères syntaxiques (possibilité d'être source ou cible d'après la matrice d'interaction) et non sémantiques : les conditions d'activation de l'interaction d'un élément assignation éligible peuvent ne pas être vérifiées pour ces agents x et $(t_j)_{j\in[1,q]}$.

Notation 3.3 *La fonction* $cond(I, x, (t_j)_{j\in[1,q]})$ *permet de vérifier si les conditions d'activation de l'interaction* I *appliquées aux agents* x *et* $(t_j)_{j\in[1,q]}$ *sont vérifiées.*

Définition 3.12 *Un élément d'assignation* $e = (I, p, d)$ *est* ***réalisable*** *par un agent source* $x \in \mathbb{A}$ *sur un t-uplet d'agents cibles* $(t_j)_{j\in[1,q]} \in \mathbb{A}^q$ *(ce que l'on note* $realisable(e, x, (t_j)_{j\in[1,q]})$*) si et seulement si* $eligible(e, x, (t_j)_{j\in[1,q]}) \wedge cond(I, x, (t_j)_{j\in[1,q]})$

Définition 3.13 *On appelle* ***potentiel d'interaction de niveau*** p ***de l'agent*** x*, noté* $P_p(x)$ *l'ensemble des couples (éléments d'assignation de priorité p, agents cibles) pour lesquels l'élément d'assignation est réalisable :* $P_p(x) = \{(e, (t_j)), (t_j) \in \mathbb{A}^{\mathbb{N}} / \exists \mathcal{S} \in \mathbb{F}, \exists \mathbb{T} \in \mathbb{F}^{\mathbb{N}} / e = (I_e, p_e, d_e) \in a_{\mathcal{S}/\mathbb{T}} \wedge realisable(e, x, (t_j)) \wedge p = p_e\}$

Ainsi, pour qu'un agent source et des agents cibles puissent effectuer l'interaction d'un élément d'assignation e, il est nécessaire de vérifier les critères d'éligibilité, puis les critères de réalisabilité de e. Par conséquent, la sélection d'action d'un agent revient à sélectionner des éléments de son potentiel d'interaction de niveau maximal non vide.

3.8. *Conclusion*

Nous avons défini dans cette section le cadre général de modélisation d'une simulation dans IODA, mais il reste à définir tout l'aspect relatif au passage à l'im-

plémentation, et donc définir concrètement, dans le moteur, l'algorithme de sélection d'actions, qui choisit les interactions parmi toutes celles qui peuvent être effectuées à un moment donné dans tout le système multi-agent, la façon dont est réifié le halo d'un agent, la notion de distance, comment un agent perçoit son voisinage, comment est représenté le temps dans la simulation, etc.

4. De la méthodologie à l'implémentation

La plate-forme[3] que nous avons baptisée JEDI vise à donner une implémentation exacte des principes de la méthodologie IODA, ce qui en fait une des seules qui permette de passer de façon univoque de l'analyse au code. Le générateur de code que nous avons réalisé permet de prototyper des modèles centrés interactions et d'en étudier les propriétés, puis d'en faire une implémentation fidèle dans JEDI.

4.1. *Choix d'implémentation*

Comme le précisent (Michel *et al.*, 2003, Kubera *et al.*, 2007), les choix d'implémentation affectent grandement le domaine des simulations pouvant être réalisées sur une plate-forme, et il est donc important de les préciser. Dans la plate-forme JEDI ces choix portent sur :

nombre de familles d'agents participant à une interaction : le calcul du potentiel d'interaction de niveau p d'un agent $x \in \mathbb{A}$ nécessite de déterminer si chaque élément d'assignation $e = (I, p, d)$ pouvant avoir x comme source est réalisable pour chaque q-uplet d'agents $(t_j)_j j \in [1, q] \in \mathbb{A}^q$ avec $q = card_T(I)$. Par exemple, si un agent x n'est la source que de deux éléments d'assignation e_1 et e_2, et que :

– $e_1 = (I_1, 1, d_1)$ avec $card_S(I_1) = 1$ et $card_T(I_1) = p_1$;
– $e_2 = (I_2, 1, d_2)$ avec $card_S(I_2) = 1$ et $card_T(I_2) = p_2$;
– $\mathcal{N}(x)$ contient n agents,

pour mesurer $P_1(x)$, il faudra effectuer $A^n_{p_1} + A^n_{p_2}$ tests de réalisabilité. En raison de cette complexité, et afin de simuler un nombre important d'agents, seules les interactions I ne faisant intervenir qu'au maximum une famille d'agents source et une famille d'agents cible sont représentées dans JEDI ($card_S(I) = 1$ et $card_T(I) \leq 1$) ;

simulation en temps discret : une simulation est exécutée par étapes (le temps est discrétisé, une étape correspondant à une unité de temps), au cours desquelles les interactions sont exécutées séquentiellement, et où un agent est la source ou la cible d'au maximum une interaction. Un tel choix résout implicitement tous les problèmes liés à la modification concurrente de l'environnement par les agents, en imposant l'ordre dans lequel ils agissent (pseudo-parallélisme). Une simulation en MAX étapes s'exécute alors suivant l'algorithme proposé en figure 3 ;

3. Disponible sur http ://www.lifl.fr/SMAC/projects/ioda

1) Initialiser la date à 0 ;

2) Initialiser la simulation :

a) Créer l'ensemble des agents participant à la simulation ;

b) Initialiser l'état de l'environnement ;

3) Tant que la date $<$ MAX :

a) Effectuer une étape de simulation (voir figure 6), c'est-à-dire faire agir chaque agent au maximum une fois ;

b) Incrémenter la date ;

Figure 3. *Algorithme général suivi lors d'une simulation en MAX étapes implémentée dans JEDI*

simulation située : l'environnement de JEDI est une grille en deux dimensions, dont l'unité de maillage est la case. Dans un tel environnement, les agents disposent d'une propriété position (en coordonnées réelles) définissant dans quelle case ils se situent (voir figure 4). La distance entre deux agents correspond à une distance euclidienne de leurs positions. Le halo d'un agent correspond à la surface qu'il perçoit de l'environnement. Pour des raisons d'optimisation de l'algorithme de perception de l'environnement, nous ne définissons pas la surface de perception d'un agent par une équation paramétrique, qui par ailleurs est peu adaptée à la représentation de surfaces de perception complexes, mais par un ensemble de cases, limitant ainsi les calculs de distance euclidienne aux agents présents dans ces dernières. La justification du gain en performances d'une telle approche est proposée dans (Kubera *et al.*, 2007) ;

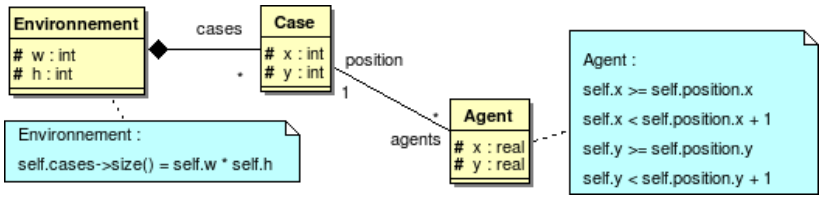

Figure 4. *Diagramme de classes décrivant l'environnement dans JEDI*

tout est agent : dans la plupart des systèmes multi-agents, il y a séparation sémantique entre agents actifs et agents passifs. L'absence de comportement chez les agents passifs (dont le nom varie selon les plates-formes : artefacts, objets, patches, outils, etc.) conduit alors à leur réification sous la forme d'entités indépendantes des agents actifs.

Définition 4.1 *Soit $x \in \mathbb{A}$ un agent dont la famille est $\mathcal{F} \in \mathbb{F}$ ($x \prec \mathcal{F}$).*

– *x est actif* $\Leftrightarrow \exists \mathbb{T} \in \mathbb{F}^{\mathbb{N}} / a_{\mathcal{F}/\mathbb{T}} \neq \emptyset$

– *x est passif* $\Leftrightarrow \forall \mathbb{T} \in \mathbb{F}^{\mathbb{N}}, a_{\mathcal{F}/\mathbb{T}} = \emptyset$

Nous soutenons qu'il est possible d'unifier ces deux notions, puisqu'un agent passif peut devenir actif si au fil des interactions de la simulation il se voit ajouter des assignations. Ceci permet d'avoir une représentation des connaissances homogène, et a l'avantage logiciel de faciliter le passage d'une simulation où un agent passif ne fait qu'être perçu (par exemple un mur qui rend opaque ce qui se situe derrière lui) à une simulation où cet agent interagit avec d'autres agents en tant que source (par exemple l'éboulement de ce mur), et n'engendre pas de surcoût excessif lors de la simulation (cf. section 4.2) ;

reproductibilité des résultats : certaines simulations, pour prouver la validité d'hypothèses portant sur un comportement, nécessitent des mécanismes permettant de reproduire des expérimentations. Il est donc primordial, afin d'assurer la reproductibilité des résultats, de contrôler le générateur de nombres aléatoires ainsi que l'ordonnancement des actions. Dans JEDI, il est possible de fixer la graine ou d'étendre le générateur aléatoire.

4.2. *Décomposition logicielle*

La figure 5 présente un sous-ensemble du diagramme de classes de la plate-forme de simulation JEDI, dans laquelle les interactions sont réifiées (classe *Interaction*), et la matrice d'interactions y est répartie entre toutes les familles d'agents : chaque famille d'agents $\mathcal{F} \in \mathbb{F}$ (représentée par une classe héritant d'*Agent*) dispose d'un ensemble *peutEffectuer* tel que $peutEffectuer(\mathcal{F}) = \{a_{\mathcal{F},\mathbb{T}} \in \mathcal{M}, \forall \mathbb{T} \in \mathbb{F}^{\mathbb{N}}\}$, où $\mathcal{M}$ est la matrice d'interaction utilisée dans la simulation. Ainsi, chaque ligne de la matrice d'interaction est définie dans une famille d'agents.

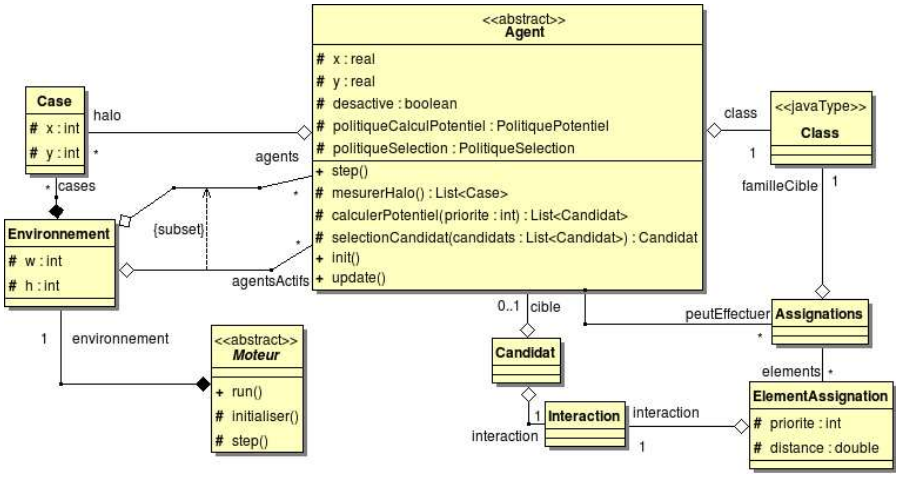

Figure 5. *Sous-ensemble du graphe UML de la plate-forme de simulation JEDI*

La classe abstraite *Moteur* constitue le cœur de la simulation : elle permet d'instancier la simulation, et de lancer *run()* qui effectue la boucle principale. L'exécution d'une étape (cf. figure 6) est déléguée à la méthode *step()* de la classe *Moteur*, qui commence par réorganiser la liste des agents actifs présents dans la simulation selon un critère paramétrable, puis évalue séquentiellement le comportement de chaque agent actif, en effectuant un appel à la méthode *step()* de la classe abstraite *Agent*. Une telle décomposition permet de faire cohabiter dans une même simulation des agents dont la sélection d'action se fait selon des modalités différentes à la fois au niveau du calcul de $P_p(x)$ (méthode *calculerPotentiel*) et de la sélection des éléments de $P_p(x)$ dont il faut exécuter l'interaction (méthode *selectionCandidat*). Cet algorithme garantit de plus que l'interaction effectuée par tout agent est de priorité maximale. En outre, la désactivation (étape h) évite qu'un agent prenne part plusieurs fois à une interaction au cours de la même étape de simulation.

Soit $\mathbb{A}_{act} \subseteq \mathbb{A}$ la liste des agents actifs dans l'environnement.

1) Réorganiser $\mathbb{A}_{act}$ selon un critère paramétrable (cf. section 4.3), par exemple selon un ordre aléatoire (pour un choix équitable),

2) Rendre tous les agents de $\mathbb{A}$ **activables**, et exécuter leur méthode *update()*,

3) Pour chaque agent $a \in \mathbb{A}_{act}$ n'étant pas désactivé :

a) Déterminer quelles cases correspondent au halo de a (construction de $\mathcal{H}_{\mathcal{F}}(a)$ avec $a \prec \mathcal{F}$),

b) Recenser l'ensemble des agents que a considère comme ses voisins (construction de $\mathcal{N}(a)$), et supprimer de cet ensemble les agents non-activables,

c) Initialiser une variable p à la priorité maximale des éléments d'assignation de $peutEffectuer(a)$

d) Calculer $P_p(a)$; si $P_p(a) = \emptyset$, décrémenter p et recommencer,

e) Si on arrive à $p = 0$ et que $P_0(a) = \emptyset$ alors l'agent a ne peut être la source d'aucune interaction, et son étape de simulation s'achève (mais il reste activable)

f) Sinon, sélectionner un élément $P_p(a)$, c'est-à-dire un couple $((I, p, d), (t_j)_{j\in[1,q]})$, avec $q = card_T(I)$, formé d'un élément d'assignation et d'une liste de cibles, selon les modalités de la sélection d'action de a

g) Exécuter l'interaction I avec a comme source et $(t_j)_{j\in[1,q]}$ comme cibles,

h) Désactiver a et tous les agents de $(t_j)_{j\in[1,q]}$.

Figure 6. *Description de l'algorithme utilisé pour une étape de simulation. Il correspond à l'étape 3a de la figure 3*

Une étape de simulation n'étudie que les agents qui peuvent effectuer quelque chose (cf. figure 6 étape 3) : les agents passifs ne sont pas dans cette liste. Si la simulation nécessite une gestion dynamique des éléments d'assignation, il faut bien prendre garde à ajouter à la liste des agents actifs $\mathbb{A}_{act}$ un agent passif qui devient capable de faire quelque chose, et enlever un agent actif qui n'est plus capable de ne rien faire. Ce n'est qu'à ce prix que le système tout agent est optimisé.

4.3. *Paramétrages possibles*

Pour des questions de passage au large échelle en termes de calculs, le concepteur doit pouvoir contrôler la complexité d'un certain nombre de paramètres du simulateur. JEDI est une plate-forme dont la décomposition est modulaire, et définit un ensemble de paramètres allant en ce sens.

Paramétrage des agents : la perception de l'état de l'environnement et des autres agents dans un espace situé pose la question de la surface de perception couverte, et donc de la définition de la fonction $\mathcal{H}_{\mathcal{F}}$. En général, un voisinage de Moore ou de Von Neumann suffit, mais certains cas nécessitent un champ de perception plus sophistiqué (par exemple prenant en compte l'opacité de certains agents) dont JEDI fournit des outils de calcul.

Paramétrage du moteur de sélection d'action : le calcul de $P_p(x)$ peut aussi être paramétré, principalement afin d'optimiser le déroulement de la simulation. En effet, l'évaluation de toutes les combinaisons possibles de cibles nécessite un grand nombre de calculs : par exemple, dans le cas où un agent perçoit 10 voisins, et qu'il dispose de 3 éléments d'assignation, tous de même priorité et dont les interactions sont de cardinalité (1,1), il faudra faire dans tous les cas le test des 30 couples (élément d'assignation, agent cible) possibles. Sous réserve d'être conscient des biais que cela peut introduire, on peut appliquer des heuristiques pour améliorer le nombre de tests effectués en moyenne. JEDI propose par défaut 3 heuristiques :

– ne recenser que le premier couple réalisable (biais introduit par l'ordre d'évaluation des éléments d'assignation et des cibles) ;

– ne recenser qu'une seule cible pour chaque élément d'assignation (biais introduit par l'ordre d'évaluation des cibles) ;

– ne recenser qu'un seul élément d'assignation par cible (biais introduit par l'ordre d'évaluation des éléments d'assignation).

Enfin, la sélection d'action peut elle aussi être paramétrée, et ce de manière indirecte, en redéfinissant les priorités de chaque comportement après la phase de perception, ou directe, en mettant en place des mécanismes de sélection d'élément de $P_p(x)$ particuliers (étape f de la figure 6) . Par défaut, dans JEDI la sélection d'un élément de $P_p(x)$ se fait de manière équitable, en choisissant au hasard un des éléments recensés.

Paramétrage du pseudo-parallélisme : l'organisation de la liste d'agents parcourue à chaque étape de simulation (étape 1 de la figure 6) va grandement influer sur les résultats obtenus, au même titre que le mode de sélection des actions. Nous permettons de définir, au travers de la liste d'agents parcourue, quels sont les agents qui agiront lors d'une étape de simulation et dans quel ordre ils agiront. Dans le cas le plus simple, il s'agit d'une réorganisation aléatoire de la liste des agents de la simulation pour permettre l'équité dans le choix des agents (organisation par défaut dans JEDI), mais peut être le sujet d'une organisation plus intelligente, par exemple en regroupant les agents par familles si l'on souhaite imposer une relation d'ordre entre elles, ou en se

basant sur certaines de leurs propriétés (par exemple : un agent lent ne peut effectuer qu'une interaction toutes les deux étapes de simulation, etc.).

Paramétrage de la portée de la matrice d'interaction : dans JEDI, chaque ligne de la matrice d'interactions est répartie dans une famille d'agents de telle façon que chaque famille d'agents $\mathcal{F} \in \mathbb{F}$ contienne toutes les assignations dans lesquelles elle peut jouer un rôle source. On peut donc optimiser la mémoire utilisée, en définissant l'assignation dans une famille d'agents (l'ensemble des assignation *peutEffectuer* est commun à toutes les instances d'une famille d'agents) plutôt que de les instancier dans chaque agent.

Cette approche n'est pas possible si la matrice d'interaction est dynamique (*i.e.* si les éléments d'assignation d'un agent peuvent évoluer au cours de la simulation) : ces changements ne concernent pas toutes les instances des familles d'agents, et ne doivent donc pas être globales. Dans ce cas, JEDI effectue des copies des éléments d'assignation au sein de chaque instance concernée par les changements.

Paramétrage de la récupération de résultats : aucune interface de récupération de résultats n'est imposée, et elle peut être définie librement par le concepteur. Néanmoins, JEDI propose l'implémentation d'une interface graphique paramétrable affichant les agents et l'environnement pour chaque étape de la simulation sur une grille en deux dimensions.

4.4. *Performances*

Actuellement, une simulation de chambre à particules exécutée sur une machine grand public récente (processeur Pentium Core Duo de 2.4 GHz), JEDI est capable de faire une simulation avec 100 000 agents actifs en temps réel (cf. figure 7).

4.5. *Un exemple simple : l'évolution d'un écosystème*

Pour illustrer notre propos, nous appliquons tout au long de cette section la méthodologie *IODA* sur l'exemple d'un modèle de prédation simplifié, faisant intervenir des agents *Loup*, *Mouton*, *Chèvre* et *Herbe*. Les agents *Herbe* poussent à un certain rythme, les agents *Loup*, *Mouton* et *Chèvre* peuvent *se déplacer* dans l'environnement et *se reproduire* avec un autre agent de la même espèce. De plus, ils *meurent* s'ils ne se nourrissent pas assez, et compensent ceci en *se nourrissant* de proies s'ils ont faim (un loup se nourrit de moutons et de chèvres, un mouton ou une chèvre se nourrit d'herbe). Dans cet exemple, nous faisons volontairement le choix de ne pas différencier la réplication de l'herbe et la reproduction des animaux.

Première étape de IODA : elle conduit à construire la matrice d'interaction du tableau 2. L'observation de cette matrice permet de reprérer deux familles d'agents abstraites : les animaux, qui sont des agents mobiles pouvant mourir, ainsi que les herbivores qui sont des agents pouvant se nourrir d'herbe.

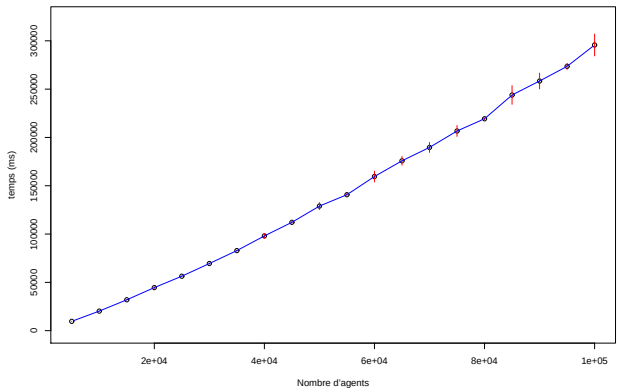

Figure 7. *Courbe affichant le temps de simulation requis en fonction du nombre d'agents (entre* $5\,000$ *et* $100\,000$*) pour effectuer 500 étapes d'une simulation de chambre à particules avec JEDI*

source \ cible	$\emptyset$	Herbe	Mouton	Chèvre	Loup
Herbe	Se reproduire				
Mouton	Mourir Se déplacer	Manger	Se reproduire		
Chèvre	Mourir Se déplacer	Manger		Se reproduire	
Loup	Mourir Se déplacer		Manger	Manger	Se reproduire

Tableau 2. *Matrice d'interaction de la simulation d'un écosystème à 4 familles d'agents*

Deuxième étape : les interactions y sont construites en spécifiant quelles primitives de base elles utilisent (voir tableau 3).

Interaction	Condition	Action
se déplacer(X)	aucune	E.deplacementAleatoire(X) X.vieillir()
se reproduire(X,Y)	aucune	A = E.cloner(X) E.ajouter(A) E.deplacementAleatoire(X)
mourir(X)	X.doitMourir()	E.detruire(X)
manger(X,Y)	X.avoirFaim()	X.inhiberFaim() E.detruire(Y)

Tableau 3. *Définition des interactions de la simulation de prédation. Les paramètres des interactions correspondent à différents rôles nécessaires à l'interaction (dans ce cas X est le rôle « agent source » et Y « agent cible »). E représente l'environnement*

Troisième étape : d'après cette spécification, la famille d'agents Herbe ne devra implémenter aucune primitive, et les familles d'agents Loup, Mouton et Chèvre devront implémenter les primitives de perception *doitMourir()*, *avoirFaim()*, et la primitive d'action *inhiberFaim()*.

Quatrième étape : dans ce modèle, la reproduction entre animaux ne peut avoir lieu que s'ils sont proches (distance de 1), les Chèvres et Moutons ne peuvent manger de l'Herbe que si elle est sous eux (distance de 0), les Loup peuvent manger que des Moutons et Chèvres se situant à proximité d'eux (distance de 1). De plus, nous imposons des priorités dans les interactions : la mort est l'interaction la plus prioritaire (si l'agent doit mourir, il n'a pas à tenter d'interagir avec l'environnement), le déplacement est le comportement par défaut, manger est plus prioritaire que se Reproduire, et le loup préfère manger des Chèvres, donc manger une chèvre est plus prioritaire que manger un mouton. Ces remarques aboutissent au raffinement de la matrice d'interaction du tableau 4, ainsi qu'à la spécification des primitives de chaque famille d'agents. Par soucis de concision, nous ne détaillons pas les primitives, leur description étant explicite au travers des contraintes OCL de la figure 8.

source \ cible	∅	Herbe	Mouton	Chèvre	Loup
Herbe	(Se reproduire ;0)				
Mouton : Animal, Herbivore	(Mourir ;3) (Se déplacer ;0)	(Manger ;2 ;0)	(Se reproduire ;1 ;1)		
Chèvre : Animal, Herbivore	(Mourir ;3) (Se déplacer ;0)	(Manger ;2 ;0)		(Se reproduire ;1 ;1)	
Loup : Animal	(Mourir ;4) (Se déplacer ;0)		(Manger ;2 ;1)	(Manger ;3 ;1)	(Se reproduire ;1 ;1)

Tableau 4. *Matrice d'interaction du modèle de JEDI de la simulation d'un écosystème à 4 familles d'agents. Les éléments de cette matrice sont des triplets* (Interaction, priorité, garde de distance) *ou des couples* (Interaction, priorité). *Les priorités sont définies par ordre croissant*

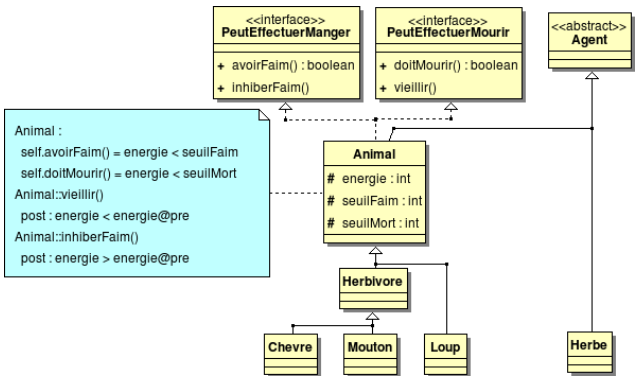

Figure 8. *Sous-ensemble du diagramme UML de cette simulation dans JEDI*

Cinquième étape : ici, il n'y a pas de modification dynamique des comportements.

Sixième étape : elle est sans objet ici, car la plateforme JEDI est, par nature, adaptée à l'implémentation de ce modèle.

Une fois la 6^{eme} étape terminée, JEDI-Builder peut générer le squelette de toutes les classes et interfaces nécessaires à la simulation. Il ne reste à l'utilisateur qu'à :

– compléter le corps de la méthode récupérant le halo de l'agent ;

– compléter le corps des primitives de chaque agent ;

– compléter le corps de la condition de chaque interaction ;

– compléter le corps de la séquence d'actions de chaque interaction ;

– compléter l'initialisation de la simulation.

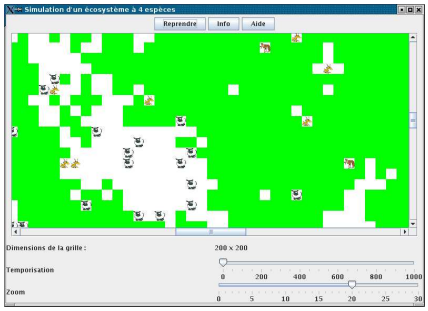

Figure 9. *Capture d'écran de cette simulation dans JEDI*

La figure 8 propose le diagramme UML d'une implémentation de ce modèle dans JEDI, et la figure 9 présente une capture d'écran de l'exécution de la simulation ainsi construite. Sur la matrice du tableau 2, on observe que les trois agents Mouton, Loup et Chèvre participent aux mêmes interactions (et implémentent donc les mêmes primitives), et il est donc très facile de modifier les relations que ces agents entretiennent sans avoir à effectuer des changements majeurs dans la simulation.

4.6. *Conclusion*

Dans cette section, nous avons proposé un cadre méthodologique de conception de simulation multi-agent faisant apparaître explicitement la représentation des connaissances des agents de l'étape de modélisation (IODA) à l'étape d'implémentation (JEDI). Cette représentation ne présume pas des capacités d'un agent, aussi bien en termes de perception de l'environnement et des autres agents qu'en termes de détermination des interactions qu'il peut adopter et en termes de sélection d'une interaction à exécuter Elle permet donc, à l'aide de sa grande paramétrabilité, de faire cohabiter des agents structurellement très différents tout en conservant des interactions suffisamment génériques pour être réutilisées par tout type d'agents.

5. Conclusion

La conception de simulations informatiques nécessite de trouver un compromis entre la précision du modèle utilisé, qui doit définir un modèle assez précis pour être implémenté sans ambiguïté, et la généricité du modèle, qui doit être facilement descriptible et réutilisable. La plupart des plates-formes de simulation délaissent l'un de ces deux aspects, certaines ne définissant même pas les caractéristiques du modèle qu'elles utilisent.

Dans cet article nous présentons l'architecture de la plate-forme de simulation multi-agent JEDI, qui constitue une implémentation de la méthodologie IODA. Cette méthodologie sépare le déclaratif du procédural, et permet donc de définir séparément les interactions qu'entretiennent les agents de la façon dont elles sont sélectionnées, permettant ainsi de construire des ontologies d'interactions réutilisables. De plus, la représentation des comportements sous forme de matrice d'interaction permet de modéliser des simulations large échelle (faisant intervenir un grand nombre d'agents et d'interactions différents) et de réaliser automatiquement l'implémentation de la simulation correspondante à l'aide d'un générateur de code.

La plate-forme de simulation JEDI donne une implémentation simple de la méthodologie IODA, et fournit en particulier une architecture de sélection d'action générique qui ne présage pas du degré de cognition des agents. La simulation peut y être paramétrée finement, de manière à s'adapter le plus efficacement possible à la simulation exécutée, tout en conservant le contrôle des biais éventuels.

Actuellement, JEDI impose des restrictions à propos de la cardinalité des interactions à cause de la complexité de modélisation et la complexité algorithmique qu'elles peuvent apporter. Notre objectif à court terme est d'étendre JEDI à tout type de cardinalité, ainsi que de déterminer le formalisme à utiliser pour décrire la séquence d'actions, les conditions des interactions, ou les propriétés devant être respectées par un simulateur sur lequel implémenter un modèle particulier construit avec IODA. IODA et JEDI permettent en outre d'étudier formellement divers critères de complexité des modèles de simulation et de leur implémentation, par exemple au moyen de l'analyse des matrices d'interaction et des rétroactions qu'elles contiennent.

Remerciements

Ce travail est cofinancé par le contrat de plan état-région TAC du Nord-Pas de Calais et les fonds européens FEDER.

6. Bibliographie

Bensaid N., Mathieu P., « A Hybrid and Hierarchical Multi-Agent Architecture Model », *Proceedings of the Second International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 1997.

Briot J.-P., Meurisse T., Peschanski F., « Une expérience de conception et de composition de comportements d'agents à l'aide de composants », *L'Objet, Revue des Sciences et Technologies de l'Information*, 2006.

Brooks R. A., « A Robust Layered Control System For A Mobile Robot », *iEEE journal of robotics and automation*, March, 1986.

Burkhart R., « The Swarm Multi-Agent Simulation System », *Position Paper for OOPSLA '94 Workshop on 'The Object Engine'*, 1994.

Doi T., Tahara Y., Honiden S., « IOM/T : an interaction description language for multi-agent systems », *AAMAS '05 : Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, ACM, 2005.

Giavitto J.-L., « Topological Collections, Transformations and Their Application to the Modeling and the Simulation of Dynamical Systems », *Proccedings of Rewriting Techniques and Applications (RTA'03)*, Valencia, Spain, 2003.

Gutknecht O., Ferber J., Michel F., « Integrating tools and infrastructures for generic multi-agent systems », *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, Montreal, Canada, 2001.

Kubera Y., Mathieu P., Picault S., « La complexité dans les simulations multi-agents », *in* , Cépaduès-Editions (ed.), *Actes des Journées Francophones sur les Systèmes Multi-Agents (JFSMA07)*, Carcassonne, France, 2007.

Kubera Y., Mathieu P., Picault S., « Interaction-Oriented Agent Simulations : From Theory to Implementation », *Proceedings of ECAI 08*, Patras Greece, July, 2008.

Mathieu P., Picault S., Routier J.-C., « Donner corps aux interactions (l'interaction enfin concrétisée) », *Actes de la conférence MFI07*, Paris, France, 2007.

Mathieu P., Routier J.-C., Urro P., « Un modèle de simulation agent basé sur les interactions », *Actes des Premières Journées Francophones sur les Modèles Formels de l'Interaction (MFI'01)*, Toulouse, France, Mai, 2001.

Michel F., Gouaïch A., Ferber J., « Weak Interaction and Strong Interaction in Agent Based Simulations », *Proceedings of the 4th Workshop on Multi-Agent-Based Simulation (MABS)*, Melbourne, Australia, July, 2003.

Papadopoulos G., Arbab F., « Coordination Models and Languages », *Advances in Computers*, *Academic Press*, Aug, 1998.

Pokahr A., Braubach L., « An Architecture and Framework for Agent-Based Web Applications », *Proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2007)*, Leipzig, Germany, September, 2007.

Ricordel P.-M., Demazeau Y., « La plate-forme Volcano - Modularité et réutilisation pour les systèmes multi-agents », *Technique et Science Informatiques (TSI)*, 2002.

von Bertalanffy L., *General System Theory : Foundations, Development, Applications*, George Braziller Revised edition, 1976.

Wilenski U., Netlogo, http ://ccl.northwestern.edu/netlogo/, Center for Connected Learning (CCL) and Computer-Based Modeling, 1999.

Wooldridge M., *Reasoning about Rational Agents*, Intelligent robotics and autonomous agents, The MIT Press, Cambridge, Massachussetts/London, England, June, 2000.