IODA comme approche transversale de conception

Plan du chapitre :

Dans ce chapitre, nous caractérisons comment construire une approche transversale de conception de simulations à partir du modèle formel décrit dans le chapitre précédent. Nous décrivons pour cela des principes permettant à la fois de construire un modèle de manière incrémentale (une méthodologie de conception), mais aussi de l'implémenter à l'aide d'algorithmes de simulation se basant sur les données du modèle. Nous profitons de plus de ce chapitre pour montrer comment notre approche se positionne vis à vis de quatre questions se posant systématiquement dans toute approche de conception transversale :

- *Toute entité est-elle un agent ?*
- *Une méthodologie de conception doit-elle être dogmatique ?*
- *Faut-il prendre en compte les performances lors de la modélisation ?*
- *Faut-il tout décrire au sein d'une règle (dans notre cas d'une interaction) ?*

La section 4.1 décrit comment remplir progressivement le modèle formel décrit dans le chapitre 3, à l'aide de représentations graphiques facilement manipulables. La section 4.2 étudie les quatre questions mentionnées précédemment et caractérise comment IODA *se positionne par rapport à ces problèmes. Enfin, la section 4.3 décrit les algorithmes tirant parti des informations présentes dans un modèle formel* IODA *afin d'exécuter une simulation. Nous en dégageons des avantages de* IODA *facilitant la conception de simulations.*

Une approche transversale de conception est caractérisée par deux phases : la phase de construction d'un modèle à partir de spécifications formulées par le langage naturel et la phase de traduction de ce modèle en un ensemble d'algorithmes pouvant être implémentés.

4.1 La méthodologie de conception IODA

Le rôle du formalisme du modèle est de fournir une structure précise à la description d'une simulation, afin de donner un cadre formel permettant sa spécification. En théorie, la connaissance d'un modèle formel permet de décrire de manière précise tout type phénomène dans la limite de l'expressivité du formalisme utilisé. En pratique, de tels modèles deviennent très rapidement impossibles à spécifier uniquement à l'aide de ce formalisme. En effet, le formalisme du modèle est un tout dont certains éléments très précis ne peuvent être spécifiés dès le début du processus de conception. Par exemple, il est inutile de s'intéresser au processus de sélection d'interaction d'une entité sans connaître ce que l'entité est capable de faire. De même, l'identification d'attributs d'une famille d'entités n'a pas de sens en dehors de la spécification précise du comportement des entités, *etc.*

Dans notre approche, nous associons donc au formalisme du modèle IODA une méthodologie de conception que nous appelons méthodologie IODA. Cette méthodologie fournit un procédé permettant

de concevoir un modèle graduellement en commençant par ses aspects macroscopiques (les interactions entretenues par les entités), pour finir sur ses aspects microscopiques (le détail des actions que les entités entreprennent). Cette section a pour rôle de de décrire ce procédé, présenter les éléments graphiques permettant cette spécification ainsi qu'identifier comment les éléments graphiques permettent de remplir le modèle formel de la simulation spécifiée.

4.1.1 Processus général

La méthodologie IODA repose sur la construction de deux matrices afin de décrire la logique générale d'une simulation : la matrice d'interaction brute et la matrice d'interaction raffinée. Ces matrices permettent une conception graduelle du modèle de la simulation à deux niveaux.

Conception graduelle à deux niveaux.

D'une part, elles permettent de décrire graduellement la logique générale de la simulation, qui passe de l'identification du graphe des interactions liant les entités (la matrice d'interaction brute) à son interprétation en termes d'actions que les entités sont capables d'initier (les éléments contenus dans une ligne de la matrice), pour finir le comportement général des entités en attribuant des priorités aux différentes capacités d'une entité (matrice d'interaction raffinée). Ainsi, le schéma général d'une méthodologie de conception transversale, tel qu'il est décrit dans la figure 2.5 page 44, est respecté, assurant une spécification de plus en plus précise du fonctionnement du phénomène.

D'autre part, pour des raisons précédemment annoncées comme nécessaires au polymorphisme des interactions, les interactions ont une description indépendante des entités pouvant y participer. La seconde justification d'une telle structure est qu'elle permet de plus de décrire les connaissances des entités de manière graduelle. En effet, la description de ce que les entités sont capables de faire est divisé en deux phases. Elle commence par une description abstraite valide pour toutes les entités de la simulation (les préconditions, déclencheurs et actions d'une interaction), pour ensuite être raffinée et précisée, au moment où les primitives abstraites de chaque entité sont décrites.

Caractérisation du processus de conception.

La méthodologie de conception permettant de concevoir des modèles IODA suit un processus décomposé en treize étapes résumées sur la figure 4.1. Chaque étape facilite la spécification d'un élément du modèle formel en :

- fournissant une représentation graphique permettant une description visuelle du modèle ;
- décrivant comment interpréter automatiquement cette représentation afin de compléter le modèle formel ;
- caractérisant les conséquences de cette étape sur le modèle spécifié dans les autres étapes de la méthodologie ;
- identifiant les étapes devant être préalablement spécifiées pour passer à cette étape.

La construction d'un modèle selon la méthodologie IODA se fait en trois phases s'attachant chacune à des informations de granularités différentes : la phase de spécification *des interactions entre les entités*, la phase de spécification *du comportement général des entités* et enfin la phase de spécification *du comportement précis des entités*.

Les lettres dans la figure sont utilisées uniquement pour nommer chaque nœud. Elles ne présument en aucun cas d'un quelconque ordre dans lequel suivre la méthodologie, qui peut être arbitraire du moment qu'une étape est spécifiée lorsque les étapes dont elle dépend sont déjà spécifiées. Une discussion concernant ce point est effectuée plus tard dans ce chapitre, dans la section 4.2.2 page 114.

4.1.2 Spécification des interactions entre les entités

La première partie de la méthodologie consiste à identifier trois éléments fondamentaux à toute simulation centrée sur les interaction :

- la topologie de l'environnement utilisé (« A » dans la figure 4.1) afin de donner un sens à la notion de distance entre deux entités ;

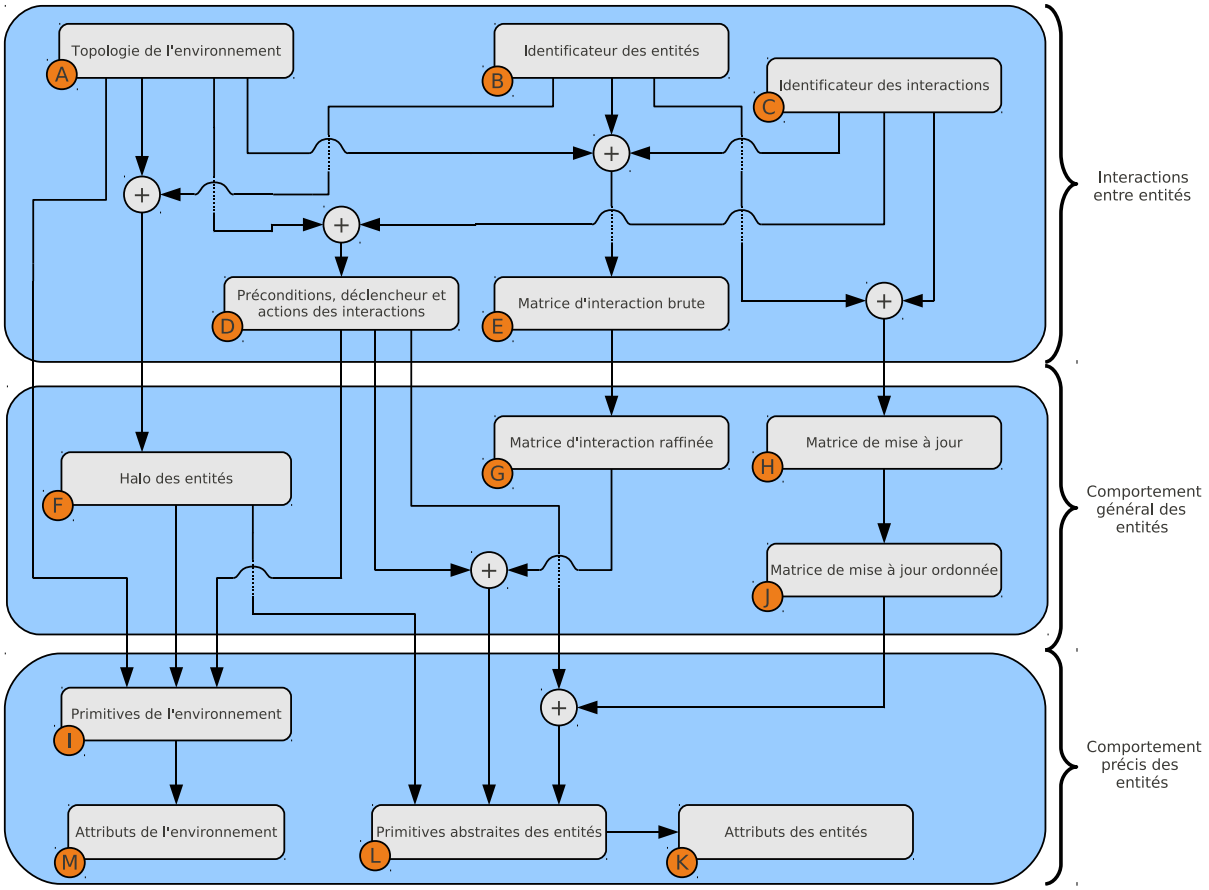


FIGURE 4.1 – Cheminement suivi dans la méthodologie IODA pour concevoir un modèle. Une flèche partant d'un nœud X vers un nœud Y signifie que pour commencer les spécifications Y d'un élément du modèle, son pendant dans X doit avoir été spécifié. Par exemple, la priorité d'un élément d'assignation dans la matrice d'interaction raffinée (G) n'est faite qu'après avoir spécifié l'élément d'assignation dans la matrice d'interaction brute (E). Deux transitions allant d'un nœud X_1 (resp. X_2) à un nœud Y signifient que le nœud Y peut être spécifié partiellement une fois que le nœud X_1 (resp. X_2) est spécifié. Une transition passant par un nœud « + » et reliant plusieurs nœuds $X_1 \ldots X_n$ à un nœud Y signifie que les nœuds $X_1 \ldots X_n$ doivent avoir été spécifiées pour commencer les spécifications du nœud Y. Par exemple, la description d'une primitive abstraite dans une entité (L) ne sera faite que si les déclencheurs, préconditions et actions de l'interaction la manipulant sont décrites (D) et si l'entité est capable d'initier ou subir cette interaction (G ou J).

- l'identifiant des différentes familles d'entités participant au phénomène simulé (« B » dans la figure 4.1) ;
- l'identifiant des différentes interactions pouvant survenir dans le phénomène (« C » dans la figure 4.1) ;
- la matrice d'interaction brute du phénomène simulé (« E » dans la figure 4.1) ;
- les préconditions, déclencheur et actions de chaque interaction du phénomène simulé (« D » dans la figure 4.1) afin de fixer la sémantique de l'interaction.

Topologie de l'environnement (« A »). La spécification de la topologie de l'environnement consiste à donner un sens à la notion de distance dans la simulation, *i.e.* à lui donner une unité, afin de pouvoir attribuer des gardes de distance dans la matrice d'interaction brute. Elle constitue un pré-requis à l'expression du halo (étape « F »), à la description des interactions (étape « D ») et à la description de la matrice d'interaction brute (étape « E »). À ce stade de la méthodologie, aucune description formelle n'est fournie à la notion de distance.

Identifiant des familles d'entités (« B »). Aucune représentation particulière n'est préconisée pour la spécification des identifiants des familles d'entités, puisque cette étape consiste simplement à dresser une liste d'identifiants, *i.e.* de chaînes de caractères. Cette étape permet de débuter la construction de l'ensemble $\mathbb{F}$: pour chaque identifiant id de la liste, une famille d'entités $\langle id, \emptyset, \emptyset, ?, \emptyset, \emptyset, \emptyset, ? \rangle$ est ajoutée à $\mathbb{F}$.

Identifiant des interactions (« C »). Tout comme lors de l'étape « B », aucune représentation particulière n'est préconisée pour la spécification des identifiants des interactions, puisqu'elle consiste aussi à dresser une liste d'identifiants. Cette étape permet de débuter la construction de l'ensemble $\mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)}$: pour chaque identifiant id de la liste, une interaction $\langle id, \emptyset, \emptyset, \emptyset, \emptyset, ?, ?, ? \rangle$ est ajouté à $\mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)}$.

À ce stade de la méthodologie, la cardinalité des interactions n'est pas connue. Elle sera déterminée soit lors de l'étape de description des interactions « D », soit lors la construction de la matrice d'interaction brute « E ».

Matrice d'interaction brute (« E »). Une fois les étapes « A », « B » et « C » terminées, il devient possible de construire le graphe des interactions entre les entités de la simulation (« E » dans la figure 4.1). Cette étape consiste dans un premier temps a construire une matrice d'interaction brute vide, en ajoutant une ligne et une colonne pour chaque identifiant de famille d'entités spécifié lors de l'étape « B ». Cette étape revient à créer une matrice telle que $\forall \mathcal{S} \in \mathbb{F}, \forall \mathcal{T} \in \mathbb{F} \cup \{\emptyset\}, \mathcal{M}(\mathcal{S}, \mathcal{T}) = \emptyset$.

Dans un second temps, cette matrice est remplie en plaçant des identifiants d'interaction, issus de l'étape « C », à l'intersection d'une ligne et d'une colonne de la matrice, afin de définir les interactions ayant lieu entre les entités de la simulation, ou les interactions dégénérées que les entités ont la capacité d'initier. Si un identifiant est ajouté dans une assignation individuelle (*i.e.* l'intersection d'une ligne et d'une colonne associées à des identifiants de famille d'entité), une garde de distance lui est associée. Cette étape permet ainsi d'obtenir une matrice dont l'apparence est illustrée sur la figure 3.12a page 91.

A partir de cette représentation graphique, il est possible de déduire l'ensemble des éléments d'assignation du modèle. En effet, si $\mathcal{S} \in \mathbb{F}$, $\mathcal{T} \in \mathbb{F}$, $\mathcal{I} \in \mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)}$ et $r \in \mathbb{R}^+$, on a :

- si l'identifiant $id(\mathcal{I})$ apparaît sous la forme $\Big(id(\mathcal{I}), d = r\Big)$ à l'intersection de la ligne associée à $id(\mathcal{S})$ et de la colonne associée à $id(\mathcal{T})$, alors un élément d'assignation individuel $(\mathcal{I}, r, \mathcal{S}, \mathcal{T})$ est ajouté à $\mathcal{M}(\mathcal{S}, \mathcal{T})$. Cet élément est aussi ajouté à $ligne(\mathcal{S})(\mathcal{T})$ et à $colonne(\mathcal{T})(\mathcal{S})$;
- si l'identifiant $id(\mathcal{I})$ apparaît sous la forme $\Big(id(\mathcal{I})\Big)$ à l'intersection de la ligne associée à $id(\mathcal{S})$ et de la colonne associée à $\emptyset$, alors un élément d'assignation dégénéré $(\mathcal{I}, \mathcal{S})$ est ajouté à $\mathcal{M}(\mathcal{S}, \emptyset)$. Cet élément est aussi ajouté à $ligne(\mathcal{S})(\emptyset)$.

À ce stade de la spécification, la cardinalité des interactions est déduite de la nature des cellules de la matrice où elles sont disposées : une interaction figurant dans la colonne "$\emptyset$" (*i.e.* dégénérée) devient une interaction dégénérée. Sinon, elle est considérée par défaut comme une interaction individuelle :

$$\forall \mathcal{I} \in \mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)} \begin{cases} \Big(\exists \mathcal{S} \in \mathbb{F}, \exists \mathcal{T} \in \mathbb{F}, \exists r \in \mathbb{R}^+ | (\mathcal{I}, r, \mathcal{S}, \mathcal{T}) \in \mathcal{M}(\mathcal{S}, \mathcal{T})\Big) \Rightarrow \Big(card(\mathcal{I}) = (1,1) \wedge \mathcal{I} \in \mathbb{I}_{(1,1)} \wedge \mathcal{I} \notin \mathbb{I}_{(1,0)}\Big) \\ \Big(\exists \mathcal{S} \in \mathbb{F} | (\mathcal{I}) \in \mathcal{M}(\mathcal{S}, \emptyset)\Big) \Rightarrow \Big(card(\mathcal{I}) = (1,0) \wedge \mathcal{I} \in \mathbb{I}_{(1,0)} \wedge \mathcal{I} \notin \mathbb{I}_{(1,1)}\Big) \end{cases}$$

Préconditions, déclencheur et actions des interactions (« D »). La définition des identifiants des différentes interactions survenant dans la simulation (étape « C »), conjointement à la spécification de la topologie de l'environnement (étape « A »), débouche également sur une étape où un sens est donné aux interaction (étape « D »). La sémantique générale des interactions est attribuée indépendamment des spécificités des entités. Elle consiste à en décrire le déclencheur, les préconditions et les actions. Cette description peut s'acquérir de deux manières. La première consiste à utiliser une interaction déjà définie dans une librairie existante, dans quel cas la spécification se limite à l'identification de l'interaction utilisée dans la librairie. Il faut toutefois s'assurer que la signature de l'environnement dans cette interaction n'entre pas en conflit avec la topologie identifiée lors de l'étape « A ». La deuxième manière de spécifier une interaction passe par la description précise de son déclencheur, de ses préconditions et de ses actions.

La description d'une interaction se fait selon les cinq étapes qui suivent, dont l'objectif est d'aboutir à une représentation graphique de la forme présentée dans la figure 4.2 pour décrire le sens de l'interaction, ainsi que les représentations graphiques de la forme présentée dans la figure 4.3 pour décrire la signature de chaque entité impliquée dans l'interaction, ainsi que la signature de l'environnement dans cette interaction :

1. L'identification de la cardinalité de l'interaction ;
2. L'attribution d'un nom aux différentes entités pouvant participer à l'interaction ;
3. La description des préconditions, déclencheur et actions à l'aide d'algorithmes ;
4. L'identification de la signature de chaque entité participant à l'interaction ;
5. L'identification de la signature de l'environnement dans l'interaction

La construction de cette représentation graphique commence par l'identification de la cardinalité de l'interaction. Elle consiste soit à interpréter le sens souhaité de l'interaction et d'en déduire la cardinalité, soit à récupérer la cardinalité déduite de la place de l'interaction dans la matrice d'interaction brute définie lors de l'étape « B », ou dans la matrice de mise à jour lors de l'étape « H ». Une fois la cardinalité de l'interaction identifiée, la représentation graphique du modèle de l'interaction est fixée à celle de la figure 4.2(a) si l'interaction est individuelle, et à celle de la figure 4.2(b) si l'interaction est dégénérée.

La seconde étape consiste à attribuer un nom aux entités participant à l'interaction. Puisque nous nous plaçons uniquement dans le cadre des interactions individuelles et dégénérées, nous avons fixé les noms des entités à {Source} si l'interaction est dégénérée et à {Source, Cible} si l'interaction est individuelle. Cette étape consiste donc à modifier l'entête de la représentation graphique pour « IDENTIFIANT DE L'INTERACTION(Source, Cible) » si l'interaction est individuelle, ou pour IDENTIFIANT DE L'INTERACTION(Source) si l'interaction est dégénérée.

Ensuite, la description des préconditions, actions et déclencheur de l'interaction consiste à écrire un algorithme manipulant à la fois les primitives de l'environnement, des primitives d'action et des primitives de perception. Bien que nous ne fournissions pas de description formelle du langage utilisé pour décrire ces algorithmes, nous imposons toutefois d'identifier explicitement les primitives abstraites des entités, en utilisant une syntaxe de la forme :

« *Nom d'une entité* »'.' « *Identifiant de primitive abstraite* »'('« *Arguments de la primtive abstraite* »')'

Nous préconisons de plus d'identifier les primitives de l'environnement sous la forme :

'Environnement.'« *Identifiant de primitive de l'environnement* » '('« *Arguments de la primitive de l'environnement* »')'

Une illustration d'une telle description est fournie dans la figure 4.4 pour l'interaction MANGER.

Puisque nous nous plaçons dans le cadre d'interactions de cardinalité $(1,1)$ ou $(1,0)$, une interaction ne définit au plus que deux signatures d'entités, que nous avons appelées $sign_{source}$ et $sign_{cible}$. Les primitives contenues dans ces signatures sont déduites de l'algorithme associé aux préconditions, aux actions et au

« interaction individuelle » IDENTIFIANT DE L'INTERACTION(Source, Cible)		
Signatures	Nom	Identifiant de la signature
	Source	*Identifiant de signature d'entité*
	Cible	*Identifiant de signature d'entité*
	Environnement	*Identifiant de signature de l'environnement*
Déclencheur	*% Déclencheur de l'interaction, décrit sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", de l'entité nommée "Cible", ainsi que des primitives de l'environnement.*	
Préconditions	*% Préconditions de l'interaction, décrites sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", de l'entité nommée "Cible", ainsi que des primitives de l'environnement*	
Actions	*% Actions de l'interaction, décrites sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", de l'entité nommée "Cible", ainsi que des primitives de l'environnement*	

(a) Représentation graphique d'une interaction individuelle

« interaction dégénérée » IDENTIFIANT DE L'INTERACTION(Source)		
Signatures	Nom	Identifiant de la signature
	Source	*Identifiant de signature d'entité*
	Environnement	*Identifiant de signature de l'environnement*
Déclencheur	*% Déclencheur de l'interaction, décrit sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", ainsi que des primitives de l'environnement.*	
Préconditions	*% Préconditions de l'interaction, décrites sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", ainsi que des primitives de l'environnement.*	
Actions	*% Actions de l'interaction, décrites sous la forme d'un algorithme. Il manipule des primitives de l'entité nommée "Source", ainsi que des primitives de l'environnement.*	

(b) Représentation graphique d'une interaction dégénérée

FIGURE 4.2 – Forme générale de la représentation graphique permettant la spécification d'une interaction.

« signature d'une entité » Identifiant de la signature				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
...	...	...	...	...

(a) Représentation graphique d'une signature d'entité dans une interaction

« signature de l'environnement » Identifiant de la signature				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
...	...	...	...	...

(b) Représentation graphique d'une signature de l'environnement dans une interaction

FIGURE 4.3 – Forme générale de la représentation graphique permettant la spécification d'une signature d'entité dans une interaction.

« interaction individuelle » MANGER(Source, Cible)		
Signatures	Nom	Identifiant de la signature
	Source	SourceDeManger
	Cible	CibleDeManger
	Environnement	EnvironnementDansManger
Déclencheur	Source.aFaim()	
Préconditions	Cible.estSain()	
Actions	Source.diminuerFaim(Cible.valeurEnergetique()) Environnement.retirer(Cible)	

FIGURE 4.4 – Spécification d'une interaction dont l'identifiant est MANGER et dont la cardinalité est $(1, 1)$.

« signature d'entité » SourceDeManger				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
aFaim	Booléen	—	—	Retourne vrai si l'entité ressent la sensation de faim.
diminuerFaim	—	valeurEnergetique	Nombre Entier	Diminue la sensation de faim de l'entité d'un montant égal à la valeur énergétique fournie en paramètre.

« signature d'entité » CibleDeManger				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
estSain	Booléen	—	—	Retourne vrai si l'entité représente un aliment sain.
valeurEnergetique	Nombre Entier	—	—	Détermine la valeur énergétique de cette entité.

FIGURE 4.5 – Spécification de la signature des entités dans l'interaction MANGER précédemment décrite dans la figure 4.4.

« signature de l'environnement » EnvironnementDansManger				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
retirer	—	e	Entité	Retire une entité de l'environnement.

FIGURE 4.6 – Spécification de la signature de l'environnement dans l'interaction MANGER, précédemment décrite dans la figure 4.4.

déclencheur : $sign_{source}$ (respectivement $sign_{cible}$) contient toutes les primitives des algorithmes des préconditions, du déclencheur et des actions dont l'appel est préfixé par "Source" (resp. "Cible"). Nous prenons pour convention de donner l'identifiant « SourceDeId » à $sign_{source}$ (avec Id l'identifiant de l'interaction), et l'identifiant « CibleDeId » à $sign_{cible}$. Une illustration d'une telle description est fournie dans la figure 4.5 pour l'interaction MANGER, qui définit deux signatures d'une entité dans l'interaction. La première, dont l'identifiant est « SourceDeManger », représente la signature de l'entité dont le nom est « Source » dans l'interaction, qui est la source de l'interaction. La seconde, dont l'identifiant est « CibleDeManger », représente la signature de l'entité dont le nom est « Cible » dans l'interaction, qui est la cible de l'interaction.

Enfin, la signature de l'environnement dans l'interaction est construite de la même manière que la signature d'une entité, en analysant les algorithmes et décelant les primitives dont l'appel est préfixé par « Environnement ». Dans le cas de l'interaction MANGER, cette spécification est résumée sur la figure 4.6.

La construction de la signature des entités et de l'environnement dans l'interaction peut être automatisé. Par conséquent, la spécification d'une interaction consiste en pratique à fournir uniquement la description présente dans la figure 4.2.

Lien entre matrice d'interaction brute et spécification des interactions. La possibilité d'utiliser des librairies d'interactions prédéfinies fait que l'étape de spécification de la matrice d'interaction brute « E » ne survient pas systématiquement avant l'étape de spécification des interactions « D ». Il s'en découle un problème auquel une attention particulière doit être apportée : puisque le fait de placer une interaction dans une cellule de la matrice d'interaction brute permet d'en déduire la cardinalité, il se révèle primordial de s'assurer que la position de chaque interaction dans la matrice n'entre pas en conflit avec le nombre d'arguments du déclencheur, des préconditions et des actions de l'interaction (et réciproquement). En particulier, une interaction figurant dans une assignation individuelle, *i.e.* figurant dans une colonne associée à un identifiant d'entité, ne doit pas avoir deux arguments dans son déclencheur, ses préconditions et ses actions (et réciproquement). De même, une interaction figurant dans une assignation dégénérée, *i.e.* figurant dans la colonne nommée $\emptyset$, ne doit pas avoir un seul argument dans son déclencheur, ses préconditions et ses actions (et réciproquement).

4.1.3 Spécification du comportement général des entités.

Cette deuxième partie de la méthodologie se concentre sur la description générale du comportement des entités. Il y est initialement possible de spécifier trois de ses étapes en parallèle :
- le halo de chaque famille d'entités (étape « F ») ;
- la matrice d'interaction raffinée (étape « G ») ;
- la matrice de mise à jour (étape « H »).

Halo des familles d'entités (« F »). Cette étape consiste à identifier comment, à partir de la seule notion de distance et de la connaissance de la topologie de l'environnement, il est possible aux entités de définir leur voisinage. La halo peut être considéré comme une primitive particulière de l'entité et, à ce titre, être décrite à l'aide d'algorithmes, manipulant des primitives de l'environnement et des primitives de perception ou d'action de l'entité.

Exemple.
Voici un exemple de halo d'une famille d'entités $\mathcal{F}$, désignant une entité comme perçue si elle figure dans la mémoire de l'entité, où si elle se situe à une distance inférieure ou égale à son acuité visuelle.

$halo\Big(\mathcal{H}(\mathcal{F})\Big)(e,e_2)$:
| **retourner** $e.aDansSaMemoire(e_2)$ **ou** $environnement.distance(e,e_2) \leq e.acuiteVisuelle()$

Ce halo implique l'ajout de deux primitives de perception dans la famille de l'entité e : $\langle aDansSaMemoire, booleen, \{Entite\}\rangle$ et $\langle acuiteVisuelle, Nombre\ a\ Virgule\ Flottante, \emptyset\rangle$.

La seule description de cet algorithme permet de définir complètement le halo d'une famille d'entités $\mathcal{F} \in \mathbb{F}$. En effet, chaque occurrence d'une primitive de l'environnement dans l'algorithme mène à l'ajout d'une primitive dans la signature $sign_{env}\Big(\mathcal{H}(\mathcal{F})\Big)$ de l'environnement dans le halo. De même, chaque occurrence d'une primitive d'action ou de perception dans l'algorithme mène à l'ajout d'une primitive dans la signature $sign_{entite}\Big(\mathcal{H}(\mathcal{F})\Big)$ de la famille d'entités $\mathcal{F}$ dans son halo. Le halo de chaque famille d'entités $\mathcal{F} \in \mathbb{F}$ est alors défini par le triplet $\langle halo, sign_{env}\Big(\mathcal{H}(\mathcal{F})\Big), sign_{entite}\Big(\mathcal{H}(\mathcal{F})\Big)\rangle$. Leur représentation graphique est similaire à celle présentée lors de la description de la spécification d'une interaction.

Matrice d'interaction raffinée (« G »). Le deuxième élément pouvant être spécifié au début de cette seconde phase de la méthodologie IODA est la matrice d'interaction raffinée, qui permet de décrire le processus comportemental d'une entité réactive. Cette spécification consiste à générer, à partir de la matrice d'interaction brute spécifiée lors de l'étape « E », une nouvelle matrice, dans laquelle un nombre entier est associé à chaque interaction y apparaissant. Plus ce nombre est élevé, plus l'entité est encline à exécuter l'interaction y étant associée. Cette étape aboutit à une matrice telle que celle de la figure 3.13 page 92.

De cette représentation graphique, on déduit le modèle de la matrice d'interaction raffinée. En effet, si $\mathcal{S} \in \mathbb{F}$, $\mathcal{T} \in \mathbb{F}$, $r \in \mathbb{R}^+$, $q \in \mathbb{Z}$ et $\mathcal{I} \in \mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)}$, alors :

- si l'élément $(\mathcal{I}, d = r, p = q)$ apparaît à l'intersection de la ligne associée à $id(\mathcal{S})$ et de la colonne associée à $id(\mathcal{T})$, alors on sait qu'il existe $a \in \mathcal{M}(\mathcal{S}, \mathcal{T})$ tel que $a = (\mathcal{I}, r, \mathcal{S}, \mathcal{T})$, et que $\mathcal{M}_{raff}(a) = q$;
- si l'élément $(\mathcal{I}, p = q)$ apparaît à l'intersection de la ligne associée à $id(\mathcal{S})$ et de la colonne associée à $\emptyset$, alors on sait qu'il existe $a \in \mathcal{M}(\mathcal{S}, \emptyset)$ tel que $a = (\mathcal{I}, \mathcal{S})$, et que $\mathcal{M}_{raff}(a) = q$.

Matrice de mise à jour (« H »). Le dernier élément pouvant être spécifié en début de cette seconde partie de la méthodologie consiste à établir, au travers d'une matrice de mise à jour, les interactions dégénérées étant exécutées par une entité afin de mettre à jour son état. La première étape de cette spécification consiste à construire une matrice de mise à jour vide, à l'aide des identifiants de familles d'entités identifiées lors de l'étape « B » : pour chaque famille d'entités présente dans $\mathbb{F}$, une colonne dont le titre est l'identifiant de la famille est ajoutée dans la matrice. Cela revient à créer une matrice de mise à jour vide : $\forall \mathcal{S} \in \mathbb{F}, \mathcal{U}(\mathcal{S}) = \emptyset$.

Dans un second temps, la matrice de mise à jour est remplie de la même manière que la colonne dédiée aux assignations dégénérées dans la matrice d'interaction brute, en plaçant l'identifiant d'interactions dans les cases de la matrice (identifiées lors de l'étape « C »). Cette étape aboutit à une matrice dont l'apparence est illustrée sur la figure 4.7.

	Végétal	Herbivore	Carnivore
Interactions de mise à jour		VIEILLIR AUGMENTERSENSATIONFAIM	VIEILLIR AUGMENTERSENSATIONFAIM

FIGURE 4.7 – Matrice de mise à jour d'une simulation de type proie/prédateurs, où des `Herbivores` côtoient des `Carnivores` ainsi que des `Végétaux`. Cette matrice spécifie que la SENSATION DE FAIM AUGMENTE progressivement au fil du temps, même lorsque les entités ne participent pas à une interaction. Elle spécifie de plus que les `Herbivores` et les `Carnivores` peuvent VIEILLIR.

Il est possible de déduire de cette représentation graphique le modèle de la matrice de mise à jour. En effet, avec $\mathcal{S} \in \mathbb{F}$ et $\mathcal{I} \in \mathbb{I}_{(1,0)}$, si l'élément $(id(\mathcal{I}))$ apparaît dans la cellule associée à $id(\mathcal{S})$, alors on peut ajouter l'élément $(\mathcal{I}, \mathcal{S})$ à $\mathcal{U}(\mathcal{S})$.

Tout comme pour la matrice d'interaction brute, l'ajout d'interactions dans cette matrice n'est possible que si leur cardinalité est $(0, 1)$. Réciproquement, une interaction présente dans cette matrice ne peut avoir qu'un seul argument lors de sa spécification dans l'étape « D ».

Matrice de mise à jour ordonnée (« J »). Puisque l'ordre d'exécution de deux interactions de mise à jour peut changer de manière drastique les résultats de simulation obtenus, la matrice de mise à jour spécifiée lors de l'étape « H » doit être complétée, afin de déterminer dans quel ordre ces dernières sont exécutées. Cet ordre est défini par un nombre entier, *i.e.* une priorité, qui est associée à chaque interaction figurant dans la matrice de mise à jour. Cette étape aboutit à une matrice de mise à jour ordonnée, dont l'apparence est illustrée sur la figure 4.8.

	Végétal	Herbivore	Carnivore
Interactions de mise à jour		(Vieillir, p=2) (AugmenterSensationFaim, p=1)	(Vieillir, p=2) (AugmenterSensationFaim, p=1)

FIGURE 4.8 – Matrice de mise à jour ordonnée de la simulation de type proie/prédateurs présentée dans la figure 4.7. Cette matrice spécifie que l'interaction de mise à jour faisant vieillir les entités est exécutée avant l'interaction de mise à jour faisant augmenter leur sensation de faim.

Il est possible de déduire de cette représentation graphique le modèle de la matrice de mise à jour ordonnée. En effet, en posant $\mathcal{S} \in \mathbb{F}$, $\mathcal{I} \in \mathbb{I}_{(1,0)}$ et $q \in \mathbb{Z}$, on sait que si l'élément $(id(\mathcal{I}), p = q)$ apparaît dans la cellule associée à $id(\mathcal{S})$, alors il existe un élément $u \in \mathcal{U}(\mathcal{S})$ tel que $u = (\mathcal{I}, \mathcal{S})$ et que $\mathcal{U}_{ord}(u) = q$.

4.1.4 Spécification du comportement précis des entités et de l'environnement.

Cette dernière partie de la méthodologie IODA spécifie les aspects les plus microscopiques de la simulation. Elle consiste à rendre explicite la description des différentes primitives identifiées lors des étapes précédentes.

Primitives de l'environnement (« I »). Différentes étapes de la méthodologie ont abouti à l'identification de primitives de l'environnement. Parmi ces primitives figurent celle permettant le calcul de la distance séparant deux entités, identifiée lors de l'étape de spécification de la topologie de l'environnement « A », ainsi que des primitives identifiées lors de la description des préconditions, des actions et du déclencheur des interactions « D », et enfin des primitives identifiées lors de la description du halo des entités « F ». À ce stade, on a donc :

$$primitives_{env} = \left\{ p \;\middle|\; \begin{array}{l} \exists \mathcal{I} \in \mathbb{I}_{(1,0)} | p \in sign_{env}(\mathcal{I}) \vee \\ \exists \mathcal{I} \in \mathbb{I}_{(1,1)} | p \in sign_{env}(\mathcal{I}) \vee \\ \exists \mathcal{F} \in \mathbb{F} | p \in sign_{env}(\mathcal{H}(\mathcal{F})) \vee \\ p = \langle distance, Nombre\ a\ Virgule\ Flottante, \{Entite, Entite\} \end{array} \right\}$$

A ces primitives s'ajoutent :
- les primitives permettant d'ajouter ou de retirer une entité à l'environnement ;
- la primitive permettant de connaître les entités situées dans l'environnement (*i.e.* permettant de connaître $\mathbb{E}$).

Puisque la forme de ces primitives dépend de la topologie de l'environnement, elles n'apparaissent pas dans la définition formelle précédente.

L'étape « I » de la méthodologie consiste à fournir une description à ces primitives, en associant un algorithme à chaque primitive de $primitives_{env}$. Le modèle de l'environnement est alors représenté graphiquement sous la forme de la figure 4.9, à laquelle est associée une signature d'entité, dont la représentation graphique a déjà été présentée.

<table>
<tr><td colspan="5">« environnement »
Identifiant de l'environnement</td></tr>
<tr><td>Signature des entités dans l'environnement</td><td colspan="4">identifiant de la signature des entités pour cet environnement</td></tr>
<tr><td rowspan="2">Identifiant</td><td rowspan="2">Valeur retournée</td><td colspan="2">Attribut</td><td rowspan="2">Description</td></tr>
<tr><td>Identifiant</td><td>Type</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
</table>

FIGURE 4.9 – Forme générale de la représentation graphique permettant la spécification de l'environnement.

Une primitive de l'environnement est décrite par un algorithme, pouvant manipuler ses arguments, ainsi que des primitives de l'environnement, des attributs de l'environnement, et éventuellement des primitives abstraites des entités présentes dans ses arguments. La syntaxe de cet algorithme est identique à celle des algorithmes permettant de décrire déclencheur, préconditions et actions d'une interaction. Il en va de même des implications de la présence d'une primitive de l'environnement, d'une primitive d'action ou de perception : pour chaque primitive de l'environnement apparaissant dans cet algorithme, une nouvelle primitive de l'environnement à spécifier est ajoutée à l'ensemble $primitives_{env}$, si elle n'y figure pas déjà. De même, si une primitive d'action y apparaît, alors toutes les familles d'entités de la simulation doivent l'implémenter.

Exemple. Calcul de distance
Le calcul de la distance distance séparant deux entités dans un environnement euclidien à deux dimensions

et non torique peut prendre la forme de la primitive de l'environnement qui suit :

distance(e_1,e_2) *:*
$produitScalaire \Leftarrow \Big(environnement.abscisse(e_1) - environnement.abscisse(e_2)\Big)^2$
$produitScalaire \Leftarrow produitScalaire + \Big(environnement.ordonnee(e_1) - environnement.ordonnee(e_2)\Big)^2$
retourner $\sqrt{produitScalaire}$

Cette primitive fait mention de deux primitives de l'environnement, qu'il faut par la suite définir :
- *la primitive* $\langle abscisse, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle$*, qui permet de connaître l'abscisse d'une entité dans l'environnement ;*
- *la primitive* $\langle ordonnee, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle$*, qui permet de connaître l'ordonnée d'une entité dans l'environnement.*

Primitives d'action et de perception (« L »). Afin de garantir un certain polymorphisme au sein des interactions, mais aussi de la description du halo des entités, ou au sein des primitives de l'environnement, leurs algorithmes de description manipulent des primitives abstraites. Afin de terminer la spécification du modèle, chaque famille d'entités doit associer un algorithme à l'ensemble de leurs primitives abstraites, déterminé par :
- les interactions présentes dans la matrice d'interaction raffinée, conjointement à la signature de cette famille d'entités dans ces dernières. Si l'interaction $\mathcal{I}$ est présente dans la ligne (respectivement la colonne) associée à une famille d'entités $\mathcal{F} \in \mathbb{F}$, alors cette famille d'entités doit fournir la description de toutes les primitives abstraites présentes dans la signature $sign_{source}(\mathcal{I})$ (respectivement $sign_{cible}(\mathcal{I})$) de cette interaction ;
- les interactions présentes dans la matrice de mise à jour ordonnée, conjointement à la signature de cette famille d'entités dans ces dernières. Si l'interaction $\mathcal{I}$ est présente dans la cellule associée à une famille d'entités $\mathcal{F} \in \mathbb{F}$, alors cette famille d'entités doit fournir la description de toutes les primitives abstraites présentes dans la signature $sign_{source}(\mathcal{I})$ de cette interaction ;
- le halo de la famille d'entités. Une famille d'entités $\mathcal{F} \in \mathbb{F}$ doit implémenter toutes les primitives abstraites présentes dans $sign_{entite}(\mathcal{H}(\mathcal{F}))$;
- les primitives de l'environnement : une famille d'entités doit implémenter toutes les primitives abstraites apparaissant dans l'algorithme décrivant chaque primitive de l'environnement.

À ce stade, on a donc :

$$\forall \mathcal{F} \in \mathbb{F}, primitives(\mathcal{F}) = \left\{ p \;\middle|\; \begin{array}{l} \exists a \in \mathcal{M}(\mathcal{F}, \emptyset) | p \in primitives\Big(sign_{source}\big(\mathcal{I}(a)\big)\Big) \vee \\ \exists \mathcal{T} \in \mathbb{F}, \exists a \in \mathcal{M}(\mathcal{F}, \mathcal{T}) | p \in primitives\Big(sign_{source}\big(\mathcal{I}(a)\big)\Big) \vee \\ \exists \mathcal{S} \in \mathbb{F}, \exists a \in \mathcal{M}(\mathcal{S}, \mathcal{F}) | p \in primitives\Big(sign_{cible}\big(\mathcal{I}(a)\big)\Big) \vee \\ \exists u \in \mathcal{U}(\mathcal{F}) | p \in primitives\Big(sign_{source}\big(\mathcal{I}(u)\big)\Big) \vee \\ p \in sign_{entite}\big(\mathcal{H}(\mathcal{F})\big) \vee \\ p \in sign_{entites}(environnement) \end{array} \right\}$$

La représentation graphique donnée à une famille d'entités correspond alors à la représentation générale décrite dans la figure 4.10.

La seule convention imposée dans la description de telles primitives est de faire apparaître les attributs de la famille d'entités de manière explicite, en préfixant l'identifiant de l'attribut par l'expression clé « this. ».

Exemple. Description d'une primitive « acuiteVisuelle »
L'algorithme qui suit décrit la primitive de perception $\langle acuiteVisuelle, Nombre\ a\ Virgule\ Flottante, \emptyset\rangle$ *pour une famille d'entité percevant les entités situées à une distance inférieure ou égale à leur acuité visuelle.*

acuiteVisuelle() *:*
retourner this.*acuité*

Cette primitive manipule un attribut dont l'identifiant est « acuité ».

« famille d'entités » Identifiant de la famille d'entités				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
halo	Booléen	e_1 e_2	Entité Entité	*algorithme décrit lors de l'étape « F »*
...	...	...	...	...

FIGURE 4.10 – Forme générale de la représentation graphique permettant la spécification de l'environnement.

Exemple. Description d'une primitive « acuiteVisuelle »
L'algorithme proposé ici fournit une autre description de la primitive de perception $\langle acuiteVisuelle, Nombre\ a\ Virgule\ Flottante, \emptyset \rangle$*, pour une famille d'entité percevant les entités situées à une distance inférieure ou égale à leur acuité visuelle divisée. De plus, les entités de cette famille considérées comme myopes voient leur acuité visuelle divisée par deux.*

acuiteVisuelle() *:*
| **Si this.***myope* **alors :**
| | **retourner this.***acuité*/2
| **Sinon :**
| | **retourner this.***acuité*
| **FinSi**

Cette primitive manipule deux attributs dont les identifiants sont « acuité » et « myope ».

Attributs des familles d'entités (« K ») et de l'environnement (« M »). Les deux dernières étapes de la méthodologie IODA consistent à terminer la spécification des familles d'entités et de l'environnement, en identifiant leurs attributs. L'identification de ces attributs se basant sur l'analyse des spécifications des diverses primitives de perception, d'action, ou des primitives de l'environnement décrites dans les étapes précédentes.

4.2 Questions issues de la pratique d'une méthodologie de conception

Notre pratique de la méthodologie IODA nous a amenés à retrouver quatre questions récurrentes survenant dans tout type de méthodologie de conception :
- Toute entité est-elle un agent ?
- Une méthodologie de conception doit-elle être dogmatique ?
- Faut-il prendre en compte les performances lors de la modélisation ?
- Faut-il tout décrire au sein d'une règle (dans notre cas, une interaction) ?

Cette section n'a pas pour ambition de résoudre ces problèmes, mais de décrire succinctement le positionnement de notre approche vis à vis de ces problèmes.

4.2.1 Toute entité est-elle un agent ?

La modélisation des agents, et des entités échappant parfois à la définition de « agent », est un point récurrent en simulation informatique, et constitue un problème non trivial à résoudre.

Dans le cas des approches telles que Madkit [GFM01], Netlogo [WC99], RePast [NTCO07] ou Swarm [Ter98], chaque entité est modélisée par un élément logiciel appelé « agent », dont le compor-

tement est spécifié de manière libre en utilisant un langage de programmation. La conception des agent et de leur comportement n'est que très peu aidée. Les concepteurs doivent y définir leur propre représentation des connaissances et/ou implémenter directement le comportement. Elles ne facilitent donc pas la conception de simulations contenant une grande variété d'entités et d'interactions.

D'autres approches telles que Soar [LNR87], Act-R [ABB+04], Jack [BHRH00], Jason [BH06], Maleva [BM07], InteRRaP [FMP95] ou Touring Machines [Fer92] fournissent une représentation explicite aux connaissances des agents. Toutefois, elles se focalisent uniquement sur les agents. Par conséquent, les entités non-proactives sont modélisées comme des agents et gérées comme tels dans l'algorithme de simulation, impliquant de très probables pertes de performances.

Les approches telles que le méta-modèle Agents & Artifacts (A& A) [ORV08], ou la plateforme SeSam [KHF06] fournissent à la fois une représentation explicite aux connaissances des agents et identifient clairement comment spécifier les entités n'étant pas des agents. Pour ce faire, elles différencient agent et autres entités à l'aide de types. Par exemple, dans SeSam, il y a distinction entre le type « agent » et le type « ressource ». De manière similaire, il y a distinction entre « Agent » et « Artefact » dans le méta-modèle Agents & Artifacts. Chaque type définit une méthodologie particulière spécifique pour définir une entité, une façon spécifique de représenter ses connaissances et des algorithmes dédiés permettant de les implémenter.

Bien que l'attribution de types aide la conception de tout type d'entité dans la simulation, et permette d'optimiser les simulateurs obtenus, elle ajoute aussi des problèmes qui, contrairement à leur vocation initiale, rend la conception de simulation plus complexe. Pour s'en convaincre, nous illustrons dans cette section trois de ces problèmes. Puisque d'une approche à l'autre, les types utilisés peuvent avoir des sens très différents, nous introduisons dans un premier temps une terminologie qui permettra de les caractériser et les comparer.

Terminologie caractérisant l'activité des entités

Nous considérons que l'activité d'une entité dans une simulation multi-agents peut être caractérisée par la présence d'aucune, une, deux ou toutes les capacités suivantes [KMP10] :

- la capacité d'*agir sur les autres entités ou sur l'environnement*, qui caractérise ce que nous appelons **entités actives**. Par exemple un piéton qui se déplace sur une route, un globule blanc qui combat les infections, un acheteur émettant des ordres sur un marché, *etc.*
- la capacité de *subir les actions d'une autre entité, ou d'être utilisée comme un outil par une autre entité*, qui caractérise ce que nous appelons **entités passives** . Par exemple un interrupteur qui est allumé ou éteint par un utilisateur, un fournisseur d'informations dans un aéroport (aussi bien un réceptionniste qu'une borde d'information), ou encore une porte permettant à une personne de transiter d'une pièce à une autre, *etc.*
- la capacité de *changer leur propre état sans agir ou subir d'action*, qui caractérise ce que nous appelons **entités labiles**. Par exemple de la nourriture, dont les propriétés nutritives diminuent avec le temps, une entité dont la sensation de faim augmente avec le temps, *etc.*

Une entité peu exprimer une ou plusieurs de ces capacités. Par exemple, en éthologie, des animaux pouvant se reproduire sont au moins actives (puisqu'elles peuvent initier la reproduction) et passives (puisqu'elles peuvent subir la reproduction). Une entité peut aussi n'exprimer aucune de ces capacités. Par exemple, un rideau dont le seul rôle dans la simulation est de contraindre la perception des entités. Puisque ce cas est récurrent en simulation, nous complétons notre terminologie par une valuation particulière des capacités mentionnées ici, que nous appelons **entités permanentes**, qui représente les entités n'étant **ni actives, ni passives, ni labiles**. Cette terminologie est résumée dans le tableau 4.1.

La plupart des types utilisés en simulation multi-agents correspondent alors à une **valuation statique** de ces capacités. Par exemple, une entité dont le type est « artefact » dans [ORV08] correspond selon notre terminologie à une entité passive. Une entité dont le type est « ressource » dans [KHF06] correspond selon notre terminologie soit à une entité passive, soit à une entité permanente. En fin, notre terminologie permet d'identifier que le type « agent » n'a pas le même sens dans [ORV08] et [FM96] : d'après notre terminologie, un agent est une entité active et passive dans le premier cas et une entité active, passive et labile dans le second.

	Définition	Capacité correspondante	Exemple
Active	Qui manifeste de l'activité, de l'énergie, qui est capable d'agir	Agir sur les autres entités ou sur l'environnement	Un globule blanc combattant les infections
Passive	Qui subit l'action, par opposition à Actif	Subir les actions d'une autre entité, ou d'être utilisée comme un outil par une autre entité	Un interrupteur qui est allumé ou éteint par un utilisateur
Labile	Qui est peu stable ; qui est sujet à se transformer	Changer leur propre état sans agir ou subir d'action	Une entité dont la sensation de faim augmente avec le temps
Permanente	Établi de manière durable et continue, sans interruption ni modification	Ni active, ni passive, ni labile	Un rideau dont le seul rôle dans la simulation est de contraindre la perception des entités

TABLE 4.1 – Résumé des différentes capacités qu'une entité d'une simulation peut exprimer. Les définition fournies dans ce tableau proviennent du dictionnaire en ligne de l'académie française [fra10].

D'autres types sont utilisés pour différencier d'autres propriétés des entités. Par exemple, la seule différence existant entre les types « turtle » et « patch » dans Netlogo [WC99] est leur représentation dans l'environnement : dans le premier cas, il s'agit d'un point mobile et dans le second un carré de taille fixe et immobile. Dans de tels cas, l'usage de types peut être évité si la représentation des entités dans l'environnement est utilisé comme un attribut de l'environnement ou des entités.

Problèmes liés à l'utilisation de types

Dans les approches telles que le méta-modèle A& A, l'un des points cruciaux de la conception d'une simulation consiste à déterminer quel type utiliser pour modéliser une entité. En effet, en fonction de ce type, une structure de données spécifique est utilisée pour représenter l'entité. Elle est de plus spécifiée à l'aide d'une méthodologie dédiée.

Dans cette sous-sous-section, nous illustrons de trois points de vue différents pourquoi l'usage de types rend la conception de simulations moins intuitive et plus difficile. Puisque le sens de types tels que « agents » ou « artefact » est fortement dépendant de l'approche d'où ils ont été extraits, les différents types mentionnés jusqu'à présent sont identifiés par la suite à l'aide de la terminologie introduite dans la sous-sous-section précédente, afin d'éviter toute ambiguïté dans nos propos.

Conception incrémentale de simulations Supposons qu'une simulation mette en jeu des personnes qui se déplacent, ainsi que des murs et portes qui cachent les personnes cachées derrière elles. On peut supposer sans risques que les personnes sont modélisées par des entités au moins actives, puisqu'elles se déplacent. Les murs et les portes semblent n'influer que la perception de personnes. Par conséquent, ce sont des entités permanentes. Si l'on considère que les portes sont un moyen utilisé par une personne pour passer d'une pièce à une autre, elles ne sont plus de simples entités permanentes : elles deviennent un outil utilisé par les personnes et donc des entités passives.

Cette simulation peut être modélisée en deux phases :

1. déplacement des personnes et leur perception. Les personnes peuvent ainsi se mettre dans une situation les poussant à utiliser les portes ;
2. utilisation des portes comme outil de téléportation.

La porte passe ainsi d'entité permanente (utilisée uniquement pour la perception) lors de la première itération, à passive (un outil de téléportation) lors de la seconde.

D'une itération de conception à l'autre, le type d'une entité peut donc évoluer. Puisque ce type conditionne comment l'entité est modélisée et quelle structure de données utiliser, il sera nécessaire de la re-spécifier et la ré-implémenter presque complètement.

Cette situation survient également dans le cas où un modèle est révisé. Elle survient aussi dans des simulations large échelle, *i.e.* de simulations contenant une grande variété d'entités pouvant effectuer et subir des actions diversifiées. En effet, pour illustrer de manière intuitive le problème de la conception incrémentale, un problème de taille réduite a été utilisé, ne nécessitant la modélisation que de peu d'entités et d'interactions. Cela a permis de trouver le type des différentes entités dès le départ. Dans des problèmes réels de simulation, le nombre d'entités et d'actions peut être bien plus grand, il n'est pas possible de

connaître *a priori* toute les fonctionnalités de la simulation. Par conséquent, de telles simulations sont construites par étapes, par une succession de modèles de complexité croissante. Chaque étape introduit de nouvelles informations dans le modèle, et rend la révision du type de certaines entités inévitable.

Types et concepts naturels L'argument le plus couramment utilisé pour justifier l'utilisation du paradigme agent pour modéliser des simulations repose sur l'ontologie qu'il propose, considérée comme proche de celle des phénomènes simulés [Edm05]. Dans ce contexte, les entités actives telles que les « agents » sont souvent interprétées comme « entité animée », « entité mobile » ou « entité vivante » puisque les entités actives sont supposées percevoir et décider par elles-mêmes des actions qu'elles initient. La notion de type entre en contradiction avec un tel argument.

Considérons par exemple une extension de la simulation présentée plus haut, dans laquelle un mur ait la possibilité de s'écrouler sur une personne. Le mur est capable d'effectuer une action et devient par conséquent une entité active. Bien qu'un mur soit interprété naturellement comme une « entité inanimée », il peut avoir la capacité d'initier des actions et peut donc être une entité active. Il en va de même pour les portes automatiques, les contrôleurs de vitesses sur une route, *etc.*

La distinction entre entités actives, passives, labiles et permanentes est opérationnelle. De tels concepts ne s'associent pas intuitivement avec des concepts naturels tels que « entité animée », « entité mobile » ou « entité vivante ». Ce problème peut sembler insignifiant du point de vue d'un expert en informatique, qui approchent la conception d'une simulation d'un point de vue opérationnel. Toutefois, la simulation est un outil au service d'experts du domaine simulé (*i.e.* biologistes, sociologues, *etc.*) qui ne partagent pas systématiquement ce point de vue. Ainsi, le choix de la notion à utiliser pour spécifier une entité est, dans certains cas, contre-intuitif et fait défaut à la conservation d'une ontologie proche des phénomènes simulés, qui est pourtant l'un des principaux intérêts de l'utilisation de systèmes multi-agents dans le cas de simulations.

Entités hybrides L'utilisation de types impose une division statique de l'espace des représentations possibles pour les entités. Dans ce contexte, la conception d'une entité hybride, *i.e.* devant exprimer les capacités exprimées dans deux types différents, peut poser problème.

Considérons par exemple une simulation faisant intervenir des employés de magasin pouvant endosser deux rôles différents :

- fournir des information aux clients lorsqu'ils sont interpellés (l'entité est alors un *informateur*) ;
- ordonner des articles sur des étagères (l'entité est alors un *trieur*) ;

Un informateur ne fait que répondre aux requêtes des clients et est donc une entité passive. Un trieur est capable d'agir dans l'environnement et est par conséquent une entité active. Un employé peut aussi être à la fois un trieur et un informateur. Dans ce cas, il est à la fois une entité passive et une entité active.

Se pose alors une question : si un trieur est représenté à l'aide d'un type T_1 et un informateur à l'aide d'un type T_2, qu'en est il d'un employé qui est à la fois trieur et informateur ? Différentes solutions existent à ce problème, en fonction de la combinaison des capacités de notre terminologie exprimées par un type (voir figure 4.11).

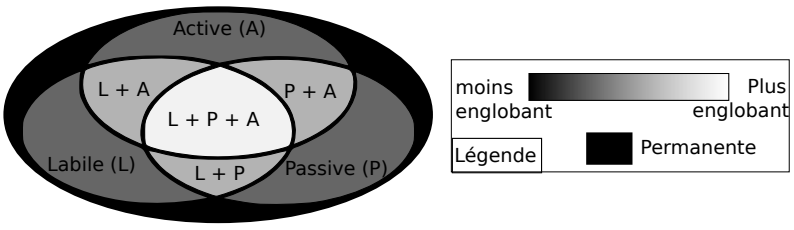


FIGURE 4.11 – Caractérisation des différentes combinaisons de compétences décrivant l'activité d'une entité d'une simulation, selon la terminologie décrite dans la section 4.2.1.

Dans le cas idéal reposant sur les types, chaque combinaison possible de capacités est modélisé par un type. Il est alors possible de représenter toute entité de la simulation. À notre connaissance, aucune approche ne permet ceci.

Un cas intermédiaire survient lorsqu'une entité est caractérisée par une combinaison de capacités qu'aucun type n'exprime directement, mais qu'il existe un type représentant un sur-ensemble de la combinaison recherchée. Dans ce cas, le type exprimant la plus petite combinaison de capacité englobantes est utilisé pour modéliser l'entité, réduisant ainsi les performances du simulateur. Par exemple, une entité uniquement labile peut être modélisée à l'aide d'un type représentant des entités labiles et actives, où l'activité de l'entité est nulle. Ce cas se retrouve par exemple dans la plateforme SeSam (voir figure 4.12a), où les entités permanentes ou passives sont représentées à l'aide du type « ressource » et où les entités exprimant toute autre combinaison de capacités sont représentées avec le type « agent ».

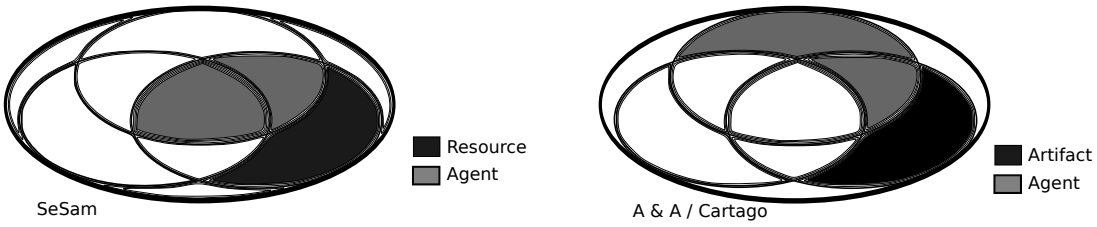


FIGURE 4.12 – Caractérisation des différentes combinaisons de compétences d'une entité exprimées par les types de la plateforme SeSam et du méta-modèle A&A. Elle se base sur la représentation utilisée sur la figure 4.11.

Dans le pire des cas, certaines combinaisons de capacités ne peuvent pas être exprimées. C'est par exemple le cas du méta-modèle A&A, où les entités labiles ne peuvent être spécifiées (voir figure 4.12b).

Ainsi, l'utilisation de types peut, dans certains cas, restreindre l'ensemble des simulations qu'il est possible de modéliser.

Modélisation des entités et agents dans IODA

L'approche IODA fournit une représentation unifiée des entités, basée sur une identification dynamique et transparente des différentes capacités de la terminologie résumée sur le tableau 4.1. En effet, dans notre méthodologie, **ces capacités sont déduites de la structure de la matrice d'interaction brute** et de la matrice de mise à jour :

- Une entité e est dite *active* si la ligne associée à sa famille d'entités dans la matrice d'interaction brute contient au moins un élément d'assignation, *i.e.* $\exists \mathcal{T} \in \mathbb{F} \cup \{\emptyset\} | \mathcal{M}\Big(famille(e), \mathcal{T}\Big) \neq \emptyset$;
- Une entité e est dite *passive* si la colonne associée à sa famille d'entités dans la matrice d'interaction brute contient au moins un élément d'assignation, *i.e.* $\exists \mathcal{S} \in \mathbb{F} | \mathcal{M}\Big(\mathcal{S}, famille(e)\Big) \neq \emptyset$;
- Une entité e est dite *labile* si la cellule associée à sa famille d'entités dans la matrice de mise à jour contient au moins un élément d'assignation, *i.e.* $\mathcal{U}\Big(famille(e)\Big) \neq \emptyset$.

Par conséquent, la conception incrémentale d'une simulation est mieux supportée qu'avec des types. En effet, une entité peut passer de non-active à active, de non-passive à passive, ou de non-labile à labile de manière complètement transparente, par la simple modification de ce que l'entité est capable de faire, sans que cela implique une re-spécification complète ou partielle de cette entité. De plus, tout type d'entité hybride peut être modélisé, puisque toute combinaison caractérisant l'activité d'une entité peut être exprimée dans notre modèle (voir figure 4.13). Enfin, la notion de type n'existe pas dans notre approche. Le fait qu'une entité est labile, active, passive ou permanente est déduit directement du contenu du modèle et ne doit donc pas être spécifié explicitement par le concepteur. Par conséquent, il n'y a plus de confusion possible entre l'ontologie induite par les types et l'ontologie provenant du phénomène modélisé : chaque entité apparaissant physiquement dans le phénomène est représentée par une entité dans IODA.

L'approche IODA fournit donc une représentation unifiée des entités de la simulation. Puisque cette approche se place dans le cadre de la simulation multi-agents, nous appelons « agent » cette représentation.

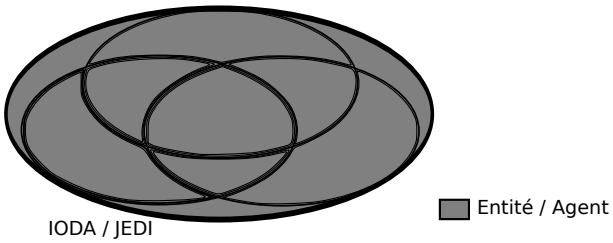
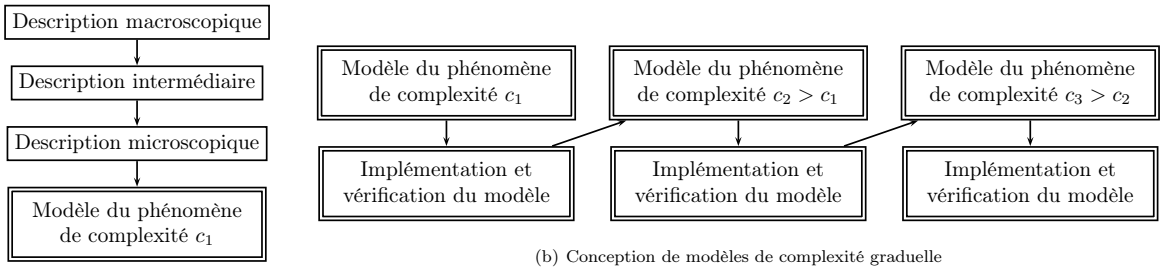


FIGURE 4.13 – Caractérisation des différentes combinaisons de compétences d'une entité pouvant être exprimées avec l'approche IODA. Elle se base sur la représentation utilisée sur la figure 4.11.

4.2.2 Une méthodologie de conception doit-elle être dogmatique ?

L'objectif d'une méthodologie de conception est de fournir un cadre formel facilitant la description un modèle. Cette simplification se fait au travers de deux traits de la méthodologie : la possibilité de construire un modèle en y introduisant de manière progressive des connaissance (en passant de descriptions macroscopiques à des descriptions microscopiques), et la possibilité d'atteindre le modèle souhaité en construisant des modèles de complexité croissante (par exemple un modèle dans lequel les entités peuvent se déplacer, puis un modèle dans lequel les entités peuvent interagir de manière plus diversifiée). Ces deux traits sont schématisés sur la figure 4.14.



(a) Conception graduelle d'un modèle

(b) Conception de modèles de complexité graduelle

FIGURE 4.14 – Cheminements pouvant être employés par une méthodologie de conception de simulation afin de concevoir le modèle d'un phénomène.

Ces schémas permettent d'illustrer les principes généraux simplifiant la conception de modèles complexes, *i.e.* de modèles contenant un grand nombre de familles d'entités interagissant de manière variée, mais ne constituent en aucun cas un dogme qu'il est nécessaire de respecter tel quel. En effet, la construction d'un modèle est un procédé itératif, lors duquel certaines descriptions microscopiques peuvent amener à réviser les descriptions macroscopiques, qui se révélaient incomplètes. Illustrons ce propos à l'aide de la méthodologie IODA.

Reprenons par exemple le cas d'une simulation d'un écosystème, dans lequel des proies sont mangées par des prédateurs. Une première étude d'une telle simulation aboutit à une matrice d'interaction raffinée, ainsi qu'à une matrice de mise à jour ordonnée décrites dans la figure 4.15. Ces matrices spécifient que le comportement des entités consiste en priorité à mourir si jamais elle le doivent, sinon à se reproduire ou à manger d'autres entités, et à se déplacer par défaut si aucune autre interaction ne pouvait être initiée. La description du déclencheur de l'interaction MANGER (voir figure 4.4) mène à l'identification d'une primitive dont l'identifiant est « a faim », ainsi qu'une autre primitive dont l'identifiant est « diminuer sensation faim ». Ces primitives impliquent implicitement que la sensation de faim d'une entité augmente avec le temps indépendamment du comportement des entités. En conséquence, la matrice de mise à jour est révisée, pour devenir celle décrite dans la figure 4.16. Lors la description de la primitive de perception « a faim » de la famille d'entités `Prédateur` (voir figure 4.17), il est spécifié que la sensation de faim est corrélée à l'âge d'une entité : plus une entité est vieille, moins elle ressent la sensation de faim. Puisque l'âge intervient dans le calcul de la sensation de faim, il se révèle nécessaire de le mettre à jour via une

Source \ Cible	∅	Proie	Prédateur
Proie	(Se Déplacer, p=0) (Mourir, p=6)	(Se Reproduire, d=0cm, p=3)	
Prédateur	(Se Déplacer, p=0) (Mourir, p=6)	(Manger, d=15cm, p=3)	(Se Reproduire, d=0cm, p=3)

(a) Matrice d'interaction raffinée

	Proie	Prédateur
Interactions de mise à jour		

(b) Matrice de mise à jour ordonnée

FIGURE 4.15 – Exemple d'une matrice d'interaction raffinée, ainsi que d'une matrice de mise à jour d'une simulation de type proie/prédateur, lors d'une première phase de la méthodologie IODA.

Source \ Cible	∅	Proie	Prédateur
Proie	(Se Déplacer, p=0) (Mourir, p=6)	(Se Reproduire, d=0cm, p=3)	
Prédateur	(Se Déplacer, p=0) (Mourir, p=6)	(Manger, d=15cm, p=3)	(Se Reproduire, d=0cm, p=3)

(a) Matrice d'interaction raffinée

	Proie	Prédateur
Interactions de mise à jour		(Augmenter sensation faim, p=1)

(b) Matrice de mise à jour ordonnée

FIGURE 4.16 – Exemple d'une matrice d'interaction raffinée, ainsi que d'une matrice de mise à jour d'une simulation de type proie/prédateur, lors d'une seconde phase de la méthodologie IODA.

« famille d'entités » `Predateur`				
Identifiant	Valeur retournée	Attribut		Description
		Identifiant	Type	
halo	Booléen	e_1 e_2	Entité Entité	...
aFaim	Booléen	—	—	Retourner **this.***energie*× **this.***age* < **this.***seuilFaim*.
...	...	...	...	...

FIGURE 4.17 – Exemple de spécification de l'interaction MANGER dans le cas d'une simulation de type proie/prédateur.

interaction de mise à jour. Il faut donc de réviser la matrice de mise à jour ordonnée telle que présentée jusqu'à présent (voir figure 4.18).

Source \ Cible	∅	Proie	Prédateur
Proie	(Se Déplacer, p=0) (Mourir, p=6)	(Se Reproduire, d=0cm, p=3)	
Prédateur	(Se Déplacer, p=0) (Mourir, p=6)	(Manger, d=15cm, p=3)	(Se Reproduire, d=0cm, p=3)

(a) Matrice d'interaction raffinée

	Proie	Prédateur
Interactions de mise à jour		(Augmenter sensation faim, p=1) (Vieillir, p=2)

(b) Matrice de mise à jour ordonnée

FIGURE 4.18 – Exemple d'une matrice d'interaction raffinée, ainsi que d'une matrice de mise à jour d'une simulation de type proie/prédateur, lors d'une troisième phase de la méthodologie IODA.

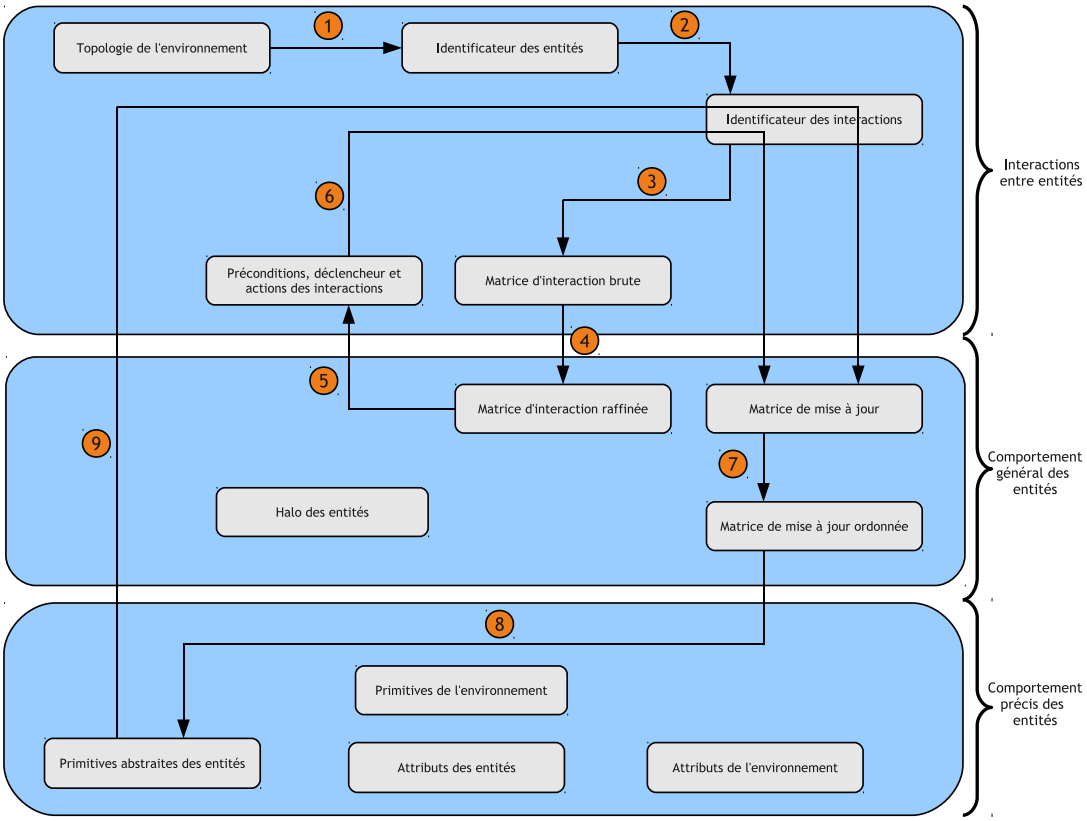
Le cheminement décrit ici transite entre aspects microscopiques et microscopiques de la spécification selon le schéma de la figure 4.19. On y constate que le schéma suivi n'est pas aussi strict que celui présenté dans la figure 4.19, mais il en respecte toutefois les principes fondamentaux : une spécification au niveau intermédiaire fait suite à une spécification au niveau macroscopique, et une spécification au niveau microscopique fait suite à une description au niveau intermédiaire. Dès lors, il est difficile d'établir un ordre strict entre les différentes étapes du processus de conception d'un modèle. Nous avons donc choisi de décrire le processus de conception d'un modèle uniquement par les dépendances existant entre les différentes étapes de la méthodologie IODA (résumées par la figure 4.1). La forme de la méthodologie IODA ayant été décrite dans la première section de ce chapitre est l'aboutissement de différents travaux cherchant à appréhender au mieux le problème décrit dans cette section. Elle fait suite à différentes descriptions imposant un ordre aux différentes étapes de la simulation [KMP08a, KMP08c], qui se sont révélées ne pas appréhender de manière satisfaisante le présent problème.

4.2.3 Faut-il prendre en compte les problèmes de performances lors de la conception du modèle d'un phénomène ?

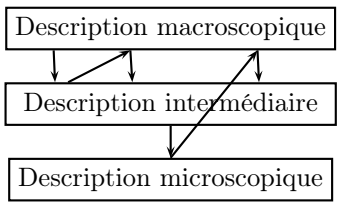
La viabilité des hypothèses émises sur le phénomène simulé ne se mesure pas par les résultats d'une seule expérimentation, mais par les résultats moyens obtenus par plusieurs expériences. Plus le nombre de ces expériences est élevé, plus résultats sont viables. Par conséquent, bien qu'annexe, l'optimisation du simulateur utilisé pour réaliser des expérience reste cruciale lors de la conception de simulations. La façon de structurer un modèle influe grandement sur les performances du simulateur produit. S'en suit un paradoxe : le modèle doit pouvoir être spécifié au maximum par des non-informaticiens, ce qui implique la quasi-absence d'optimisations dans la structure du modèle, mais doit aussi pouvoir rendre efficace le simulateur sous-jacent, sans quoi la simulation risque de ne pas pouvoir être exécutée, faute d'obtention de résultats suffisamment rapide. Ce problème a amené [MEU04] à ajouter un modèle transitoire, appelé modèle opérationnel, afin de prendre en compte ces optimisations sans pour autant altérer la nature du modèle spécifié par les experts du domaine.

Dans notre approche, cette distinction n'est pas faite. Cela ne rend pas pour autant impossible la spécification du modèle par des experts du domaine. Nous illustrons ceci sur un exemple : la spécification de la notion de distance entre deux entités.

Considérons la spécification suivante, décrivant comment la distance entre deux entités est calculée dans un environnement euclidien à deux dimensions non torique, où les entités sont assimilées à des points :



(a) Cheminement précis dans la méthodologie Ioda



(b) Cheminement équivalent dans le schéma issu de la figure 4.14

FIGURE 4.19 – Cheminement utilisé pour décrire une partie du modèle de type proie/prédateur illustré dans la section 4.2.2.

distance(e_1,e_2) :
$produitScalaire \Leftarrow \Big(environnement.abscisse(e_1) - environnement.abscisse(e_2)\Big)^2$
$produitScalaire \Leftarrow produitScalaire + \Big(environnement.ordonnee(e_1) - environnement.ordonnee(e_2)\Big)^2$
retourner $\sqrt{produitScalaire}$

Il est intéressant de remarquer que, dans l'exemple précédent, les coordonnées d'une entité sont connues à l'aide des primitives de l'environnement $\langle abscisse, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle$ et $\langle ordonnee, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle$, pouvant être implémentées de plusieurs manières. Une première façon serait de considérer que l'environnement contienne une structure de données associant à chaque entité ses coordonnées :

abscisse(e) :
retourner **this.**$abscisses[e]$

Une autre façon serait de considérer que ces primitives fassent appel à une primitive abstraite des entités, permettant de connaître leurs coordonnées :

abscisse(e) :
retourner e**.**$abscisse()$

Dans le premier cas, chaque lecture ou modification d'une coordonnée implique le parcours d'une structure de données plus ou moins efficace, qui nécessite une quantité de calculs corrélée au nombre d'entités dans la simulation. Plus ce nombre est élevé, moins ce calcul est efficace. Dans le second cas, l'utilisation d'une structure de données n'est plus nécessaire, optimisant d'autant le calcul de la distance entre deux entités.

Dans la méthodologie IODA, les optimisations ont lieu en pratique lors de la spécification des primitives abstraites des entités ou de l'environnement. Elle a donc lieu à la fin du processus de conception d'un modèle. Toutes les étapes préalables peuvent être faites sans prêter attention aux optimisations et peuvent donc être spécifiées par des experts du domaine.

4.2.4 Faut-il tout décrire au sein d'une règle/interaction ?

Dans IODA, le recensement des identificateurs des interactions, la description de leurs préconditions, actions et déclencheur, ou la description des primitives de l'environnement et des primitives abstraites amène à se questionner sur le contenu de chacun de ces éléments. En effet, un même comportement peut être décrit de manière très différentes au sein d'un modèle. Il se pose en la question de quand utiliser des primitives abstraites, et quand écrire directement une spécification précise, ne permettant pas de polymorphisme.

Par exemple, la description du déclencheur d'une interaction manger peut aller de très abstrait et générique (voir figure 4.20a), et donc réutilisable dans un maximum de cas en lui faisant perdre de son expressivité, à très spécifique et précis (voir figure 4.20b), et donc très expressive mais peu réutilisable, en passant par des descriptions intermédiaires (voir figure 4.20c), qui fournit un compromis entre les deux cas précédemment mentionnés.

declencheur(e_1,e_2) :
retourner e_1**.**$veutManger(e_2)$

(a) Spécification très abstraite

declencheur(e_1,e_2) :
retourner e_1**.**$get\ energie()$< e_1**.**$seuil\ de\ faim()\wedge$
e_2**.**$valeur\ energetique()$> e_1**.**$seuil\ d\ interet()$

(b) Spécification très spécifique

declencheur(e_1,e_2) :
retourner e_1**.**$a\ faim()\wedge e_1$**.**$faim\ significativement\ reduite\ par(e_2)$

(c) Spécification intermédiaire

FIGURE 4.20 – Différentes façons de spécifier le déclencheur d'une interaction MANGER, dans le contexte d'une simulation en éthologie, cherchant à modéliser un écosystème.

Ce problème n'est toutefois pas restreint à la méthodologie IODA. Il se pose aussi dans des approches telles que :

- l'architecture de subsomption [Bro86] lorsqu'il faut déterminer si une partie du comportement d'un robot doit être décrit au sein d'un module, ou donner lieu à la création d'un second module ;
- les architectures à planification réactive telles que Jack [BHRH00] lorsqu'il faut déterminer s'il faut décrire une partie du comportement d'un agent au sein d'un seul plan ou me décomposer en des sous-plans.

Le choix de la façon de spécifier une interaction, une primitive de l'environnement ou une primitive abstraite est un problème difficile, qui dépend principalement du compromis recherché par le concepteur. En effet, la délégation d'une partie de la spécification d'une interaction à des primitives abstraites rend l'interprétation directe des interactions moins aisée. Elle permet toutefois d'obtenir une grande diversité comportementale en un nombre réduit de modifications du modèle, tout en favorisant son utilisation dans des contextes différents. L'approche IODA offre la possibilité de personnaliser ces éléments finement, à l'aide des différents niveaux d'abstraction qu'elle propose.

4.3 Simulation d'un modèle centré sur les interactions

La construction d'un modèle n'est qu'une des deux étapes de la construction d'une simulation à l'aide d'une approche transversale. La seconde consiste à définir par quel procédé il est possible de passer d'un modèle à une implémentation.

Dans l'approche IODA, **nous soutenons que l'implémentation doit conserver la même structure que le modèle, ainsi que sa terminologie**. Sous réserve que cette structure soit conservée, il est possible de décrire des algorithmes tirant parti des informations contenues dans le modèle pour produire automatiquement une simulation, en conservant un morphisme presque total entre éléments du modèle et éléments de l'implémentation. Bien entendu, une implémentation ne préservant pas ce morphisme est possible, mais réduit significativement l'intérêt de notre approche.

Cette section décrit les algorithmes permettant d'implémenter un modèle décrit avec le formalisme présenté dans le chapitre 3 et utilisant la terminologie introduite dans la section 4.2.1 du chapitre courant résumée dans le tableau 4.1 page 111. Nous choisissons de décomposer la présentation de ces algorithmes en deux parties. Dans une première partie, nous décrivons comment sont utilisées les notions de halo, de matrice d'interaction brute et matrice d'interaction raffinée afin de construire le comportement d'une entité. Dans une seconde partie, nous décrivons comment implémenter une simulation dans le cas particulier du temps discret.

4.3.1 Processus comportemental d'une entité

Le processus utilisé par une entité pour déterminer l'interaction qu'elle souhaite initier est indépendant de la représentation du temps utilisée. Il suit un processus générique en quatre étapes résumé dans la figure 4.21 : la perception du voisinage de l'entité, le recensement des interactions que l'entité est capable d'initier dans son contexte actuel, la sélection de l'interaction initiée en fonction de son modèle comportemental et enfin l'exécution l'interaction. Dans cette section, nous proposons de décrire les concepts et algorithmes permettant d'implémenter ces différentes étapes.

Perception du voisinage

Avant de pouvoir déterminer les interactions qu'elle a l'opportunité d'effectuer, une entité doit déterminer quelles sont les entités présentes dans son voisinage. Ce calcul, décrit dans l'algorithme 1, s'appuie sur le halo de la famille de cette entité, ainsi que sur l'ensemble des entités pouvant être perçues dans l'environnement, qui sont caractérisées par la capacité de pouvoir subir au moins une interaction. Cet ensemble correspond donc à l'ensemble $\mathbb{E}_{passive}$ des entités passives de la simulation.

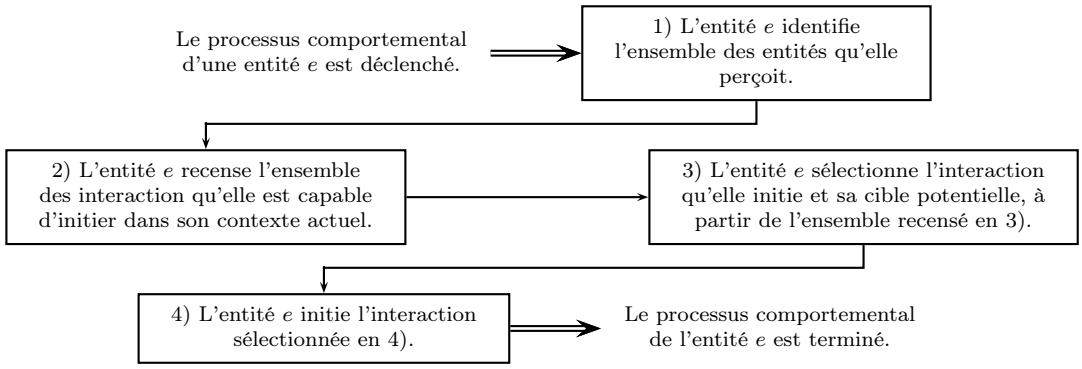


FIGURE 4.21 – Principes régissant le comportement d'une entité e dans l'approche IODA.

Algorithme 1 : Algorithme décrivant comment est calculé du voisinage d'une entité e. L'ensemble $\mathbb{E}_{passive}$ représente l'ensemble des entités passives présentes dans l'environnement.

e**.recenser voisinage() :**
début
  $\mathcal{V} \Leftarrow \emptyset$;
  pour $e' \in \mathbb{E}_{passive}$ **faire**
    si $e' \neq e$ ***et*** $halo\Big(\mathcal{H}\big(famille(e)\big)\Big)\Big(e,\, e'\Big) == vrai$ **alors**
      $\mathcal{V} \Leftarrow \mathcal{V} \cup \{e'\}$;
  retourner $\mathcal{V}$;
fin

Recensement des interactions qu'une entité a l'opportunité d'initier

Afin de construire l'ensemble des interactions qu'une entité a l'opportunité d'initier dans sont contexte actuel, nous introduisons quatre notions : les *tuples*, le critère d'éligibilité d'un *tuple*, le critère de *réalisabilité* d'un tuple et le *Potentiel d'interaction* d'une entité.

Les interactions initiées effectivement par une entité source sont soit des interactions dégénérées, soit des interactions individuelles auxquelles est associée une entité du voisinage de la source en tant que cible. Puisqu'une interaction qu'une entité a la capacité d'initier est définie par un élément d'assignation, nous représentons une interaction qu'une entité peut potentiellement effectuer par un tuple, associant à un élément d'assignation une entité source et, si l'élément d'assignation n'est pas dégénéré, une entité cible.

Définition 29. ***Tuple éligible***
Soient $s \in \mathbb{E}$ une entité dont la famille est $\mathcal{F}_S$, $t \in \mathbb{E}$ une entité dont la famille est $\mathcal{F}_T$, $a_{indiv} = (\mathcal{I}, d, \mathcal{F}_S, \mathcal{F}_T) \in \mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}$ un élément d'assignation individuel et $a_{deg} = (\mathcal{I}', \mathcal{F}_S) \in \mathbb{I}_{(1,0)} \times \mathbb{F}$ un élément d'assignation dégénéré.
Un tuple $\mathcal{T} = (a_{indiv}, s, t)$ est dit **éligible** si :
- $a_{indiv} \in \mathcal{M}(\mathcal{F}_S, \mathcal{F}_T)$, *i.e.* si s a la capacité d'initier l'interaction individuelle associée à l'élément d'assignation a_{indiv} avec t pour cible ;
- $environnement.distance(s,t) \leq dist(a_{indiv})$, *i.e.* si la distance entre s et t est inférieure ou égale à la garde de distance associée à l'élément d'assignation a_{indiv} ;
- $t \in \mathcal{V}(e)$, *i.e.* si l'entité t appartient au voisinage de l'entité s.

De même, un tuple $\mathcal{T} = (a_{deg}, s)$ est dit **éligible** si s a la capacité d'initier l'interaction dégénérée associée à l'élément d'assignation a_{deg}, *i.e.* $a_{deg} \in \mathcal{M}(\mathcal{F}_S, \emptyset)$

On dit qu'un tuple est éligible si l'entité source a la capacité d'initier l'interaction du tuple. Ce critère est syntaxique et se fonde sur l'interprétation de la matrice d'interaction brute et l'exploitation du voisinage d'une entité. Il ne porte pas sur la sémantique de l'interaction lui étant associée. Ainsi,

les préconditions ou le déclencheur d'une interaction associée à un tuple éligible peuvent très bien être fausses pour la source et la cible éventuelle contenues dans le tuple. La validité sémantique d'un tuple est vérifiée à l'aide du critère de *réalisabilité*.

Définition 30. ***Tuple réalisable***
Soient $s \in \mathbb{E}$ une entité dont la famille est $\mathcal{F}_S$, $t \in \mathbb{E}$ une entité dont la famille est $\mathcal{F}_T$, $a_{indiv} = (\mathcal{I}, d, \mathcal{F}_S, \mathcal{F}_T) \in \mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}$ un élément d'assignation individuel et $a_{deg} = (\mathcal{I}', \mathcal{F}_S) \in \mathbb{I}_{(1,0)} \times \mathbb{F}$ un élément d'assignation dégénéré.
Le tuple $\mathcal{T} = (a_{indiv}, s, t)$ est dit **réalisable** si :
- $\mathcal{T}$ est éligible, et ;
- $\mathcal{I}.preconditions(s,t) = vrai$, *i.e.* les préconditions de l'interaction $\mathcal{I}$ sont vérifiées pour les entités s et t, et ;
- $\mathcal{I}.declencheur(s,t) = vrai$, *i.e.* le déclencheur de l'interaction $\mathcal{I}$ est vérifié pour les entités s et t ;

De même, le tuple $\mathcal{T} = (a_{deg}, s)$ est dit **réalisable** si :
- $\mathcal{T}$ est éligible, et ;
- $\mathcal{I}.preconditions(s) = vrai$, *i.e.* les préconditions de l'interaction $\mathcal{I}$ sont vérifiées pour l'entité s, et ;
- $\mathcal{I}.declencheur(s) = vrai$, *i.e.* le déclencheur de l'interaction $\mathcal{I}$ est vérifié pour l'entité s ;

Les interactions qu'une entité $s \in \mathbb{E}$ peut potentiellement initier dans son contexte actuel est alors décrit par l'ensemble de tous les tuples réalisables où l'entité s est la source. Nous appelons cet ensemble particulier le *potentiel d'interaction* de l'entité s.

Définition 31. ***Potentiel d'interaction d'une entité***
On appele **potentiel d'interaction** d'une entité $e \in \mathbb{E}$ l'ensemble des tuples réalisables où cette entité apparaît comme source. Cet ensemble décrit toutes les interactions que l'entité a l'opportunité d'initier dans son contexte actuel.

L'algorithme 2 décrit comment calculer le potentiel d'interaction d'une entité, en se reposant directement sur les éléments déclaratifs du modèle IODA décrits dans le chapitre 3.

Algorithme 2 : Algorithme permettant de calculer le potentiel d'interaction d'une entité x.

```
potentiel d'interaction(x)
début
    R ⇐ ∅;
    % Recensement des tuples contenant des interactions individuelles
    pour tous les y ∈ V(x) faire
        pour tous les a ∈ M(famille(x), famille(y)) faire
            si (a, x, y) est réalisable alors
                R ⇐ R ∪ {(a, x, y)};
    % Recensement des tuples contenant des interactions dégénérées
    pour tous les a ∈ M(famille(x), ∅) faire
        si (a, x) est réalisable alors
            R ⇐ R ∪ {a, x};
    retourner R;
fin
```

Sélection réactive de l'interaction effectuée

Une fois que l'entité source a calculé sont potentiel d'interaction, elle doit déterminer quelle interaction elle va initier. Ce procédé dépend de la nature réactive, cognitive ou hybride du processus de sélection d'interaction de l'entité.

Dans le chapitre précédent, nous avons défini un modèle de sélection réactive d'interaction, fondé sur la notion de matrice d'interaction raffinée. Selon ce modèle, une priorité est associée à chaque élément d'assignation (et donc à chaque tuple). Il est toujours fait de sorte que le tuple sélectionné a la plus grande priorité parmi les tuples contenus dans le potentiel d'interaction de l'entité. Si plusieurs de ces tuples ont cette priorité, le tuple sélectionné est choisi parmi eux aléatoirement. L'algorithme 3 décrit ce procédé.

Algorithme 3 : Algorithme décrivant comment une entité utilisant un modèle de sélection réactive d'interaction détermine le tuple dont il initie l'interaction. Dans cet algorithme, on suppose disposer d'une fonction nommée *élément* permettant de connaître l'élément d'assignation d'un tuple.

sélection(x)
début
  $\mathcal{R} \Leftarrow$ potentiel d'interaction(x);
  % $\mathcal{O}$ contiendra l'ensemble des priorités des éléments d'assignation où e est une source
  $\mathcal{O} \Leftarrow \emptyset$;
  pour tous les $a \in ligne\Big(famille(x)\Big)$ **faire**
    si $\mathcal{M}_{raff}(a) \notin \mathcal{O}$ **alors**
      $\mathcal{O} \Leftarrow \mathcal{O} \cup \{\mathcal{M}_{raff}(a)\}$;
  élémentChoisi $\Leftarrow$ null;
  tant que *élémentChoisi == null* ***et*** $\mathcal{O} \neq \emptyset$ **faire**
    % Récupération de la plus grande priorité contenue dans $\mathcal{O}$
    p $\Leftarrow$ max($\mathcal{O}$);
    $\mathcal{O} \Leftarrow \mathcal{O} \backslash \{p\}$;
    % Récupération de tous les tuples de priorité p
    $\mathcal{R}_p \Leftarrow \emptyset$;
    pour tous les $\mathcal{T} \in \mathcal{R}$ **faire**
      si $\mathcal{M}_{raff}\Big(\textit{élément}(\mathcal{T})\Big) == p$ **alors**
        $\mathcal{R}_p \Leftarrow \mathcal{R}_p \cup \{\mathcal{T}\}$;
    si $\mathcal{R}_p \neq \emptyset$ **alors**
      élémentChoisi $\Leftarrow$ élément au hasard de $\mathcal{R}_p$;
  retourner élémentChoisi;
fin

Dans des simulations où le voisinage des entités contient un nombre élevé d'entités, et où les entités peuvent initier une grande variété d'interactions, l'algorithme présenté précédemment, bien que correct, n'est pas satisfaisant. En effet, en supposant que le voisinage d'une entité contienne n_v voisins, qu'une entité ait la capacité d'initier n_d interactions dégénérées et n_i interactions individuelles, le calcul du potentiel d'interaction nécessite $n_d \times n_i + n_d$ tests d'éligibilité et au pire $n_d \times n_i + n_d$. Ce nombre important de calculs n'est pas inévitable dans le cas d'une sélection réactive. En effet, on sait que si un tuple de priorité p est sélectionné, alors il n'est même pas la peine d'effectuer le test d'éligibilité et de réalisabilité pour les tuples de priorité inférieure. On introduit donc la notion de potentiel d'interaction de priorité $p \in \mathbb{Z}$ pour remédier à ce problème.

Définition 32. ***Potentiel d'interaction de priorité*** p
Le **potentiel d'interaction de priorité** $p \in \mathbb{Z}$ d'une entité contient tous les tuples réalisables du potentiel d'interaction de cette entité, dont l'élément d'assignation a pour priorité p.

Cette optimisation aboutit à l'algorithme 4.

Initiation d'une interaction

Une fois un tuple réalisable sélectionné, une entité initie l'interaction lui correspondant. Cette étape consiste à exécuter les actions de l'interaction, en remplaçant chaque occurrence d'une primitive abstraite

Algorithme 4 : Algorithme optimisé décrivant comment une entité utilisant un modèle de sélection réactive d'interaction détermine le tuple dont il initie l'interaction. Dans cet algorithme, on suppose disposer d'une fonction nommée *priorité* permettant de connaître la priorité de l'élément d'assignation d'un tuple, à l'aide de la matrice d'interaction raffinée.

```
sélection(x)
début
    % O contiendra l'ensemble des priorités des éléments d'assignation où e est une source
    O ⇐ ∅;
    pour tous les a ∈ ligne(famille(x)) faire
        si M_raff(a) ∉ O alors
            O ⇐ O ∪ {M_raff(a)};
    élémentChoisi ⇐ null;
    tant que élémentChoisi == null et O ≠ ∅ faire
        % Récupération de la plus grande priorité contenue dans O
        p ⇐ max(O);
        O ⇐ O\{p};
        % Récupération de tous les tuples de priorité p
        R_p ⇐ potentiel d'interaction de priorité(x, p);
        si R_p ≠ ∅ alors
            élémentChoisi ⇐ élément au hasard de R_p;
    retourner élémentChoisi;
fin
```

par la spécification lui correspondant dans la famille d'entités associée, et chaque occurence d'une primitive de l'environnement par sa spécification.

4.3.2 Simulation en temps discret

L'algorithme général permettant d'effectuer une simulation dépend grandement du modèle du temps utilisé. Dans cette thèse, nous décrivons comment faire pour implémenter une simulation usant d'un modèle de temps discret. Les algorithmes présentés jusqu'à présent restent toutefois valides pour toute autre représentation du temps.

Le modèle de temps discret que nous utilisons modélise le temps comme un ensemble discret et totalement ordonné d'unités atomiques de temps, que nous appelons pas de temps. La simulation avance dans le temps en parcourant de manière séquentielle cet ensemble. Afin de garantir la cohérence de la simulation, les entités doivent être à jour avant de percevoir et pouvoir initier une quelconque interaction lors d'un pas de temps. Par conséquent, **un pas de temps commence par mettre à jour chaque entité de la simulation**, avant de donner la possibilité aux entités d'exécuter leur processus comportemental.

Le nombre de pas de temps que dure une simulation dépend d'un critère fixé lors de la préparation des expérimentations. La description de ce critère sort du cadre de cette thèse.

Mise à jour des entités

La mise à jour d'une entité consiste à initier toutes les interactions de mise à jour présentes dans la cellule de la matrice de mise à jour ordonnée associée à la famille de cette entité dont les préconditions et le déclencheur sont vérifiés, dans l'ordre défini par leur priorités. L'algorithme 5 décrit ainsi comment est mis à jour l'état d'une entité e.

Déclenchement du processus comportemental des entités

Puisqu'un pas de temps est une unité de temps indivisible, en théorie, toutes les entités de l'environnement doivent sélectionner l'interaction qu'elles souhaitent initier en parallèle. Cette simultanéité

Algorithme 5 : Algorithme décrivant comment a lieu la mise à jour de l'état d'une entité e.

```
miseAJour(e)
début
    % O contiendra l'ensemble des priorités des interactions de mise à jour de e
    O ⇐ ∅;
    pour tous les u ∈ U(famille(e)) faire
        si U_ord(u) ∉ O alors
            O ⇐ O ∪ {U_ord(u)};
    tant que O ≠ ∅ faire
        p ⇐ max(O);
        O ⇐ O\{p};
        % Exécution de toutes les interactions de mise à jour de e de priorité p
        pour tous les u ∈ U_ord^-1(famille(e), p) faire
            si I(u).preconditions(e) == Vrai et I(u).declencheur(e) == Vrai alors
                I(u).actions(e);
fin
```

introduit deux problèmes non triviaux à résoudre : la détection de conflits pouvant survenir lors de l'exécution de deux interactions modifiant de manière non compatible l'environnement, ou l'état d'une entité. Par exemple deux entités ne peuvent pas ramasser la même entité cible. Des théories telles que le modèle Influence/Réaction [FM96] et ses extensions [WH03, Mic07] offrent un cadre formel permettant de spécifier de tels problèmes.

Une telle représentation du temps nécessite d'une part un grand nombre de calculs et d'autre part de déterminer dans le modèle comment détecter et résoudre les conflits. Dans cette thèse, nous utilisons une représentation du temps presque équivalente, que l'on retrouve de manière récurrente dans les plateformes de simulations. Cette approche se fonde sur le constat que, dans la majorité des cas, lorsque plusieurs actions sont en conflit, l'action initiée est choisie aléatoirement. Une façon alternative permettant d'éviter de tels conflits consiste a faire agir chaque entité en une séquence prédéfinie. La résolution de conflits aléatoire est alors assurée si la séquence utilisée pour faire agir les entité est mélangée au début de chaque pas de temps.

L'algorithme 6 résume comment s'exécute une simulation en temps discret.

4.4 Synthèse du chapitre

Dans ce chapitre, nous montrons comment utiliser le modèle formel IODA afin d'établir une approche transversale de conception où un modèle est construit graduellement puis implémenté selon des principes pouvant être automatisés.

Conception graduelle d'un modèle La conception graduelle d'un modèle dans IODA est assurée par une méthodologie de conception reposant sur 13 étapes, résumées dans le tableau 4.2, où :

- un modèle est conçu à l'aide de **représentations graphiques** qui facilitent sa description ;
- la construction de modèles conséquents est facilitée par la **complétion automatique de certains éléments du modèle**. Par exemple, la description des conditions et des actions d'une interaction, et l'ajout de cette interaction dans la matrice d'interaction implique la génération automatique de primitives abstraites vides que les familles d'agents pouvant initier ou subir l'interaction doivent spécifier ;
- **chaque étape** de la méthodologie se **focalise sur un niveau d'abstraction** particulier du modèle ;
- la **conception progressive** d'un modèle est **guidée** en suivant les dépendances entre étapes de la méthodologie ;

Algorithme 6 : Algorithme général d'une simulation en temps discret. Dans cet algorithme, on note $\mathbb{E}_{labile}$ l'ensemble des entités labiles de la simulation.

```
effectuerSimulation()
début
    t ⇐ 0;
    tant que ¬ la simulation est terminée faire
        % Début d'un nouveau pas de temps
        t ⇐ t + 1;
        % Mise à jour de l'état des entités
        pour tous les e ∈ E_labile faire
            e.miseAJour();
        % Exécution du processus comportemental des entités
        Mélanger aléatoirement le contenu de E_active;
        pour tous les e ∈ E_active faire
            % Sélection du tuple réalisable dont l'interaction est effectuée
            T ⇐ sélection(e);
            si T ≠ null alors
                Exécuter l'interaction représentée par le tuple T;
fin
```

– les hypothèses de fonctionnement du phénomène sont introduites à la fin du processus de conception, lors de la spécification des primitives abstraites des agents. La conception est donc focalisée sur des descriptions plus objectives du phénomène.

Algorithmes de simulation Nous montrons qu'il est possible d'écrire des algorithmes de simulation reposant sur le modèle IODA afin de construire un simulateur. Ces algorithmes permettent de **conserver la structure du modèle lors de l'implémentation**, et ainsi constituer une **architecture modulaire facilement modifiable** favorisant les tests unitaires. Ils permettent de plus de construire des **bibliothèques d'interactions réutilisables**, facilitant ainsi les révisions du modèle.

Nous positionnons ensuite IODA vis à vis de quatre questions se posant systématiquement dans toute approche de conception transversale. Nous mettons ainsi en lumière certains autres avantages de IODA facilitant la conception de simulation.

Toute entité est-elle un agent ? De nombreux concepteurs en programmation multi-agents utilisent plusieurs notions pour définir les entités utilisées dans leurs simulations. Ces dernières sont concrétisées sous divers noms, allant de « agent » à « objet », en passant par « artefact », « patch », *etc.* Nous montrons que ces différentes **entités** peuvent être **avantageusement unifiées** avec IODA afin d'être traitées par **un seul et même algorithme** facilitant à la fois les processus de conception et d'implémentation. Dans la phase de conception, la question de l'appartenance familiale des entités ne se pose plus, simplifiant d'autant le processus de réflexion. Dans la phase d'implémentation, il n'est plus nécessaire d'avoir différentes structures de données et surtout différents algorithmes pour traiter chacune des familles d'entités du logiciel. Le logiciel devient alors **plus léger et plus maintenable**.

Une méthodologie de conception doit-elle être dogmatique ? Une méthodologie dogmatique n'est pas appropriée à la construction itérative de modèles, qui est pourtant le fondement de la simulation. En effet, la description du modèle à un niveau d'abstraction peut parfois mettre en lumière des lacunes, amenant à compléter les spécifications d'un niveau d'abstraction plus élevé. Par exemple, la description des conditions d'une interaction peut amener à identifier d'autres interactions. Un processus dogmatique ne permet pas un tel « retour en arrière » dans ses étapes.

L'approche IODA ne souffre pas de ce problème, puisqu'elle repose sur un unique modèle, et sur une **méthodologie dont les étapes peuvent être entremêlées aisément**. En effet, les seules contraintes

TABLE 4.2 – Méthodologie utilisée pour concevoir des modèles dans IODA, décomposée en 13 étapes. Chaque étape est caractérisée dans ce tableau synthétique par l'élément du modèle qu'elle édite, son objectif et les moyens graphiques utilisés pour éditer le modèle. Le modèle spécifié lors de chaque étape a une influence ou un effet sur le modèle édité lors d'autres étapes qui peuvent être automatisés. Par exemple, l'ajout d'une interaction lors de l'étape C implique l'ajout d'une interaction devant être spécifiée à l'étape D. La colonne « pré-requis » exprime cette relation, en spécifiant quelles étapes ont un effet ou une influence sur l'étape de la ligne. Elle contient soit le nom de ces étapes, des opérateurs $\vee$ signifiant que deux étapes peuvent séparément avoir un effet sur cette étape, et des opérateurs $\wedge$ signifiant que deux étapes n'ont un effet que conjointement sur cette étape. Par exemple, le pré requis B $\wedge$ C de l'étape H signifie que pour ajouter une interaction de mise à jour dans la matrice de mise jour, l'interaction doit avoir été ajoutée lors de l'étape B, et l'entité à laquelle l'interaction est attribuée doit avoir été ajoutée lors de l'étape C.

Étape	Modèle édité	Objectif de l'étape	Moyens graphiques	Pré-requis
A	Topologie l'environnement	Indiquer la signification de la distance, en lui donnant une unité	aucun	aucun
B	Identificateur des entités	Identifier le « nom » des entités impliquées dans la simulation	Construction d'une liste de noms	aucun
C	Identificateur des interactions	Identifier le « nom » des interactions apparaissant dans la simulation	Construction d'une liste de noms	aucun
D	Conditions et actions des interactions	Décrire les conditions et les actions des interactions, à l'aide de primitives abstraites identifiées à cette occasion	Description algorithmique	C
E	Matrice d'interaction brute	Construire la matrice d'interaction brute en plaçant des noms d'interaction dans les cellules de la matrice	Faire glisser des éléments de la liste des interactions dans les cellules d'une matrice	A $\wedge$ B $\wedge$ C
F	Halo des entités	Décrire comment le halo des entités détermine si une entité est perçue ou non	Description algorithmique	A $\wedge$ B
G	Matrice d'interaction raffinée	Construire la matrice d'interaction raffinée, en attribuant une priorité à chaque élément de la matrice d'interaction brute	Associer un entier à chaque élément présent dans la matrice d'interaction brute	E
H	Matrice de mise à jour	Construire la matrice de mise à jour en plaçant des noms d'interaction dans les cellules de la matrice	Faire glisser des éléments de la liste des interactions dans les cellules d'une matrice	B $\wedge$ C
I	Primitives de l'environnement	Spécifier l'effet de chaque primitive fournie par l'envionnement	Description algorithmique	A $\vee$ (A $\wedge$ B) $\vee$ D
J	Matrice de mise à jour ordonnée	Construire la matrice de mise à jour ordonnée, en attribuant une priorité à chaque interaction de la matrice de mise à jour	Associer un entier à chaque élément présent dans la matrice de mise à jour	H
K	Attribut des entités	Identifier les attributs d'une entité	aucune	L
L	Primitives abstraites des entités	Spécifier l'effet de chaque fournie primitive par la famille d'agents	description algorithmique	F $\vee$ (D $\wedge$ G) $\vee$ (D $\wedge$ J)
M	Attributs de l'environnement	Identifier les attributs de l'environnement	aucune	M

sur notre méthodologie sont d'ordre logique. Par exemple, une primitive abstraite n'est spécifiée par une entité que si cette entité est la source ou la cible d'une interaction manipulant la primitive dans ses conditions ou actions.

Faut-il prendre en compte les performances lors de la modélisation ? La conception d'une simulation performante est un problème lié à l'implémentation. Il est paradoxal de le prendre en compte lors de la modélisation, puisque les experts du domaine doivent être impliqués au maximum dans la construction du modèle.

L'utilisation d'un modèle décrivant un phénomène à différents niveaux d'abstraction fournit un compromis à ce problème. En effet, les descriptions effectuées à un niveau fin d'abstraction sont par définition les plus proches de l'implémentation. Il est donc moins gênant que la question des performances y soit considérée.

Faut-il tout décrire au sein d'une règle ? Déterminer le **montant d'informations** à décrire **dans les interactions**, et la part à **déléguer dans** la spécification des **primitives** des entités est un problème **complexe**, auquel seul le concepteur de la simulation peut répondre.

Ce point n'est toutefois **pas une limitation intrinsèque à l'approche IODA**, puisqu'une question similaire se pose dans toute approche réifiant les actions entreprises par des entités : faut-il décrire un comportement complexe par une seule action complexe, ou par un ensemble d'actions simples ?

Preuve des avantages attendus de IODA Afin de confirmer les avantages théoriques de IODA, nous décrivons dans le chapitre suivant une implémentation fidèle de notre modèle formel, de nos algorithmes, et de notre méthodologie en une plateforme de simulation appelée JEDI, ainsi qu'un environnement de développement intégré nommé JEDI-BUILDER.