IODA comme modèle formel

Plan du chapitre :

Nous avons établi dans la partie précédente que pour répondre aux besoins inhérents aux simulations large échelle le modèle doit exprimer un certain nombre de propriétés. Dans ce chapitre, nous définissons formellement le modèle de notre approche ***Interaction Oriented Design of Agent simulations*** *(*IODA*) et montrons que les concepts y étant développés sont conformes à ces besoins. En conséquence,* IODA *facilite la conception de simulations large échelle.*

Pour cela, la section 3.1 commence par fixer le sens donné dans IODA *aux termes « agent », « environnement » et « temps ». Nous présentons ensuite les six concepts se trouvant au cœur de l'approche* IODA *et justifions en quoi ils facilitent la conception de simulations large échelle. Ces concepts sont :*

- *les interactions (section 3.2) ;*
- *la matrice d'interaction (section 3.3) ;*
- *les entités (section 3.4) ;*
- *la matrice de mise à jour (section 3.4.1) ;*
- *l'environnement (section 3.5) ;*
- *le modèle de sélection d'interaction (section 3.6)*

Les simulations large échelle peuvent difficilement être exécutées si le comportement des agents nécessite un grand nombre de calculs, puisque dans de tels cas, seul un nombre restreint d'agents peuvent être simulés. Le compromis le plus couramment utilisé consiste à utiliser un modèle de sélection d'action réactif, dans lequel les règles condition/effet peuvent être plus ou moins cognitives. Nous intégrons à IODA *un modèle de sélection d'interaction réactif fondé sur ce principe, à l'aide d'une matrice d'interaction raffinée.*

Le modèle formel utilisé dans l'approche IODA [KMP08a, KMP08c] est structuré en six parties déclarées indépendamment : les interactions, la matrice d'interaction, l'environnement, les entités, la matrice de mise à jour et le modèle de sélection d'interaction (*c.f.* figure 3.1). Nous présentons ce modèle selon ces six axes en regroupant toutefois la présentation du modèle des entités et de la matrice de mise à jour dans une même section.

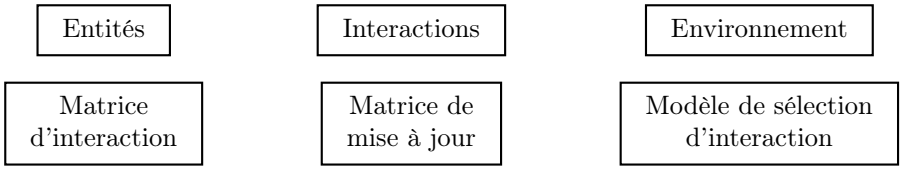


FIGURE 3.1 – Les six principales parties d'un modèle IODA.

3.1 Cadre général

Le modèle IODA est basé sur le paradigme agent et fait donc naturellement apparaître au moins trois notions : les agents, l'environnement et la représentation du temps. Ces notions peuvent prendre des sens très différents et avoir des représentation plus ou moins complexes selon les choix de modélisation effectués [KMP07]. Cette première section a pour but de dégager le sens de chacune de ces notions dans l'approche IODA, afin de définir le cadre général de ce chapitre.

3.1.1 Agent

La notion d'agent est fortement connotée et son sens est très dépendant du contexte de son utilisation. Ainsi, l'utilisation de ce terme dans ce chapitre pourrait entraver la compréhension des principes que nous énonçons. Par conséquent, bien que cela puisse paraître surprenant pour une approche de conception multi-agents, nous n'utilisons pas le terme « agent » dans ce chapitre : nous lui préférons le terme neutre « entité ». Ce choix est plus amplement justifié dans la section 4.2.1 page 109 du chapitre 4.

De plus, nous distinguons d'une part le terme *entité* qui se réfère à une entité apparaissant physiquement lors d'une simulation et d'autre part les termes *famille d'entités* qui se réfèrent à un ensemble abstrait d'entités. Cette différence est similaire à la différence entre instance et classe dans les langages orientés-objet.

Définition 2. ***Distinction Instance/Famille d'entités***
Une **famille d'entités** est une spécification abstraite d'entités pouvant initier et subir les mêmes interactions et ayant le même comportement. Une **entité** (ou **instance d'entité**) est une instance particulière d'une famille d'entités.
Une **instance d'entité** a une seule famille d'entités et une famille d'entités peut être réduite à une seule instance.

3.1.2 Environnement

La notion d'environnement est partagée entre deux sens :

1. « tout ce qui est extérieur à une entité » ;
2. « l'espace dans lequel évoluent les entités d'une simulation ».

La première définition est issue de la systémique. Elle est principalement utilisée pour concevoir le comportement de robots, où chaque entité doit construire elle-même son modèle du monde (et donc des autres entités) à partir de ce que ses senseurs perçoivent. Au contraire, en simulation explicative le modèle du monde (et donc l'environnement) fait partie des hypothèses émises sur l'origine du phénomène étudié. Par conséquent, nous considérons uniquement la seconde définition dans cette thèse.

Pour des raisons que nous exposons dans la section 3.3 page 77, l'environnement doit disposer d'une métrique permettant de calculer la distance séparant deux entités. Cette métrique n'est pas limitée à la distance euclidienne. Elle peut exprimer tout type de mesure de proximité dans un espace topologique : une distance de von Neumann, un degré de différence entre deux entités (par exemple de richesse, d'ethnie, de taille, *etc.*), un nombre d'arcs séparant deux entités dans un graphe, *etc.* Cette généralisation de la notion de distance repose sur une métaphore et des concepts similaires à ceux développés par Giavitto *et al.*[GM02] pour construire leur langage de programmation MGS. Dans leur approche, un calcul est modélisé comme un chemin parcouru dans un espace de données, qui peut être décomposé en étapes au cours desquelles un calcul élémentaire est effectué. Ils y définissent en particulier la notion de voisinage dans un espace de données pouvant directement être utilisée dans IODA pour définir la notion de distance.

Notation 1.
On pose $\mathbb{F}$ l'ensemble de toutes les familles d'entités d'une simulation, $\mathbb{E}$ l'ensemble de toutes les instances d'entité présentes dans l'environnement d'une simulation ainsi que $e \prec F$ le fait qu'une instance e ait pour famille d'entité F :

$$\forall e \in \mathbb{E}, \forall \mathcal{F} \in \mathbb{F}, (e \text{ a pour famille d'entité } F \Leftrightarrow e \prec F)$$

De plus, on définit la fonction permettant de connaître la famille d'une entité :

$$famille : \begin{array}{ccc} \mathbb{E} & \rightarrow & \mathbb{F} \\ e & \rightarrow & \mathcal{F} \text{ tel que } e \prec F \end{array}$$

3.1.3 Temps

Dans le cadre de la simulation, un modèle n'a de sens que s'il peut être utilisé pour construire un simulateur. À cette fin, il faut pouvoir faire évoluer le modèle dans le temps, afin de simuler le comportement du phénomène. Par conséquent, que cela soit fait implicitement ou explicitement, le modèle d'un phénomène spécifie toujours un modèle du temps.

Pour des raisons pratiques, le modèle IODA décrit dans cette thèse repose sur une représentation simple du temps dans laquelle chaque entité n'initie qu'une seule interaction en réponse au déclenchement de son processus comportemental. Il faut toutefois remarquer qu'il ne s'agit pas d'une limite de l'approche IODA, mais d'un choix effectué pour éviter d'ajouter des éléments superflus à la compréhension de notre approche.

3.2 L'interaction enfin concrétisée

Dans IODA, la spécification des simulations est centrée sur la notion d'*interaction* qui sert d'élément de base pour construire le comportement des entités. Une interaction est une séquence sémantique d'actions qui implique simultanément un nombre fixé d'entités et dont l'exécution est soumise à des conditions.

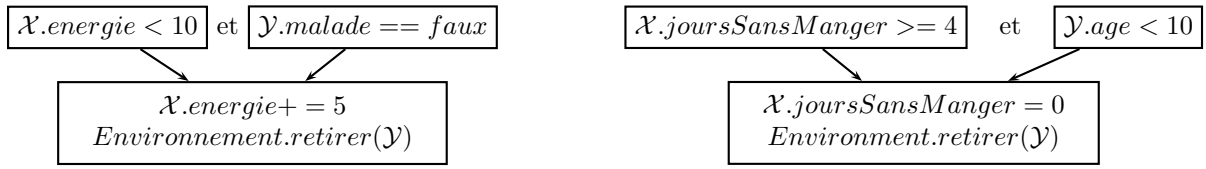
3.2.1 Polymorphisme des interactions

La présence de règles de type *conditions/actions* n'est pas rare dans les systèmes multi-agents. Toutefois, les approches décrites dans l'état de l'art reposant sur de telles règles ont un défaut commun : chaque interaction est en pratique exprimée avec des règles spécifiques à chaque entité interagissant.

Exemple.
La figure 3.2 présente la description de deux interactions sous la forme de règles appelées respectivement INTERACTION1 *et* INTERACTION2. *Ces règles semblent reposer sur des principes totalement différents puisque dans un cas l'entité $\mathcal{X}$ initie l'interaction* INTERACTION1 *en fonction d'un seuil d'énergie alors que dans l'autre cette entité initie l'interaction* INTERACTION2 *en fonction du nombre de jours de diète.*

Cet exemple montre que deux règles pourtant différentes peuvent désigner une même interaction (en l'occurrence, le fait qu'une entité mange une autre entité). Ce sens commun aux deux règles donne l'intuition de l'existence d'une description générique de l'interaction MANGER à laquelle ces deux règles se conforment. De plus, cette décomposition montre que les conditions d'une interaction peuvent être divisées en deux parties : une partie décrivant les pré-requis logiques ou physiques nécessaires à l'exécution de l'interaction et une partie décrivant les buts explicites ou implicites auxquels l'interaction répond.

Exemple.
L'interaction MANGER *mentionnée ici peut être exprimée de manière générique. En effet, $\mathcal{X}.energie < 10$ et $\mathcal{X}.joursSansManger >= 4$ sont deux façons différentes d'exprimer que $\mathcal{X}$ ressent la sensation de*



(a) Description de INTERACTION1 (b) Description de INTERACTION2

FIGURE 3.2 – Description de deux règles nommées INTERACTION1 et INTERACTION2 qui représentent une même interaction MANGER pour des entités aux spécificités différentes. Dans ce schéma, la partie supérieure de la figure représente les conditions de la règle et la partie inférieure les actions de la règle. **Ces descriptions sont utilisées pour illustrer le problème du polymorphisme des interactions. Elles ne sont pas exprimées selon l'approche défendue dans IODA.**

faim. De même, $\mathcal{Y}.age < 10$ et $\mathcal{Y}.malade == faux$ expriment que l'entité mangée doit être saine. Cette description générique est illustrée par la suite dans la figure 3.3b.

Le modèle IODA se fonde sur ce constat et considère que toute interaction peut être décrite de manière générique et indépendante des entités y participant. Pour cela, les conditions et les actions d'une interaction ne manipulent pas directement l'état des entités, mais des primitives abstraites appelées *primitives de perception* ou *d'action.*

Définition 3. ***Primitive d'action***
Une **primitive d'action** est une primitive abstraite manipulée dans les actions d'une interaction qui a un effet de bord sur au moins une des entités participant à l'interaction.
Une primitive d'action est un triplet $p = \langle id, V, (a_i)_{i\in\mathbb{N}}\rangle$ où :
– id est l'identifiant de la primitive, représenté sous la forme d'une chaîne de caractères ;
– V est un élément optionnel décrivant le type de la valeur retournée par la primitive ;
– $(a_i)_{i\in\mathbb{N}}$ représente l'ensemble ordonné (éventuellement vide) des paramètres de la primitive.

Les conditions d'une interaction ne doivent pas avoir d'effet de bord sur leurs arguments, *i.e.* les entités pour lesquelles les conditions sont testées. Par conséquent, les primitives manipulées dans ces méthodes de l'interaction ne peuvent pas manipuler des primitives d'actions. À la place, elles manipulent des *primitives de perception.*

Définition 4. ***Primitive de perception***
Une **primitive de perception** est une primitive abstraite manipulée dans les conditions ou les actions d'une interaction qui n'a aucun effet de bord sur les entités participant à l'interaction.
Une primitive de perception est un triplet $p = \langle id, V, (a_i)_{i\in\mathbb{N}}\rangle$ où :
– id est l'identifiant de la primitive, représenté sous la forme d'une chaîne de caractères ;
– V est un élément optionnel décrivant le type de la valeur retournée par la primitive ;
– $(a_i)_{i\in\mathbb{N}}$ représente l'ensemble ordonné (éventuellement vide) des paramètres de la primitive.

Exemple.
La figure 3.3b spécifie de manière générique l'interaction MANGER. *Elles repose sur deux primitives de perception appelées $aFaim()$ et $estSain()$ qui déterminent respectivement si l'entité $\mathcal{X}$ a pour but implicite de réduire sa sensation de faim et si l'entité $\mathcal{Y}$ est considérée comme saine. Elle repose de plus sur une primitive d'action appelée $reduireFaim()$ qui a pour action de réduire la sensation de faim de $\mathcal{X}$ en fonction des propriétés de $\mathcal{Y}$.*

Pour participer à une interaction, une entité doit implémenter les primitives de perception et d'action selon ses propres spécificités. Le polymorphisme des interactions est ainsi garanti tout en favorisant la réutilisation des interactions existantes.

Exemple.
La primitive abstraite $\mathcal{X}.aFaim()$ peut avoir pour sens :

- *« Avoir son énergie sous un certain seuil ». Dans ce cas, l'entité implémente cette primitive sous une forme similaire à : $\mathcal{X}.energy < 20$. On obtient alors par polymorphisme une interaction similaire à* Interaction1 *de la figure 3.2 ;*
- *« Ne pas avoir mangé pendant un certain temps ». Dans ce cas, l'entité implémente cette primitive sous une forme similaire à : $\mathcal{X}.joursSansManger > 10$. On obtient alors par polymorphisme une interaction similaire à* Interaction2 *de la figure 3.2 ;*
- *« Prédire le décès de $\mathcal{X}$ si jamais il ne mange pas dans les 10 prochains jours ». Dans ce cas, l'entité implémente cette primitive avec un algorithme d'anticipation plus ou moins complexe, etc.*

Il est à noter que l'environnement est toujours implicitement présent dans une interaction. Les conditions et les actions d'une interaction peuvent donc aussi manipuler des informations relatives à la topologie de l'environnement. La distance entre deux entités en est un exemple. Nous désignons ces primitives particulières par les termes *primitives de l'environnement*. Leur définition est similaire à celle d'une primitive abstraite d'une entité (voir section 3.5).

La **description générique des conditions et des actions rend la spécification des interactions indépendante des entités pouvant y participer**. Par conséquent, il y a séparation explicite entre l'étape de spécification des interactions et l'étape de spécification des capacités des entités. La réutilisation de cette description pour des entités différentes en devient plus aisée. De plus des **librairies d'interactions réutilisables sont construites** au fil des simulations.

3.2.2 Structure d'une interaction

Les interactions sont structurées de manière à pouvoir être utilisées aussi bien dans des architectures comportementales réactives que dans des architectures à planification délibérative ou à planification réactive. Pour ce faire, elles ont toutes une représentation homogène sous la forme de règles, composées de trois parties :

- Les **préconditions** d'une interaction décrivent des pré-requis logiques ou physiques nécessaires à l'exécution de l'interaction ;
- Le **déclencheur** d'une interaction décrit les pré-requis téléonomiques de l'interaction, *i.e.* les buts explicites ou implicites auxquels l'interaction cherche à répondre ;
- Les **actions** d'une interaction décrivent l'effet de l'interaction sur les entités y participant ainsi que sur l'environnement.

Cette structure est résumée et illustrée pour une interaction nommée Manger sur la figure 3.3. Dans cette figure, les préconditions de l'interaction Manger spécifient qu'une entité en mange une autre seulement si l'entité mangée est saine (par exemple si elle n'est pas toxique). Le déclencheur y spécifie que l'entité doit avoir faim pour déclencher cette interaction. Il sous-entend ainsi que le but implicite de cette interaction est de réduire la sensation de faim de l'entité. Enfin, les actions décrivent les effets de cette interaction, en l'occurrence la réduction de la sensation de faim et le retrait de l'entité mangée de l'environnement.

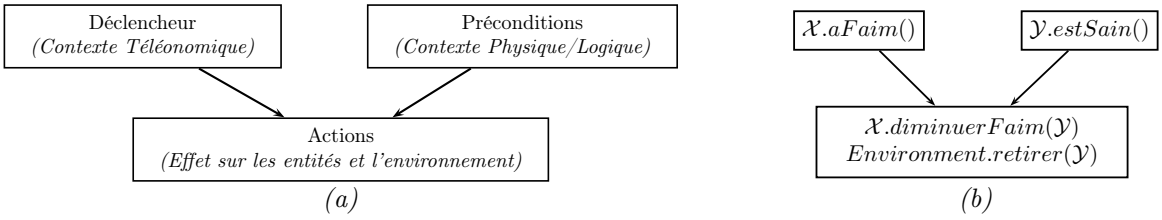


Figure 3.3 – Structure générale d'une interaction dans Ioda (a) illustrée pour l'interaction Manger (b) qui décrit comment et sous quelles conditions une entité $\mathcal{X}$ mange une entité $\mathcal{Y}$.

3.2.3 Typologie des interactions

Les entités impliquées dans une interaction n'y jouent généralement pas la même fonction. Pour des raisons exposées ultérieurement, nous différencions en particulier les entités qui initient une interaction des entités qui subissent l'interaction par les termes *source* et *cible*.

Définition 5. ***Entité Source/Cible d'une interaction***
On appelle entités **source** d'une interaction l'ensemble des entités qui initient l'interaction, *i.e.* les entités dont la participation dans l'interaction est le fruit de leurs propres décisions. On appelle entités **cible** d'une interaction l'ensemble des entités qui subissent cette interaction, *i.e.* les entités dont la participation dans l'interaction est le fruit de la décision d'une autre entité.

Exemple.
Dans l'interaction MANGER *décrite dans la figure 3.3b, l'entité $\mathcal{X}$ est l'unique source de l'interaction* MANGER *et l'entité $\mathcal{Y}$ en est l'unique cible.*

Une interaction est caractérisée par le nombre de sources et de cibles qu'elle implique. Nous les appelons *cardinalité de l'interaction*.

Définition 6. ***Cardinalité d'une interaction***
La **cardinalité d'une interaction** $\mathcal{I}$ est un couple $\big(card_S(\mathcal{I}), card_T(\mathcal{I})\big)$ où $card_S(\mathcal{I})$ (resp. $card_T(\mathcal{I})$) représente le nombre d'entités source (resp. cible) impliquées dans $\mathcal{I}$.
On notera par la suite $card(\mathcal{I})$ la cardinalité d'une interaction $\mathcal{I}$.

Exemple.
L'interaction MANGER *décrite dans la figure 3.3b a pour cardinalité* $(1, 1)$.

Nous étudions dans cette section une division de l'espace des cardinalités d'une interaction en cinq ensembles.

Les interactions telles que $card_S(\mathcal{I}) = 0$. Ces interactions n'ont aucune entité source, ce qui sous-entend qu'aucune entité ne peut décider d'initier l'interaction. Compte tenu du fait que seule une entité peut décider d'initier une interaction de tels cas ne peuvent exister.

Les interactions telles que $card(\mathcal{I}) = (1, 1)$. Les interaction de cardinalité $(1, 1)$ impliquent exactement une source et une cible. Il s'agit du cas trivial d'interaction rencontré en simulation.

Exemple.
L'interaction SUIVRE *permet à une entité source de définir un de ses déplacements en fonction de la position d'une entité cible.*

Puisque ce cas se retrouve fréquemment dans les simulations, elles sont par la suite désignées par les termes *interactions individuelles*.

Définition 7. ***Interaction individuelle***
Une **interaction individuelle** est une interaction dont la cardinalité est $(1, 1)$.

Les interactions telles que $card(\mathcal{I}) = (1, 0)$. Le second type d'interaction le plus usuellement rencontré a pour cardinalité $(1, 0)$. Elles représentent les interactions effectuées par une entité source afin d'influer sur elle-même et son propre état. Elles sont considérées comme des interactions car elles prennent implicitement leur source pour cible. Les exemples de telles interactions sont nombreux et existent dans un vaste ensemble de domaines d'application.

Exemple.
Parmi les interactions de cardinalité $(1, 0)$*, on peut trouver en éthologie :*
- *l'interaction* DORMIR *qui fait passer la source dans un état de sommeil. Elle induit un changement de rythme biologique tant que l'interaction* SE RÉVEILLER *n'est pas initiée ;*
- *l'interaction* MOURIR *qui retire l'entité source de l'environnement ;*

– *l'interaction* ERRER *qui déplace l'entité dans l'environnement.*

Parmi les interactions de cardinalité $(1, 0)$*, on peut trouver en biochimie :*

– *en biochimie l'interaction* MUTER *qui modifie la structure génétique de l'entité source en fonction d'une probabilité ;*
– *en biochimie l'interaction* MITOSE *qui divise l'entité en deux entités dans l'environnement.*

Puisque ces interactions sont retrouvée fréquemment en simulation, elles sont par la suite désignées par le terme *interaction dégénérée.*

Définition 8. ***Interaction dégénérée***
Une **interaction dégénérée** est une interaction dont la cardinalité est $(1, 0)$.

Les interactions telles que $card_T(\mathcal{I}) = n$ **ou** $card_S(\mathcal{I}) = m, \forall n > 1, \forall m > 1$**.** La complexité de certains problèmes à simuler nécessite la prise en compte d'interactions mettant en scène plus de deux entités et en particulier plus d'une seule cible.

Exemple.

Exemples d'interactions de cardinalité $(1, n), \forall n > 1$ *:*

– *l'interaction* CATALYSER *en biochimie, qui a pour cardinalité* $(1, 2)$*. Dans cette interaction, la source (le plus souvent une enzyme) favorise une réaction chimique ayant lieu entre deux autres entités moléculaires ou plus (les cibles) ;*
– *l'interaction* DIFFUSER *en éthologie, qui a pour cardinalité* $(1, 8)$*. Cette interaction suppose que l'environnement est pavé par une grille d'entités, qui représentent chacune la concentration d'une phéromone dans une parcelle de l'environnement. Elle est exécutée par une entité source afin de diffuser une partie des phéromones qu'elle contient dans les parcelles de l'environnement à proximité.*

Dans d'autres cas, il se révèle même nécessaire d'impliquer plus d'une seule source. De telles interactions dénotent le plus souvent un problème de coordination entre entités.

Exemple.

Exemples d'interactions de cardinalité $(m, n), \forall m > 1, \forall n \geq 0$ *:*

– *l'interaction* SE REPRODUIRE *en éthologie, qui a pour cardinalité* $(2, 0)$*. Dans cette interaction, deux sources ajoutent à l'environnement leur descendance. Dans cette interaction, il n'y a pas de cible puisque la participation de chaque entité est censée provenir du processus décisionnel de chacune des deux sources ;*
– *l'interaction* TRANSPORTER À 4 *en logistique, qui a pour cardinalité* $(4, 1)$*. Dans cette interaction, quatre sources doivent coopérer afin de transporter une entité trop lourde pour être transportée par 3 entités ou moins.*

En pratique, les interactions de cardinalité (m, n) ne sont pas utilisées. En effet, seule l'une des m sources décide d'initier l'interaction. Les autres sources se contentent d'accepter ou de refuser d'y participer. En pratique ce type d'interaction s'exprime donc avec une cardinalité $(1, m + n - 1)$.

L'utilisation d'interactions de cardinalité $(1, n)$ et (m, n) est principalement liée à des problèmes de granularité de la représentation des connaissance des entités. Nous faisons le choix de restreindre la cardinalité des interactions pouvant être spécifiées aux interactions dégénérées et individuelles. Ce choix ne restreint pas de manière significative les simulations pouvant être spécifiées avec notre approche. En effet, nous soutenons dans cette thèse que ces interactions peuvent dans la majorité des cas être décomposées en un ensemble d'interactions dégénérées et individuelles. Cette affirmation est justifiée dans la section 6.1 page 161 du chapitre 6.

3.2.4 Définition formelle d'une interaction

Comme nous l'avons mentionné précédemment, chaque entité ne joue pas la même fonction dans une interaction. Par conséquent, elles ne doivent pas implémenter les mêmes primitives. Nous caractérisons les primitives devant être implémentées par les entités participant à l'interaction par la notion de *signature d'une entité dans l'interaction.*

Signatures

La *signature d'une entité dans l'interaction* permet de caractériser les primitives d'action et de perception manipulées dans les préconditions, le déclencheur et les actions de l'interaction. Elle permet de plus d'identifier l'ensemble des primitives abstraites qu'une entité doit spécifier à l'aide d'algorithmes pour participer à une interaction. La définition qui suit est illustrée dans la figure 3.4.

Définition 9. ***Signature d'une entité dans une interaction***
La **signature** S **d'une entité dans une interaction** est définie par l'ensemble $S = \langle id, (p_i)_{i\in\mathbb{N}}\rangle$ où p_i est une primitive de perception ou d'action et id est un identifiant décrivant explicitement la fonction de l'entité dans l'interaction. On note par la suite $primitives(S)$ l'ensemble des primitives de la signature S et $id(S)$ l'identifiant de S.
On dit qu'une entité **se conforme** à une signature si elle fournit une spécification à chacune des primitives abstraites qu'elle contient.

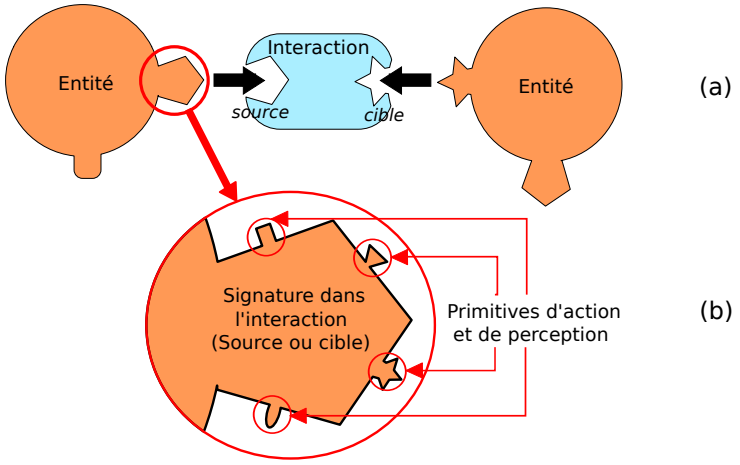


FIGURE 3.4 – Schéma illustrant le lien entre les primitives abstraites, les interactions et la signature d'une entité dans une interaction.
(a) Chaque entité (disque orange) se conforme à un ensemble de signatures (excroissances oranges). Symétriquement, chaque interaction (rectangle bleu) définit une signature d'entité à laquelle doit se conformer sa source (cavité en forme de pentagone) et une signature d'entité à laquelle doit se conformer sa cible (cavité en forme d'étoile). Afin de pouvoir être la source (resp. la cible) d'une interaction et donc pouvoir initier (resp. subir) l'interaction, une entité doit se conformer à la signature de la source (resp. de la cible) dans l'interaction. Par conséquent, une interaction entre une entité source et une entité cible n'est possible que si la source peut initier l'interaction et la cible peut la subir.
(b) Dans le détail, chaque signature décrit en fait un ensemble de primitives manipulées dans l'interaction. Elles doivent être spécifiées par toutes les entités se conformant à la signature.

La notion de signature permet alors de caractériser si une entité peut participer à une interaction. En effet, chaque interaction individuelle définit une signature d'entité à laquelle doit se conformer sa source ainsi qu'une signature d'entité à laquelle doit se conformer sa cible. On considère qu'une entité **peut initier** une interaction si elle se conforme à la **signature de la source** dans l'interaction. De même, une entité **peut subir** une interaction si elle se conforme à la **signature de la cible** dans l'interaction. Par conséquent, une **interaction peut survenir** entre deux entités si l'une d'elles est **capable d'initier l'interaction** et si l'autre entité est **capable de la subir**.

Le nombre de signatures à définir pour une interaction $\mathcal{I}$ est directement lié à la cardinalité de cette dernière. Il est égal à $card_S(\mathcal{I}) + card_T(\mathcal{I})$. Toutefois, plusieurs entités impliquées simultanément dans une interaction peuvent y jouer un rôle symétrique et doivent donc implémenter les mêmes primitives. On considère alors qu'elles ont la même signature (ce point est illustré dans l'exemple ci-après). La signature d'une entité dans une interaction peut aussi ne contenir aucune primitive. Dans ce cas, l'entité ne doit

implémenter aucune primitive abstraite particulière pour pouvoir participer à l'interaction.

Exemple.
Dans la figure 3.3b, l'interaction MANGER *est une interaction de cardinalité* $(1,1)$ *définissant par conséquent deux signatures :*

- *Une signature liée à l'entité* $\mathcal{X}$ *qui effectue l'interaction. Elle contient une primitive de perception* $\langle aFaim, booleen, \emptyset\rangle$ *et une primitive d'action* $\langle diminuerFaim, \emptyset, \{Entite\}\rangle$*. Par exemple :*

$$\langle sourceDeManger, \{\langle aFaim, booleen, \emptyset\rangle, \langle diminuerFaim, \emptyset, \{Entite\}\rangle\}\rangle$$

- *Une signature liée à l'entité* $\mathcal{Y}$ *qui subit l'interaction. Elle contient une primitive de perception* $\langle estSain, booleen, \emptyset\rangle$*. Par exemple :*

$$\langle cibleDeManger, \{\langle estSain, booleen, \emptyset\rangle\}\rangle$$

L'interaction TRANSPORTER À 4 *a pour cardinalité* $(4,1)$*, mais n'a pourtant à définir que deux signatures :*

- *Une signature liée à l'entité transportée (i.e. la cible de l'interaction) ;*
- *Une signature liée aux entités effectuant le transport (i.e. les sources de l'interaction). A priori, ces 4 entités jouent un rôle symétrique dans l'interaction et implémentent donc les mêmes primitives abstraites. Elles ont donc la même signature dans l'interaction.*

Le déclencheur, les préconditions et les actions d'une interaction peuvent aussi être amenés à manipuler des primitives de l'environnement. Par conséquent, une interaction est non seulement caractérisée par la signature des entités, mais aussi par la signature de l'environnement.

Définition 10. ***Signature de l'environnement dans une interaction***
La **signature de l'environnement dans une interaction** est définie par l'ensemble $\langle (p_i)_{i\in\mathbb{N}}\rangle$ où p_i est une primitive de l'environnement.

La signature de l'environnement a un rôle primordial au sein d'une bibliothèque d'interactions. En effet, elle caractérise la topologie de l'environnement dans lequel chaque interaction peut avoir lieu. **Une interaction ne peut avoir lieu dans un environnement que si ce dernier se conforme à la signature de l'environnement dans l'interaction.**

Définition d'une interaction

Pour des raisons pratiques liées à la spécification des préconditions, du déclencheur et des actions d'une interaction, nous associons un nom à chaque entité participant à l'interaction. Nous associons de plus à chaque nom la signature de l'entité lui correspondant dans l'interaction ainsi qu'une valeur dans $\{Source, Cible\}$ qui détermine si cette entité est une source ou une cible de l'interaction.

Nous définissons une interaction dans IODA de la manière suivante, dont un métamodèle est fourni dans la figure 3.5 :

Définition 11. ***Interaction***
Une **interaction** est un bloc sémantique décrivant comment et sous quelles conditions des entités peuvent interagir ensemble.
Il s'agit d'un 8-uplet $\mathcal{I} = \langle id, noms, cardinalite, signature, sign_{env}, preconditions, declencheur, actions \rangle$ tel que :
- *id* est l'identifiant de l'interaction représenté sous la forme d'une chaîne de caractères. Il exprime de manière explicite ce à quoi correspond l'interaction. Cet identifiant est supposé être unique ;
- *noms* est un ensemble de chaînes de caractères représentant le nom attribué à chaque entité participant à l'interaction ;
- *cardinalite* est une fonction prenant en paramètre un nom et retournant une valeur dans $\{Source, Cible\}$. Cette fonction détermine si l'entité caractérisée par un nom est une source ou une cible de l'interaction. Par conséquent, $card_S(\mathcal{I}) = |cardinalite(nom) = Source|_{\forall nom}$ et $card_T(\mathcal{I}) = |cardinalite(nom) = Cible|_{\forall nom}$
- *signature* est une fonction associant à chaque élément de *noms* la signature d'entité qui lui correspond ;
- $sign_{env}$ est la signature de l'environnement dans l'interaction. Elle définit les primitives devant être fournies par l'environnement pour que l'interaction puisse y avoir lieu ;
- *preconditions* décrit les préconditions de l'interaction, *i.e.* les pré-requis physiques ou logiques nécessaires à l'exécution de l'interaction ;
- *declencheur* décrit le déclencheur de l'interaction, *i.e.* les buts implicites ou explicites auxquels l'interaction répond ;
- *actions* décrit les actions, *i.e.* les effets de l'interaction.

Notation 2.
On pose $\mathbb{I}_{(1,1)}$ l'ensemble des interactions individuelles d'une simulation et $\mathbb{I}_{(1,0)}$ l'ensemble de ses interactions dégénérées.

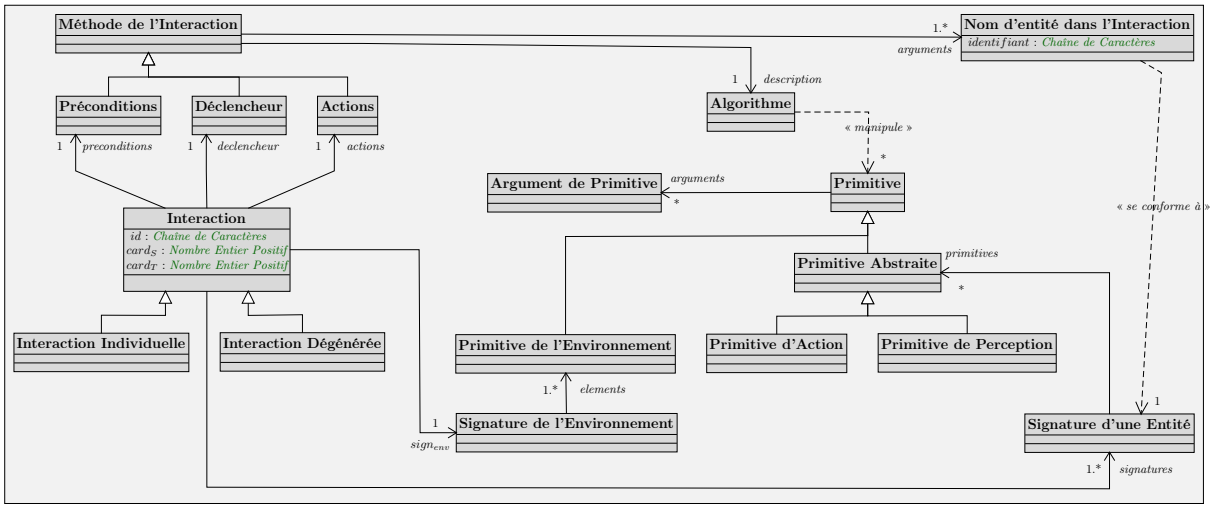


FIGURE 3.5 – Métamodèle général d'une interaction dans le modèle IODA

Dans ce chapitre, nous nous plaçons uniquement dans le cas d'interactions dont la cardinalité est soit $(1, 1)$, soit $(1, 0)$. Par conséquent, nous désignons les entités par les noms *source* et *cible* dans l'interaction, dont les signatures respectives sont $sign_{source}(\mathcal{I})$ et $sign_{cible}(\mathcal{I})$.

Préconditions, actions et déclencheur

Les préconditions, actions et déclencheur d'une interactions sont exprimées par des algorithmes manipulant les entités de l'interaction au travers de primitives. L'appel à une primitive particulière d'une

entité dans les préconditions, le déclencheur ou les actions de l'interaction se fait au travers du nom de l'entité dans l'interaction et de l'identifiant de la primitive appelée. La description de ces appels se fait avec une syntaxe particulière, que nous introduisons dans la notation qui suit.

Notation 3.
Dans les algorithmes décrivant les préconditions, le déclencheur ou les actions d'une interaction, l'appel à une primitive dont l'identifiant est "identifiant_primitive" d'une entité dont le nom est "nom_entité" dans l'interaction se fait avec la syntaxe :

nom_entité**.**identifiant_primitive(...)

Dans cette expression "..." est à remplacer par les paramètres utilisés pour cette primitive.

L'appel à une primitive de l'environnement se fait de la même manière en utilisant le nom "Environnement".

***Exemple. Syntaxe de la description de l'interaction* Manger**
Considérons l'interaction Manger *telle que décrite dans la figure 3.3 page 71. Cette interaction a pour cardinalité* $(1, 1)$ *et implique donc deux entités nommées respectivement « Source » et « Cible ». La description des préconditions, déclencheur et actions de cette interaction est alors :*

Déclencheur	*Source.aFaim()*
Préconditions	*Cible.estSain()*
Actions	*Source.diminuerFaim(Cible)* *Environnement.retirer(Cible)*

3.3 Un approche centrée sur les interactions

Afin de déterminer les interactions que les entités sont capables d'initier ou de subir et donc définir le graphe des interactions entre entités, l'approche Ioda repose sur une *matrice d'interaction brute* (voir figure 3.6). Chaque ligne de cette matrice représente les interactions qu'une entité peut initier (*i.e.* les interactions pour lesquelles l'entité est la source) et chaque colonne les interactions qu'une entité peut subir (*i.e.* les interactions pour lesquelles l'entité est la cible). Par conséquent, l'intersection d'une ligne et d'une colonne définit les interactions que deux entités sont capables d'effectuer ensemble.

Source \ Cible	∅	Herbe	Herbivore	Carnivore
Herbe	Pousser			
Herbivore	Mourir Se Déplacer	(Manger, d=0cm)	(Se Reproduire, d=30cm)	
Carnivore	Mourir Se Déplacer		(Manger, d=30cm)	(Se Reproduire, d=30cm)

Figure 3.6 – Structure d'une matrice d'interaction brute, illustrée sur l'exemple de la simulation d'un écosystème contenant proies et prédateurs. L'élément *(*Manger*, d=0cm)* situé à l'intersection de la ligne *Herbivore* et de la colonne *Herbe* se lit « Une entité Herbivore a la capacité d'initier l'interaction Manger avec une entité Herbe pour cible si cette dernière se situe au plus à une distance de 0cm de la source ».

La définition formelle de la matrice d'interaction repose sur trois notions différentes : la *garde de distance*, les *éléments d'assignation* ainsi que les *assignations*.

3.3.1 Garde de distance

Une interaction individuelle ne peut avoir lieu entre deux entités que si elles sont suffisamment *proches*. La notion de proximité n'est pas propre aux simulations se déroulant dans un espace euclidien en deux ou trois dimensions : elle peut être exprimée dans tout type d'espace topologique pouvant exprimer une

distance telle qu'une différence de richesse entre deux entités ou la distance entre deux nœuds dans un graphe.

Pour les mêmes raisons ayant conduit à l'utilisation de primitives abstraites dans les préconditions, les actions et le déclencheur d'une interactions, la garde de distance n'est pas propre à une interaction individuelle.

Exemple.
Un herbivore initie l'interaction MANGER *au contact de végétaux (i.e. à une distance de 0), alors qu'un caméléon peut initier cette interaction à une distance plus grande (de 10 à 70 cm).*

En conséquence, une *garde de distance* est associée à chaque interaction individuelle présente dans la matrice d'interaction.

Définition 12. ***Garde de distance***
La **garde de distance** d'une interaction individuelle dans la matrice d'interaction brute est un entier correspondant à la distance maximale en deçà de laquelle l'interaction peut être initiée.

La garde de distance est spécifiée indépendamment de la description de l'interaction, afin de garantir un plus grand polymorphisme. Elle est définie lors de l'assignation de l'interaction à des entités, lorsque la *matrice d'interaction brute* est construite.

3.3.2 Matrice d'interaction brute

La *matrice d'interaction brute* exprime l'ensemble des interactions pouvant avoir lieu entre les entités d'une simulation en fonction de la famille à laquelle elles appartiennent. Elle constitue ainsi une représentation synthétique du graphe liant les entités interagissant ensemble. Nous appelons *assignation* chaque cellule de cette matrice. Une assignation décrit l'ensemble des interactions pouvant être initiées par une entité source avec une entité cible particulières. Ces assignations contiennent chacune des *éléments d'assignation*. Un élément d'assignation représente une interaction qu'une entité source a la capacité d'initier avec une entité cible.

Définition 13. ***Élément d'assignation***
Un **élément d'assignation** est la représentation d'une interaction qu'une entité source est capable d'initier. Il peut représenter :
- une interaction individuelle $\mathcal{I}$ qu'une entité source de la famille $\mathcal{F}_S$ est capable d'initier avec une entité cible de la famille $\mathcal{F}_T$. Cette interaction ne peut être initiée que si la distance séparant ces entités est inférieure ou égale à la **garde de distance**. On parle alors d'**élément d'assignation individuel**. Il est décrit sous la forme du n-uplet $(\mathcal{I}, dist, \mathcal{F}_S, \mathcal{F}_T) \in \mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}$ où $dist$ est une garde de distance ;
- une interaction dégénérée $\mathcal{I}$ qu'une instance de la famille d'entités $\mathcal{F}_S$, est capable d'initier. On parle alors d'**élément d'assignation dégénéré**, décrit sous la forme du n-uplet $(\mathcal{I}, \mathcal{F}_S) \in \mathbb{I}_{(1,0)} \times \mathbb{F}$.

Notation 4.
On pose $\mathcal{I}(a)$ la fonction permettant de connaître l'interaction associée à un élément d'assignation, $dist(a)$ la fonction permettant d'en connaître la garde de distance, $source(a)$ la fonction permettant d'en connaître la famille d'entités source et *cible* la fonction permettant d'en connaître la famille d'entités cible.

Une *assignation* représente l'ensemble des interactions qu'une instance d'une famille d'entités peut initier avec une instance d'une famille d'entités cible. Par conséquent, une assignation constitue une cellule de la matrice d'interaction brute.

Définition 14. ***Assignation individuelle***
Une **assignation individuelle** $a_{\mathcal{S}/\mathcal{T}}$ est un ensemble d'éléments d'assignation individuels. Elle représente l'ensemble des interactions individuelles qu'une instance de la famille d'entités $\mathcal{S}$ peut initier en tant que source conjointement avec une instance de la famille d'entités $\mathcal{T}$ en tant que cible. Plus formellement, on a :

$$\begin{cases} \forall \mathcal{S} \in \mathbb{F}, \forall \mathcal{T} \in \mathbb{F}, a_{\mathcal{S}/\mathcal{T}} \subset \big(\mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}\big) \\ \forall \mathcal{S} \in \mathbb{F}, \forall \mathcal{T} \in \mathbb{F}, a \in a_{\mathcal{S}/\mathcal{T}} \Leftrightarrow source(a) = \mathcal{S} \wedge cible(a) = \mathcal{T} \end{cases}$$

D'après la définition précédente, on sait qu'une entité a la capacité d'initier une interaction avec une autre entité comme cible s'il existe une cellule de la matrice d'interaction brute (*i.e.* une assignation) dans laquelle un élément d'assignation contient cette interaction. On a donc la propriété qui suit.

Propriété 1.
Soient $e \in \mathbb{E}, e' \in \mathbb{E}$ et $\mathcal{I} \in \mathbb{I}_{(1,1)}$

$$e \text{ a la capacité d'initier } \mathcal{I} \text{ avec } e' \text{ pour cible} \Leftrightarrow$$
$$\exists \mathcal{S} \in \mathbb{F}, \exists \mathcal{T} \in \mathbb{F}, \exists a \in a_{\mathcal{S}/\mathcal{T}} | e \prec \mathcal{S} \wedge e' \prec \mathcal{T} \wedge \mathcal{I}(a) = \mathcal{I}$$

Ces définitions peuvent aussi être appliquées au cas des interactions dégénérées.

Définition 15. ***Assignation dégénérée***
Une **assignation dégénérée** $a_{\mathcal{S}/\emptyset}$ est un ensemble d'éléments d'assignation dégénérés. Elle représente l'ensemble des interactions dégénérées qu'une instance de la famille d'entités $\mathcal{S}$ peut initier. Plus formellement, on a :

$$\begin{cases} \forall \mathcal{S} \in \mathbb{F}, a_{\mathcal{S}/\emptyset} \subset \mathbb{I}_{(1,0)} \times \mathbb{F} \\ \forall \mathcal{S} \in \mathbb{F}, a \in a_{\mathcal{S}/\emptyset} \Leftrightarrow source(a) = S \end{cases}$$

Propriété 2.
Soient $e \in \mathbb{E}$ et $\mathcal{I} \in \mathbb{I}_{(1,0)}$

$$e \text{ a la capacité d'initier } \mathcal{I} \Leftrightarrow \exists \mathcal{S} \in \mathbb{F}, \exists a \in a_{\mathcal{S}/\emptyset} | e \prec \mathcal{S} \wedge \mathcal{I}(a) = \mathcal{I}$$

On appelle alors *matrice d'interaction brute* l'ensemble de toutes les assignations pour une simulation. Le métamodèle issu de la définition qui suit est présenté sur la figure 3.7.

Définition 16. ***Matrice d'interaction brute***
Une **matrice d'interaction brute** $\mathcal{M}$ représente toutes les interactions pouvant survenir entre les différentes entités que contient l'environnement. Elle constitue l'agrégation de toutes les assignations pour la simulation. Plus formellement, on a :

$$\mathcal{M} : \begin{array}{lcl} \mathbb{F} \times \big(\mathbb{F} \cup \{\emptyset\}\big) & \rightarrow & \mathcal{P}\Big(\big(\mathbb{I}_{(1,0)} \times \mathbb{F}\big) \cup \big(\mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}\big)\Big) \\ (\mathcal{S}, \mathcal{T}) & \rightarrow & a_{\mathcal{S}/\mathcal{T}} \end{array}$$

Une illustration d'une telle matrice est présentée sur la figure 3.6 pour une simulation d'un écosystème contenant de l'herbe, des herbivores et des carnivores. Cet exemple constitue un modèle simple de type proie/prédateur où de l'herbe pousse, des herbivores se déplacent dans l'environnement, se reproduisent avec d'autres herbivores, mangent de l'herbe et meurent s'ils n'ont pas assez mangé. De plus des carnivores peuvent s'y déplacer, s'y reproduire avec d'autres carnivores, y manger des herbivores et y mourir s'ils n'ont pas assez mangé.

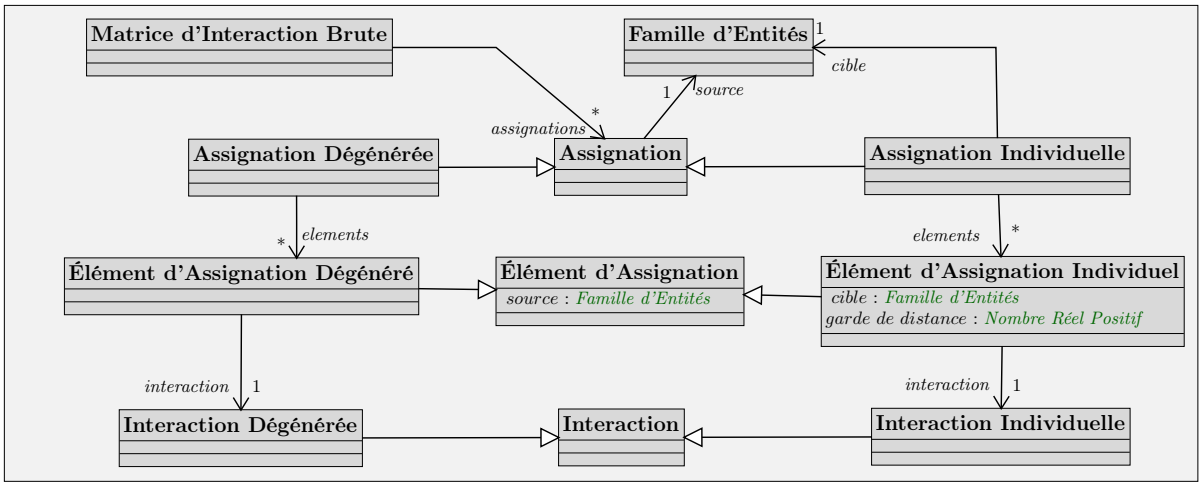


FIGURE 3.7 – Métamodèle d'une matrice d'interaction brute dans le modèle IODA

3.4 Les entités

Dans IODA, les entités d'une simulation sont représentées de manière homogène et leur comportement est régi par un processus générique schématisé dans la figure 3.8. Cette section s'articule selon les trois éléments figurant dans cette dernière : la mise à jour de l'état de l'entité, la perception des autres entités et la sélection de l'interaction initiée. Une version détaillée des algorithmes correspondant à cette figure sont décrits dans la section 4.3 page 119 du chapitre 4.

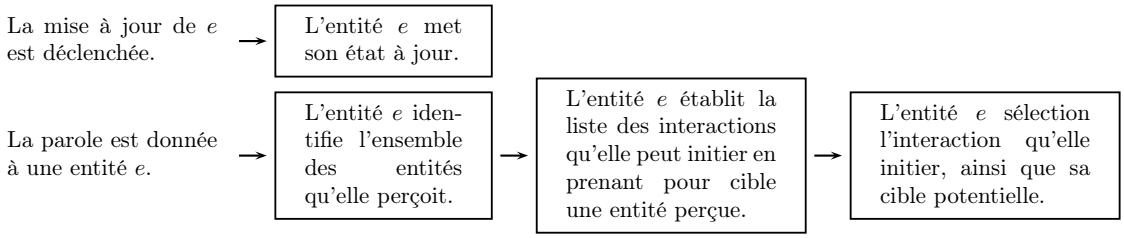


FIGURE 3.8 – Principes régissant l'évolution et le comportement d'une entité dans l'approche IODA. Ces principes se décomposent en deux parties. La première (partie supérieure) consiste à mettre à jour l'état de l'entité indépendamment des interactions qu'elle initie ou qu'elle subit. La seconde (partie inférieure) décrit le comportement de l'entité. Il consiste à recenser les entités lui étant voisines, identifier les interactions pouvant être initiées avec une entité voisine et enfin sélectionner puis initier une interaction parmi ces dernières.

3.4.1 Mise à jour

Une entité n'a pas besoin d'initier ou subir une interaction pour que son état évolue. Certaines caractéristiques peuvent évoluer en dehors de tout comportement. C'est par exemple le cas de l'âge, puisqu'une entité vieillit indépendamment de toute action qu'elle entreprend. Il en va de même pour la sensation de satiété dans une simulation de type proie/prédateur qui diminue progressivement au fil de la simulation.

Puisque la matrice d'interaction brute a pour rôle de décrire ce que les entités sont capables de faire, la mise à jour des entités ne peut y être exprimée. Nous choisissons toutefois d'exprimer la mise à jour par un moyen similaire que nous appelons *matrice de mise à jour*, afin d'unifier la représentation des connaissances des entités. Ce choix permet de plus de profiter des avantages de notre formalisme autant dans la description du comportement des entités que dans la description de leur mise à jour. En effet, les

éléments figurant dans la matrice de mise à jour sont représentés de la même manière que les éléments d'assignation dégénérés et donc à l'aide d'interactions dégénérées.

Définition 17. ***Matrice de mise à jour***
La **Matrice de mise à jour** $\mathcal{U}$ exprime quelles actions sont exécutées afin de mettre à jour l'état des entités d'une simulation. Elle est constituée d'assignations dégénérées. Plus formellement, si $\mathcal{P}$ est l'ensemble des partitions d'un ensemble, on a :

$$\mathcal{U} : \begin{array}{ccc} \mathbb{F} & \rightarrow & \mathcal{P}(\mathbb{I}_{(1,0)} \times \mathbb{F}) \\ \mathcal{S} & \rightarrow & (u^i_{\mathcal{S}}, \mathcal{S})_{i \in \mathbb{N}} \end{array}$$

La mise à jour d'une entité consiste à initier l'ensemble des interactions dégénérées qui lui correspondent dans la matrice de mise à jour. Par abus de langage, nous désignons par « interactions de mise à jour » les interactions dégénérées présentes dans la matrice d'interaction. L'avantage de cette matrice est de rendre la déclaration de la mise à jour des agents à la fois modulaire et facile à modifier.

L'ordre dans lequel les interactions de mise à jour sont initiées est important. En effet, ces interactions peuvent avoir des effets différents selon l'ordre dans lequel elles sont initiées.

Exemple.
Considérons une simulation d'un écosystème de type proie/prédateur où le comportement de prédateurs est basé sur leur propre santé, représentée par une quantité d'énergie (un nombre entier). Lorsque cette dernière passe sous un certain seuil, l'entité ressent la sensation de faim, ce qui l'amène à déclencher l'interaction Manger *développée dans les sections précédentes. Dans cet exemple, nous considérons que l'effet de l'interaction* Manger *n'est pas immédiat : l'énergie de l'entité augmente progressivement de manière naturelle au fil de sa digestion. De plus, dans tous les cas, la santé d'une entité décroit de manière inversement proportionnelle à la quantité d'énergie de l'entité. Dans ce cadre, deux interactions de mise à jour sont définies :*

- Diminuer Satiété*, qui consiste a diminuer la sensation de satiété de l'entité. Dans cet exemple, cette interaction consiste à retrancher à l'énergie de l'entité un montant inversement proportionnel à sa quantité d'énergie actuelle ;*
- Digérer*, qui consiste à poursuivre l'assimilation des éléments ingérés par l'entité. Dans cet exemple, cette interaction consiste à ajouter à l'énergie de l'entité une valeur dépendant des entités ayant été ingérées.*

Il apparaît alors évident que la valeur de l'énergie de l'entité sera différente selon l'ordre d'exécution des interactions de mise à jour Diminuer Santé *et* Digérer *: si x est la quantité d'énergie de l'entité, m la quantité d'énergie ajoutée par l'interaction* Digérer *et k la proportion d'énergie perdue lors de l'interaction* Diminuer Santé*, alors la quantité d'énergie de l'entité à la fin de la mise à jour sera dans un cas $x + m - \frac{k}{x}$, et dans l'autre $x + m - \frac{k}{x+m}$.*

Puisque l'ordre d'exécution des interactions de mise à jour a une influence sur les résultats obtenus, l'aspect déclaratif de la matrice de mise à jour doit être complété afin d'éviter tout biais lors de l'implémentation. Pour remédier à ce problème, nous choisissons d'établir une relation d'ordre entre les interactions présentes dans la matrice de mise à jour. Cet ordre est obtenu en attribuant une priorité à chaque élément présent dans la matrice de mise à jour, aboutissant à la description d'une *matrice de mise à jour ordonnée.*

Définition 18. ***Matrice de mise à jour ordonnée***
La **Matrice de mise à jour ordonnée** $\mathcal{U}_{ord}$ exprime comment a lieu la mise à jour des entités d'une simulation. Elle consiste à attribuer à chaque élément de la matrice de mise à jour une priorité représentée par un nombre entier. Plus ce nombre est élevé, plus l'interaction de mise à jour est prioritaire et est exécutée tôt lors de la mise à jour. Plus formellement, si $\mathcal{P}$ est l'ensemble des partitions d'un ensemble, on a :

$$\mathcal{U}_{ord} : \begin{array}{lll} \mathbb{I}_{(1,0)} \times \mathbb{F} & \rightarrow & \mathbb{Z} \\ (u) & \rightarrow & \begin{cases} \mathcal{U}_{ord}(u) & \text{si } u \in \mathcal{U}(\mathcal{S}) \\ \text{non défini} & \text{sinon} \end{cases} \end{array}$$

On note $\mathcal{U}_{ord}^{-1}$ l'application permettant de connaître l'ensemble des interactions de mise à jour ayant une priorité donnée pour une famille d'entités $\mathcal{F}$:

$$\mathcal{U}_{ord}^{-1} : \begin{array}{lll} \mathbb{F} \times \mathbb{Z} & \rightarrow & \mathcal{P}(\mathbb{I}_{(1,0)} \times \mathbb{F}) \\ (\mathcal{F}, p) & \rightarrow & \{u \in \mathcal{U}(\mathcal{F}) | \mathcal{U}_{ord}(u) = p\} \end{array}$$

Un métamodèle de la matrice de mise à jour ordonnée est proposé dans la figure 3.9.

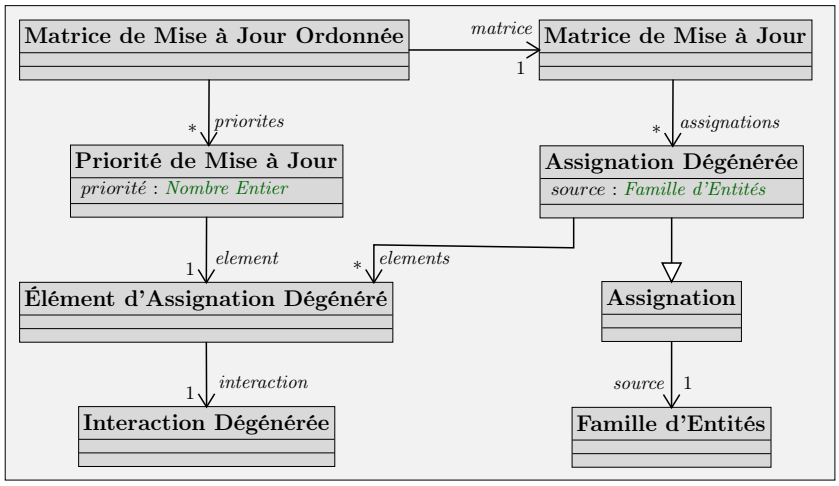


FIGURE 3.9 – Métamodèle d'une matrice de mise à jour ordonnée dans le modèle IODA

3.4.2 Perception

L'un des principes de base des systèmes multi-agents est que les entités ne sont pas omniscientes et n'interagissent qu'avec les entités qu'elles perçoivent. Il en va de même dans IODA, où la perception d'une entité est fondée sur le concept de *halo de perception*.

Le halo de perception d'une famille d'entités $\mathcal{F} \in \mathbb{F}$ est une fonction qui détermine si une entité $e' \in \mathbb{E}$ est perçue par une entité $e \prec \mathcal{F}$. Tout comme le déclencheur, les préconditions et les actions d'une entité, le halo est décrit sous la forme d'un algorithme manipulant à la fois des primitives de l'environnement et des primitives abstraites des entités. Par conséquent une famille d'entités a une signature pour son halo. Il en va de même pour l'environnement. Ces informations aboutissent à la définition du halo qui suit.

Définition 19. ***Halo de perception d'une famille entité***
Le **Halo de perception** $\mathcal{H}(\mathcal{F})$ d'une famille d'entités $\mathcal{F} \in \mathbb{F}$ permet de déterminer si une entité e' est perçue par une entité $e \prec \mathcal{F}$. Il est représenté par un triplet $\langle halo, sign_{entite}, sign_{env} \rangle$ où :

- $halo : \begin{array}{rcl} \mathbb{E} \times \mathbb{E} & \rightarrow & \{vrai, faux\} \\ (e, e') & \rightarrow & \begin{cases} vrai & \text{si } e' \text{ est perçue par } e \\ faux & \text{si } e' \text{ n'est pas perçue par } e \end{cases} \end{array}$
- $sign_{entite}$ est la signature de la famille d'entités pour son halo. Elle détermine l'ensemble des primitives abstraites que la famille d'entités doit spécifier pour utiliser son propre halo ;
- $sign_{env}$ est la signature de l'environnement pour le halo de la famille d'entités $\mathcal{F}$. Elle détermine l'ensemble des primitives de l'environnement devant être spécifiées pour que le halo $\mathcal{H}(\mathcal{F})$ puisse être utilisé.

Exemple. Halo manipulant des primitives de perception
Soit un halo désignant une entité e_2 comme perçue par e si elle figure dans la mémoire de l'entité e ou si elle si situe à une distance inférieure ou égale à l'acuité visuelle e. Il s'exprime sous la forme suivante :

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

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

Exemple. Halo ajoutant une primitive de l'environnement
Soit un halo désignant une entité e_2 comme perçue par e si elle ne se situe pas derrière une autre entité de la simulation. Il s'exprime sous la forme suivante :

halo$\Big(\mathcal{H}\big($**famille(e)**$\big)\Big)(e,e_2)$ *:*
| **retourner** $environnement.aucuneEntiteOpaqueEntre(e, e_2)$

Il implique l'ajout d'une primitive de l'environnement qui vérifie qu'aucune entité opaque ne se situe entre e et e_2 $\langle aucuneEntiteOpaqueEntre, booleen, \{Entite, Entite\} \rangle$. Si cette primitive retourne faux, on peut considérer que e_2 se situe derrière une autre entité.

Le voisinage d'une entité $e \in \mathbb{E}$ constitue l'ensemble des entités perçues par e. Il représente l'ensemble des entités avec lesquelles e est susceptible d'interagir et est construit à l'aide du halo de la famille de e.

Définition 20. ***Voisinage d'une entité***
Le **Voisinage** $\mathcal{V}(e)$ d'une entité e correspond à l'ensemble des entités perçues par e. Plus formellement, on a :

$$\mathcal{V} : \begin{array}{rcl} \mathbb{E} & \rightarrow & \mathcal{P}(\mathbb{E}) \\ e & \rightarrow & \{e' \in \mathbb{E} | \mathcal{H}\big(famille(e)\big)(e') == vrai \wedge e \neq e'\} \end{array}$$

Il est important de noter que le halo ne se limite pas à une proximité spatiale : une entité mémorisée peut aussi faire partie du voisinage d'une entité.

3.4.3 Sélection de l'interaction initiée

La sélection de l'interaction initiée par une entité est un processus consistant à choisir selon des modalités variables un couple « interaction individuelle/entité du voisinage » ou une interaction dégénérée afin d'en initier les actions. Les éléments déclaratifs permettant de décrire un tel processus dépendent grandement de la nature du processus décisionnel de l'entité. Le modèle général de Ioda n'émet aucune hypothèse sur ce processus, qui peut autant être à planification délibérative, à planification réactive, purement réactive, *etc.*

Pour des raisons que nous évoquons ultérieurement dans la section 3.6, l'approche Ioda se concentre sur la spécification d'entités dont le **processus de sélection d'interaction est réactif**. Le modèle permettant de décrire de tels comportements est présenté dans la section 3.6.2. **Ce processus ne présume**

pas du degré de cognition utilisé pour exprimer les primitives abstraites des entités, qui peuvent autant être réactives que cognitives.

Exemple. Expression de différents degrés de cognition dans une primitive
Dans une supermarché virtuel, un `Client` *ne peut initier une interaction* RAMASSER *prenant pour cible une entité de famille* `Article` *que si la source a envie de ramasser la cible. Cela se traduit dans l'interaction par l'usage d'une primitive abstraite* $\langle estInteresse, booleen, \{Entite\}\rangle$ *dans le déclencheur de l'interaction.*

Différents degrés de cognition peuvent alors être exprimés :

- *Une entité purement réactive posséderait par exemple une liste de courses. La primitive* $S.estInteresse(T)$ *retournerait alors la valeur «* $T \in S.listeCourses$ *».*
- *Une entité plus cognitive se baserait par exemple sur de l'inférence, des croyances ou des connaissances. Dans ce cas, la primitive* $S.estInteresse(T)$ *pourrait retourner la valeur «* $S.sait("T \in S.listeCourses")$ *ou* $S.croit("UTILE(T)")$ *ou* $(S.sait("AIME(T)")$ *et non* $S.croit("GRAS(T)"))$ *», avec :*
  - $\langle sait, booleen, \{Chaîne\ de\ Caracteres\}\rangle$ *une primitive définissant si le prédicat donné en paramètres fait partie des connaissances de l'entité ;*
  - $\langle croit, booleen, \{Chaîne\ de\ Caracteres\}\rangle$ *une primitive définissant si le prédicat donné en paramètres fait partie des croyances de l'entité.*

Le **polymorphisme permet de spécifier des entités aux capacités de cognition variables à l'aide d'un seul et même formalisme**. Les interactions peuvent ainsi être **spécifiées indépendamment du degré de cognition des entités**. Cette propriété contribue à l'aspect transversal de l'approche IODA, puisque les interactions peuvent être spécifiées sans connaître la nature réactive ou cognitive des agents.

3.4.4 Définition formelle d'une famille d'entités

Lorsqu'une entité a la capacité de participer à une interaction, elle doit se conformer à une des signatures d'entité de cette interaction et par conséquent spécifier les primitives abstraites y figurant. Cette spécification se fait à l'aide d'algorithmes dont nous ne contraignons pas l'expression dans ce modèle formel. À l'instar des préconditions, du déclencheur et des actions d'une interaction, ce choix est fait afin de ne pas contraindre l'ingénierie logicielle dans leur expression. On peut toutefois constater qu'en règle générale ces primitives manipulent l'état des entités, que nous caractérisons dans IODA par un ensemble d'*attributs*.

Le métamodèle d'une famille et d'une instance d'entité est fourni dans la figure 3.10.

Définition 21. ***Famille d'entités***
Une **Famille d'entités** $\mathcal{F}$ est une spécification abstraite d'un ensemble d'instances d'entités qui partagent toutes leurs primitives abstraites, capacité à interagir ainsi que leur comportement. Elle est caractérisée par un 8-uplet $\langle id, attributs, primitives, \mathcal{H}, \mathcal{U}_{ord}, ligne, colonne, selection\rangle$ où :
- id est l'identifiant de la famille d'entités représenté sous la forme d'une chaîne de caractères. Il exprime de manière explicite ce à quoi correspondent les entités instances de cette famille ;
- $attributs = \left(a_i\right)_{i\in\mathbb{N}}$ est l'ensemble des identifiants de chaque **attribut** de cette famille d'entités. Chaque élément a_i est représenté sous la forme d'une chaîne de caractères exprimant de manière explicite le rôle de cet attribut dans le fonctionnement interne de l'entité ;
- $primitives$ est l'ensemble des primitives abstraites spécifiées par l'entité. Elles sont spécifiées sous la forme d'algorithmes pouvant manipuler les attributs cette famille ;
- $\mathcal{H}$ est le halo des entités de cette famille. Il leur permet de percevoir les autres entités de l'environnement ;
- $\mathcal{U}_{ord}$ est l'ensemble ordonné des interactions de mise à jour des entités de cette famille. Il permet de mettre à jour l'état de l'entité sans que cela soit le fruit de leur comportement ;
- $ligne$ est la fonction permettant de connaître la ligne de la matrice d'interaction brute associée à la famille d'entités $\mathcal{F}$. Elle décrit les interactions que toute entité de cette famille est capable d'initier. On a :
$$ligne(\mathcal{F}) : \begin{array}{ccc} \mathbb{F}\cup\{\emptyset\} & \rightarrow & \mathcal{P}\Big((\mathbb{I}_{(1,0)}\times\mathbb{F})\cup(\mathbb{I}_{(1,1)}\times\mathbb{R}^+\times\mathbb{F}\times\mathbb{F})\Big) \\ (\mathcal{T}) & \rightarrow & \mathcal{M}(\mathcal{F},\mathcal{T}) \end{array}$$
- $colonne$ est la fonction permettant de connaître la colonne de la matrice d'interaction brute associée à la famille d'entités $\mathcal{F}$. Elle décrit les interactions que toute entité de cette famille est capable de subir. On a :
$$colonne(\mathcal{F}) : \begin{array}{ccc} \mathbb{F} & \rightarrow & \mathcal{P}\Big(\mathbb{I}_{(1,1)}\times\mathbb{R}^+\times\mathbb{F}\times\mathbb{F}\Big) \\ (\mathcal{T}) & \rightarrow & \mathcal{M}(\mathcal{T},\mathcal{F}) \end{array}$$
- $selection$ est le modèle du processus utilisé afin de sélectionner les interactions initiées selon des modalités dépendant de la nature réactive/cognitive de l'entité.

La différence fondamentale existant entre famille d'entités et instance d'entités est qu'une famille d'entités n'associe aucune valeur à ses attributs et se contente de les identifier. Ainsi, une instance d'entité est caractérisée par tous les éléments définis dans une famille d'entités, auxquels s'ajoute une fonction permettant de connaître la valuation des attributs :

Définition 22. ***Instance d'entité***
Une **instance d'entité** $e \in \mathbb{E}$ correspond à une entité située dans l'environnement de la simulation. Elle est représentée par un couple $\langle famille(e), valeur_{attr}(e)\rangle$ tel que :
- $famille(e) \in \mathbb{F}$ est la famille de l'entité e ;
- $valeur_{attr}(e)$ est une fonction associant une valeur à chaque attribut défini dans la famille de e.

La valuation des attributs d'une entité doit être initialisée lors de la création de cette dernière. Une primitive particulière appelée *primitive d'initialisation* est définie dans ce but.

Définition 23. ***Primitive d'initialisation***
La **primitive d'initialisation** d'une famille d'entités $\mathcal{F} \in \mathbb{E}$ est une primitive décrivant comment initialiser la valuation des attributs d'une instance de cette famille d'entités. Elle a pour forme $\langle initialisation, \emptyset, \emptyset\rangle$.

Propriété 3.
Toute famille d'entités dispose d'une primitive d'initialisation :

$$\forall\mathcal{F}\in\mathcal{F}, \exists p_{init} = \langle initialisation, \emptyset, \emptyset\rangle | p_{init} \in primitives(\mathcal{F})$$

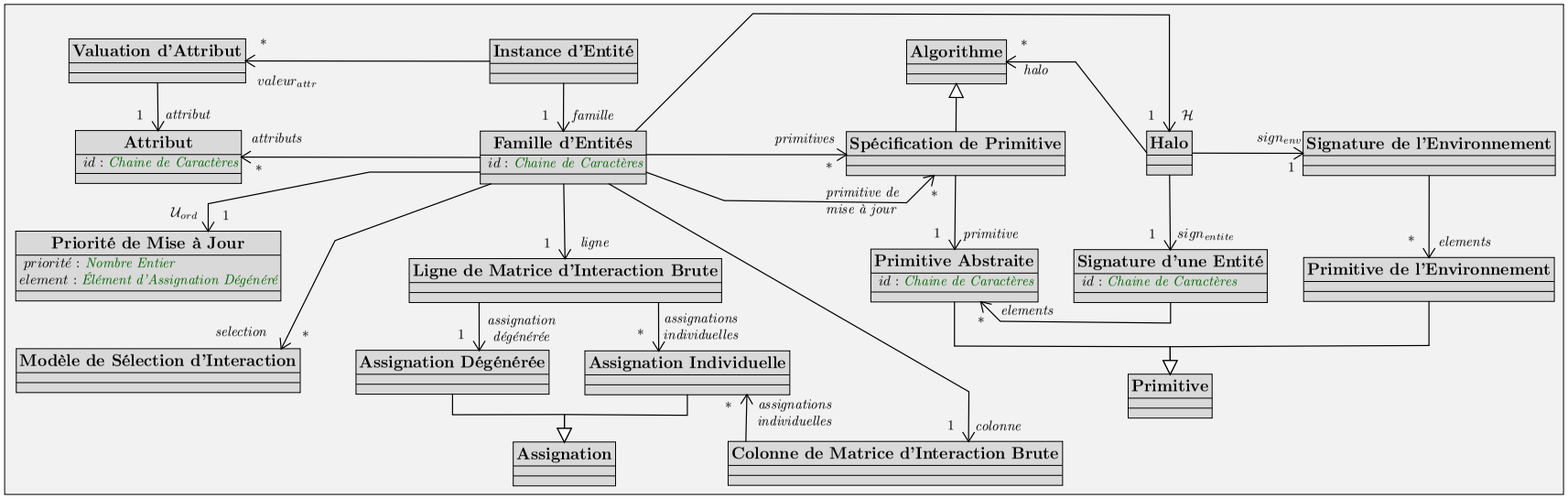


FIGURE 3.10 – Métamodèle d'une famille d'entité et d'une instance d'entité dans IODA.

3.5 L'environnement

D'après les définitions fournies dans la section 3.2.1, les interactions peuvent manipuler des primitives de l'environnement. Par conséquent, un environnement est défini par un ensemble de primitives.

Notation 5.
On note $primitives_{env}$ l'ensemble des primitives de l'environnement.

Le formalisme décrit dans ce chapitre est indépendant de la topologie de l'environnement Nous imposons seulement que l'environnement doive fournir une primitive permettant le calcul de la distance entre deux entités, des primitives permettant d'ajouter ou de retirer des entités ainsi qu'une primitive permettant de connaître l'ensemble des entités de la simulation. Cela se traduit par la propriété qui suit.

Propriété 4.
L'ensemble des primitives de l'environnement contient une primitive permettant de calculer la distance entre deux entités :

$$\langle distance, Nombre\ a\ Virgule\ Flottante, \{Entite, Entite\}\rangle \in primitives_{env}$$

Il contient de plus des primitives permettant d'ajouter ou de retirer des entités ainsi que de connaître l'ensemble des entités de la simulation. Ces primitives ne sont pas décrites ici, car leur forme est fortement dépendante de la topologie de l'environnement (voir exemples qui suivent).

Exemple. Ajout dans un environnement euclidien continu en deux dimensions
Dans le cadre d'une simulation en éthologie, l'environnement est usuellement représenté sous la forme d'un espace continu en deux dimensions dans lequel les entités sont assimilées à des points. Dans un tel environnement, l'ajout d'une entité se fait en précisant la position à laquelle elle est ajoutée. Par conséquent, la primitive de l'environnement ajoutant une entité prend la forme : $\langle ajouter, \emptyset, \{Entite, Nombre\ a\ Virgule\ Flottante, Nombre\ a\ Virgule\ Flottante\}\rangle$.

Exemple. Ajout dans un environnement discret
Dans le cadre de certaines simulations en sociologie, par exemple le modèle de ségrégation construit par Schelling [Sch71], l'environnement est représenté sous la forme d'un graphe. La position d'une entité dans l'environnement est alors un nœud du graphe. Par conséquent, la primitive de l'environnement ajoutant une entité prend la forme : $\langle ajouter, \emptyset, \{Entite, N\text{œ}ud\}\rangle$.

Ces primitives ne sont toutefois pas les seules que l'environnement ait à définir. En effet, chaque interaction a la capacité de percevoir des informations de l'environnement ou d'en modifier le contenu via des primitives de l'environnement. Par conséquent, une interaction ne peut être utilisée pour modéliser un phénomène que si elle est compatible avec l'environnement utilisé pour cette simulation, *i.e.* si l'environnement est conforme à la signature de l'environnement dans cette interaction. Ce critère permet de déterminer si une librairie d'interactions peut être utilisée pour modéliser un phénomène dans un environnement particulier. Cela se traduit par la propriété qui suit.

Propriété 5.
$\mathcal{I} \in \mathbb{I}_{(1,0)} \cup \mathbb{I}_{(1,1)}$ *peut etre utilisée pour modéliser un phénomène* $\Leftrightarrow sign_{env}(\mathcal{I}) \subseteq primitives_{env}$

Chaque primitive de l'environnement est spécifiée à l'aide d'algorithmes dont la structure est identique à celle de ceux spécifiant les préconditions, le déclencheur et les actions d'une interaction : une primitive de l'environnement peut manipuler d'autres primitives de l'environnement aussi bien que manipuler des primitives abstraites provenant des entités de la simulation. Ainsi, une primitive de l'environnement est caractérisée par une signature de l'environnement et une signature des entités de la simulation.

Définition 24. ***Primitive de l'environnement***
Une primitive de l'environnement est un triplet $p = \langle id, V, (a_i)_{i\in\mathbb{N}}, sign_{env}, sign_{entites}\rangle$ où :
- id est l'identifiant de la primitive représenté sous la forme d'une chaîne de caractères ;
- V est un élément optionnel décrivant le type de la valeur retournée par la primitive ;
- $(a_i)_{i\in\mathbb{N}}$ représente l'ensemble ordonné (éventuellement vide) des paramètres de la primitive ;
- $sign_{env}$ est la signature de l'environnement dans cette primitive, *i.e.* l'ensemble des primitives de l'environnement manipulées dans l'algorithme de cette primitive ;
- $sign_{entites}$ est la signature des entités dans cette primitive, *i.e.* l'ensemble des primitives abstraites des entités manipulées dans l'algorithme de cette primitive.

Exemple. Algorithme d'une distance en environnement euclidien
Dans le cadre de simulations en biochimie, l'environnement est souvent représenté sous la forme d'un espace continu en deux dimensions dans lequel évoluent des cellules. Dans de telles simulations, les cellules ont une épaisseur devant être prise en compte pour calculer la distance entre deux entités. La primitive correspondante peut alors avoir pour spécification :
distance(e_1,e_2) *:*

$scalaire \Leftarrow \Big(environnement.abscisse(e_1) - environnement.abscisse(e_2)\Big)^2$
$scalaire \Leftarrow scalaire + \Big(environnement.ordonnee(e_1) - environnement.ordonnee(e_2)\Big)^2$
retourner $\sqrt{scalaire} - environnement.epaisseur(e_1) - environnement.epaisseur(e_2)$

Dans cette primitive, on a :
- $sign_{env} = \left\{ \begin{array}{l} \langle abscisse, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle, \\ \langle ordonnee, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle, \\ \langle epaisseur, Nombre\ a\ Virgule\ Flottante, \{Entite\}\rangle \end{array} \right\}$
- $sign_{entites} = \emptyset$

Un environnement est aussi caractérisé par un ensemble d'attributs pouvant être manipulés au sein de ses primitives : la largeur d'un espace euclidien, le graphe des accointances entre entités, *etc.* Les attributs de l'environnement sont structurellement identiques aux attributs des entités : ils sont représentés par un identifiant et se voient associer une valeur qui évoluera au cours de la simulation.

En résumé, dans IODA, un environnement est caractérisé par la définition qui suit :

Définition 25. ***Environnement***
Un **environnement** est un espace métrique dans lequel figurent les agents. Il est caractérisé par les primitives de l'environnement qu'il spécifie et par les primitives abstraites que les entités doivent implémenter pour pouvoir y figurer. Il définit donc une **signature des entités** dans l'environnement. Plus formellement, un environnement est un tuple $\langle id, primitives_{env}, sign_{entites}, attributs, valeur_{attr}\rangle$ où :
- id est l'identifiant de l'environnement représenté sous la forme d'une chaîne de caractères. Cette chaîne exprime de manière explicite ce à quoi correspond l'environnement. Par exemple « environnement euclidien en trois dimensions » ;
- $primitives_{env}$ est l'ensemble des primitives de l'environnement spécifiées par cet environnement ;
- $sign_{entites}$ est l'ensemble des primitives abstraites devant être implémentées par toutes les entités de la simulation afin de pouvoir figurer dans cet environnement. On a donc $sign_{entites} = \bigcup_{p\in primitives_{env}} sign_{entites}(p)$;
- $attributs = \big(a_i\big)_{i\in\mathbb{N}}$ est l'ensemble des identifiants de chaque **attribut** de l'environnement. Chaque élément a_i est représenté sous la forme d'une chaîne de caractères, exprimant de manière explicite le rôle de cet attribut dans le fonctionnement interne de l'environnement ;
- $valeur_{attr}$ est une fonction associant une valeur à chaque identifiant d'attribut défini dans $attributs$.

Une entité ne peut être utilisée pour modéliser un phénomène que si elle est compatible avec l'environnement utilisé pour cette simulation, *i.e.* si elle implémente toutes les primitives abstraites manipulées

dans les primitives de l'environnement.

3.6 Modèle de comportements réactifs dans IODA

La simulation explicative consiste en particulier à appliquer le principe de parcimonie et donc à fournir le modèle le plus simple possible permettant d'expliquer l'apparition d'un phénomène. La description du de la sélection d'interaction n'y échappe pas. Dans un tel cadre, l'utilisation d'architectures à planification délibérative n'est pas la plus indiquée. En effet, un comportement basé sur des architectures réactives ou à planification réactive constitue dans un grand nombre de cas une approximation acceptable du comportement réel d'entités existant dans un phénomène, même lorsque ces dernières raisonnent pour produire leur comportement.

Dans cette thèse, nous nous focalisons donc sur la description d'un processus de sélection d'interaction réactif. L'approche IODA n'y est toutefois pas restreinte, puisque la structure des interaction permet leur utilisation dans tout type d'architecture fondée sur l'utilisation de règles. L'approche centrée sur les interaction développée par l'équipe SMAC pour le domaine d'application des jeux-vidéo [DMR05] en témoigne en définissant une architecture à planification délibérative reposant sur les mêmes principes fondamentaux que IODA.

3.6.1 Principes de la sélection réactive d'interaction

Dans sa forme la plus simple, un comportement réactif consiste à réagir systématiquement à chaque stimulus reçu. Autrement dit, une entité initie une interaction dès que ses conditions sont vérifiées. Cette approche ne permet de modéliser que des phénomènes physiques comme les réactions chimiques ou le déplacement de nuages de gaz. La spécification de comportement plus complexes, par exemple celui d'animaux en éthologie, nécessite un modèle comportemental plus raffiné. Ce raffinement consiste à donner aux entités la capacité de ne pas réagir aveuglément à chaque stimulus reçu. Pour cela, elles disposent de la capacité de choisir les interactions initiées parmi les interactions dont les conditions sont vérifiées.

Dans IODA un tel choix est exprimé à l'aide d'un ordre attribué aux interactions qu'une entité a la capacité d'initier. Cet ordre est exprimé de sorte qu'une interaction dont les conditions sont vérifiées subsumera l'initiation de toute interaction d'ordre moins élevé, considérées comme moins prioritaires.

Exemple. Ordre total entre deux interactions
Dans le cadre d'une simulation en éthologie, un animal peut donner à l'élément d'assignation individuel « MANGER une proie » une priorité supérieure à l'élément d'assignation dégénéré « SE DÉPLACER ». En effet, si une entité a le choix entre se déplacer et manger, elle choisira systématiquement de manger afin de réduire sa sensation de faim et donc éviter de mourir.

L'ordre permet ainsi d'obtenir une structure similaire à une instruction de type *if then... else ...* utilisée dans les plateformes de simulation ouvertes. Il permet de plus d'exprimer des préférences parmi les cibles d'une même interaction.

Exemple. Préférences entre deux cibles
Dans le cadre d'une simulation en éthologie, un prédateur peut donner à l'élément d'assignation individuel « MANGER une proie » une priorité supérieure à « MANGER une carcasse ». Ainsi, lorsqu'un prédateur a faim, il ne mangera une carcasse pour assurer sa survie que s'il ne peut pas manger une proie vivante à ce moment-là.

Une entité peut aussi ne pas exprimer de préférence entre la famille des cibles d'une même interaction ou même entre deux interactions différentes.

Exemple. Ordre partiel entre deux interactions
Dans le cadre d'une simulation en éthologie, un lion peut ne pas avoir de préférence :
- *entre les éléments d'assignation individuels « MANGER une antilope » et « MANGER une gazelle ». Cela signifie qu'un lion mange indistinctement des antilopes et des gazelles ;*

– *entre les éléments d'assignation individuels «* SE DÉPLACER VERS *une proie » et «* SE DÉPLACER VERS *un lac ». Cela signifie qu'un lion considère que se nourrir est aussi important que se désaltérer.*

Enfin, deux interactions peuvent avoir des conditions exclusives, auquel cas il n'est pas forcément utile d'établir d'ordre entre elles (voir figure 3.11).

Il est à noter que le dernier exemple ne peut être exprimé si l'ordre entre les éléments d'assignations est total. Afin de conserver la plus grande variété comportementale possible, nous considérons donc que cet ordre est partiel. Dans le cas où deux interactions sont les plus prioritaires, l'interaction initiée par une entité est choisie aléatoirement parmi elles.

3.6.2 Modèle réactif de sélection d'interaction

Dans IODA, l'ordre imposé entre les éléments d'assignation d'une matrice d'interaction brute est exprimé à l'aide de nombres entiers que nous appelons *priorité* d'un élément d'assignation. La priorité de chaque élément d'assignation pour une entité $e \in \mathbb{E}$ est connue à l'aide d'une fonction appelée *fonction d'attribution des priorités*.

Définition 26. ***Priorité d'un élément d'assignation***
La **priorité** d'un élément d'assignation a pour une entité e est nombre entier relatif utilisé pour déterminer la position de a dans l'ensemble partiellement ordonné des interactions que l'entité e a la capacité d'initier. Cet ordre est utilisé pour construire le comportement réactif de cette entité.

Définition 27. ***Fonction d'attribution des priorités***
Soit $\mathcal{S} \in \mathbb{F}$.
On appelle **fonction d'attribution des priorités** d'une entité $e \prec \mathcal{S}$, notée $priorite(e)$, la fonction associant une priorité à chaque élément d'assignation contenu dans la ligne de la matrice d'interaction brute associée à $\mathcal{S}$. Plus formellement, pour toute entité $e \in \mathbb{E}$ on a :

$$priorite(e) : \begin{array}{c} \left(\mathbb{I}_{(1,0)} \times \mathbb{F}\right) \cup \left(\mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}\right) \\ (a) \end{array} \begin{array}{l} \rightarrow \\ \rightarrow \end{array} \begin{array}{c} \mathbb{Z} \\ \left\{ \begin{array}{ll} priorite(e)(a) & \text{si } a \in \mathcal{M}(famille(e), \mathcal{T}) \\ non\ défini & \text{sinon} \end{array} \right. \end{array}$$

On considère qu'un élément d'assignation e a un ordre plus élevé que l'élément d'assignation e' si la priorité de e est plus grande que celle de e'.

La seule connaissance de la fonction d'attribution des priorités permet de construire le comportement d'une entité sans avoir recours à une quelconque notion liée à la programmation. Les algorithmes que nous décrivons dans la section 4.3 page 119 du chapitre 4 en sont la preuve.

L'allure de la fonction d'attribution des priorités est fortement dépendante du phénomène étudié et des hypothèses de simulation émises. Dans le cas le plus simple, une priorité est associé une fois pour toute à chaque élément d'assignation.

Exemple. Priorités n'étant jamais modifiées
Dans un modèle de tri collectif tel que celui proposé par Resnick [Res97], une simulation implique deux familles d'entités que nous appelons ici `Trieur` *et* `Trié`*. Le comportement d'une entité e de la famille* `Trieur` *consiste à* SE DÉPLACER *dans l'environnement, à* RAMASSER *une entité de la famille* `Trié` *si e ne transporte rien et à* DÉPOSER *l'entité transportée à coté d'une entité de la famille* `Trié`*. Un exemple de matrice d'interaction brute de ce modèle est fournie dans la figure 3.11a. Dans cette simulation, le comportement des instances de la famille* `Trieur` *reste identique tout au long de la simulation. La fonction d'attribution des priorités est donc telle que décrite dans la figure 3.11b.*

Dans le cas le plus évolué, un changement dans l'état d'une entité peut l'amener à modifier son comportement (et donc les priorités associées à ses éléments d'assignation) sans pour autant modifier le comportement des autres instances de sa famille.

Exemple. Priorités pouvant évoluer
Soit une simulation en éthologie étudiant le comportement social d'animaux. Supposons qu'une telle simulation implique une seule famille d'entités que nous appelons `Animal`*. Le comportement d'une instance*

Source \ Cible	∅	`Trieur`	`Trié`
`Trieur`	(Se Déplacer)		(Ramasser, d=0) (Déposer, d=1)
`Trié`			

(a)

Source	Cible	Élément d'assignation	Priorité
`Trieur`	∅	(Se Déplacer)	0
`Trieur`	`Trié`	(Déposer, d=1)	1
`Trieur`	`Trié`	(Ramasser, d=0)	1

(b)

Figure 3.11 – Matrice d'interaction brute d'une simulation de tri collectif (a) et fonction d'attribution des priorités lui étant associée (b). Les éléments d'assignation (Déposer, d=1) et (Ramasser, d=0) ont la même priorité car leurs conditions sont mutuellement exclusives. Il n'y a donc en pratique jamais de choix aléatoire entre elles.

de la famille ***`Animal`*** *consiste par défaut à* Se Déplacer. *Si jamais cet animal rencontre un autre animal du genre opposé, il* Se Reproduit *avec ce dernier. Cela a pour conséquence d'ajouter un nouvel animal dans l'environnement et de provoquer une courte période de stérilité. Si jamais la reproduction n'est pas possible et si un animal du même genre est rencontré, alors ce dernier est* Attaqué. *Lorsqu'un animal est attaqué, il devient agressif et préfère alors* Attaquer *d'autres* ***`Animaux`*** *plutôt que de* Se Reproduir. *Un exemple de matrice d'interaction brute de ce modèle est fournie dans la figure 3.12a. Dans cette simulation, le comportement des instances de la famille* ***`Animal`*** *change selon leur état. La fonction d'attribution des priorités est donc telle que décrite dans la figure 3.12b.*

Source \ Cible	∅	`Animal`
`Animal`	(Se Déplacer)	(Se Repoduire, d=1) (Attaquer, d=1)

(a)

Source	Cible	Élément d'assignation	Priorité	
			¬ agressif	agressif
`Animal`	∅	(Se Déplacer)	0	0
`Animal`	`Animal`	(Se Reproduire, d=1)	2	1
`Animal`	`Animal`	(Attaquer, d=0)	1	2

(b)

Figure 3.12 – (a) Matrice d'interaction brute d'une simulation étudiant les comportements sociaux d'animaux et (b)fonction d'attribution des priorités lui étant associée. Dans cette simulation, un `Animal` non agressif préfère Se Reproduire à Attaquer. Au contraire, un `Animal` agressif préfère Attaquer à Se Reproduire.

Un grand nombre de simulations peuvent être réalisées sans avoir recours à des priorités pouvant changer au cours de la simulation. Pour nous conformer au principe de parcimonie, nous faisons le choix de ne pas considérer les cas où la priorité d'un élément d'assignation peut dépendre de l'état d'une entité. Par conséquent, il devient possible d'attribuer une unique priorité à chaque élément d'assignation directement dans la matrice d'interaction. Nous introduisons pour cela la notion de *matrice d'interaction raffinée* dont nous proposons un méta-modèle dans la figure 3.14.

Définition 28. ***Matrice d'interaction raffinée***
La **matrice d'interaction raffinée** $\mathcal{M}_{raff}$ d'une simulation est une matrice associant une priorité à chaque élément d'assignation a d'une matrice d'interaction brute $\mathcal{M}$. Plus formellement, elle est notée :

$$\mathcal{M}_{raff} : \begin{array}{ccc} \left(\mathbb{I}_{(1,0)} \times \mathbb{F}\right) \cup \left(\mathbb{I}_{(1,1)} \times \mathbb{R}^+ \times \mathbb{F} \times \mathbb{F}\right) & \rightarrow & \mathbb{Z} \\ (a) & \rightarrow & \begin{cases} priorite(e)(a), \forall e \prec source(a) & \text{si } \exists \mathcal{S} \in \mathbb{F}, \exists \mathcal{T} \in \mathbb{F} \cup \{\emptyset\} | a \in \mathcal{M}(\mathcal{S}, \mathcal{T}) \\ \text{non défini} & \text{sinon} \end{cases} \end{array}$$

La figure 3.13 présente un exemple d'une telle matrice pour une simulation de tri collectif dont la fonction d'attribution des priorités est décrite dans la figure 3.11.

3.7 Synthèse du chapitre

Nous avons établi dans la partie précédente que pour répondre aux besoins inhérents aux simulations large échelle le modèle doit exprimer un certain nombre de propriétés. Dans ce chapitre, nous définissons formellement le modèle de notre approche **Interaction Oriented Design of Agent simulations**

Source \ Cible	∅	`Trieur`	`Trié`
`Trieur`	(SE DÉPLACER, $p = 0$)		(RAMASSER, d=0, $p = 1$) (DÉPOSER, d=1, $p = 1$)
`Trié`			

FIGURE 3.13 – Matrice d'interaction raffinée d'une simulation de tri collectif, dont le modèle est résumé dans la section 3.6.2. Dans cette matrice, l'élément « (DÉPOSER, d=1, $p = 1$) » est lu « l'élément d'assignation (DÉPOSER, d=1) a pour priorité 1 ».

(IODA). Nous montrons de plus que les concepts y étant développés sont conformes à ces besoins et facilitent ainsi la conception de simulations large échelle.

Le modèle formel de IODA est fondé sur six notions fondamentales et repose en particulier sur le terme neutre « entité » pour que sa lecture ne soit pas biaisée par la connotation du terme « agent ». Dans le reste de ce manuscrit, nous utilisons les termes « entité » et « agent » comme synonymes. La notion la plus fondamentale du modèle est « l'interaction ».

Interaction. Une **interaction** représente une séquence sémantique d'actions qui n'est initiée que si ses **conditions** d'exécution sont vérifiées. Elle implique des entités **source** (qui initient l'interaction) conjointement à des entités **cible** (qui subissent l'interaction). Nous caractérisons le nombre d'entités source et cible d'une interaction par un couple (n, m) appelé **cardinalité**. Il caractérise le nombre n de sources et m de cibles impliquées dans une interaction.

Dans le cœur de l'approche IODA, nous nous focalisons sur deux types d'interactions, qui permettent d'exprimer une majorité de simulations :

- les **interactions individuelles** de cardinalité est $(1, 1)$;
- les **interactions dégénérées** de cardinalité est $(1, 0)$, qui prennent implicitement leur source pour cible.

Afin de décrire les interaction indépendamment des spécificités des entités ou de l'environnement, les actions et les conditions des interactions sont exprimées à l'aide de **primitives abstraites**. Ces primitives doivent être spécifiées par l'environnement pour que l'interaction puisse y avoir lieu ou par les entités pour qu'elles puissent participer à l'interaction. Une interaction définit pour cela des **signatures** contenant l'ensemble des primitives :

- devant être spécifiées par la source de l'interaction (**signature de la source**) ;
- devant être spécifiées par la cible de l'interaction (**signature de la cible**) ;
- devant être spécifiées par l'environnement (**signature de l'environnement**).

Une entité **peut** alors **initier** (resp. **subir**) une interaction si elle spécifie les primitives de la signature de la source (resp. cible) dans l'interaction. Par conséquent, une interaction est possible entre une entité source et une entité cible si la **source peut initier** l'interaction et si la **cible peut la subir**.

Les cinq autres notions développées dans IODA permettent d'intégrer de manière effective les interactions dans le comportement des entités.

Entité. La notion d'**entité** unifie la représentation de toute entité pertinente du phénomène simulé, qu'elle soit autonome et proactive ou n'initiant aucune interaction et qu'elle soit réactive ou cognitive. Nous effectuons dans IODA une distinction similaire à la différence classe/instance des langages objets en distinguant **instance d'entité** et **famille d'entités**. Une famille d'entités est en particulier caractérisée par :

- un **halo** qui permet d'identifier les entités perçues ;
- un ensemble de primitives qu'elle spécifie ;
- un ensemble d'interactions qu'elle peut initier ;
- un **modèle de sélection d'interaction** décrivant son comportement.

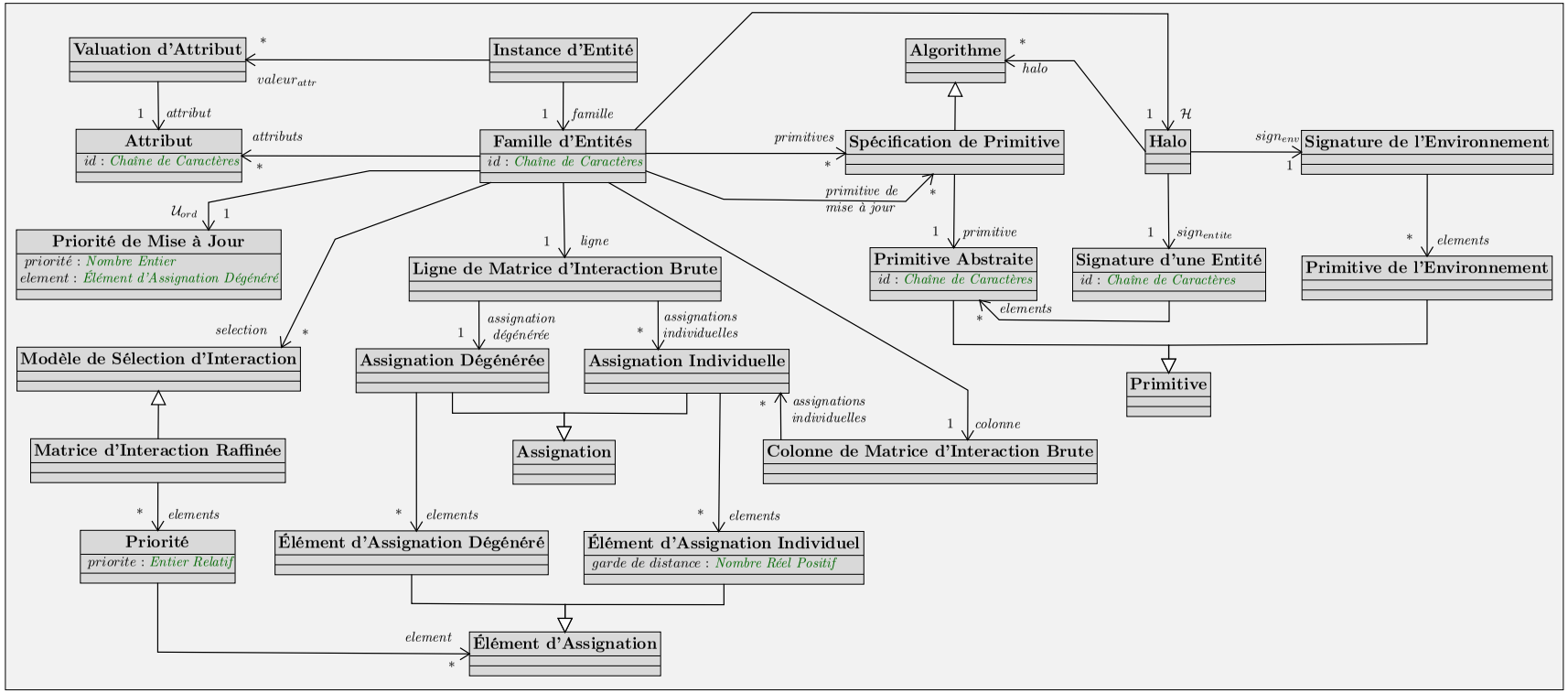


FIGURE 3.14 – Métamodèle d'une famille d'entité et d'une instance d'entité dans IODA, intégrant la notion de matrice d'interaction raffinée.

Environnement. L'**environnement** représente l'espace où se situent les entités. Il est caractérisé par un ensemble de primitives permettant de connaître son état et de le modifier. **Tout type d'environnement peut être utilisé** dans IODA du moment qu'une métrique y soit définie sous la forme d'une primitive de calcul de distance entre deux entités.

Matrice d'interaction brute. La **matrice d'interaction brute** définit l'ensemble des interactions pouvant survenir entre les entités d'une simulation. Une ligne y représente les interactions pouvant être initiées par une famille d'entités et une colonne les interactions pouvant être subies par une famille d'entités. Ainsi, une interaction figurant à l'**intersection** d'une **ligne** et d'une **colonne** caractérise une interaction pouvant avoir lieu entre la famille d'entités source et la famille d'entités cible concernées.

Matrice de mise à jour. La **matrice de mise à jour** représente les interactions initiées par une entité indépendamment de son comportement afin de mettre son état à jour. Elle permet de modifier simplement la façon dont l'état d'une entité évolue à l'aide d'interactions dégénérées.

Modèle de sélection d'interaction et matrice d'interaction raffinée. Le **modèle de sélection d'interaction** décrit comment une entité choisit l'interaction qu'elle initie à partir de sa ligne de la matrice d'interaction brute et de l'ensemble des entités qu'elle perçoit.

Les simulations large échelle peuvent difficilement être exécutées si le comportement des agents nécessite un grand nombre de calculs puisque dans de tels cas seul un nombre restreint d'agents peuvent être simulés. Le compromis le plus couramment utilisé consiste à utiliser un modèle de sélection d'action réactif dans lequel les règles condition/effet peuvent être plus ou moins cognitives. Nous intégrons à IODA un modèle de sélection d'interaction réactif fondé sur ce principe. Ce modèle repose sur une **matrice d'interaction raffinée** dans laquelle des priorités sont attribuées aux interactions figurant dans la matrice d'interaction brute. Le comportement d'une entité consiste alors à initier une interaction de priorité maximale dont les conditions sont vérifiées.

Modèle du temps. Afin de ne pas entraver la compréhension des concepts de notre approche centrée sur les interactions, nous avons choisi dans cette thèse de reposer sur une représentation du temps couramment rencontrée dans les simulations multi-agents :

- le modèle de sélection d'interaction ne peut sélectionner qu'une seule interaction ;
- le temps est discrétisé et décomposé en pas de temps de durée égale. Lors de chaque pas de temps, la parole est donnée aux agents séquentiellement.

Apports du modèle. Afin de faciliter la conception de simulations large échelle, nous avons établi dans la partie précédente que le modèle d'une simulation doit exprimer un certain nombre de propriétés. Le tableau 3.1 montre que le modèle formel IODA exprime de telles propriétés et facilite ainsi la conception de simulations large échelle.

Vers une approche transversale de conception. Le modèle IODA n'a d'intérêt que s'il peut reposer sur une approche de conception transversale afin d'implémenter une simulation. Dans le chapitre qui suit, nous prouvons ce point en décrivant la méthodologie IODA qui permet de passer d'une description d'un phénomène dans le langage naturel en un modèle IODA prêt à être implémenté en conservant sa structure à l'aide d'algorithmes.

TABLE 3.1 – Preuve que le modèle IODA facilite la conception de simulations large échelle. Dans cette figure nous résumons les propriétés facilitant la conception de simulation identifiées dans le chapitre 2 (colonne gauche) et exprimons à droite comment le modèle IODA s'y conforme (colonne droite).

Propriété souhaitée	Concept dans IODA
Le modèle est précis et décrit le phénomène à différents niveaux d'abstractions.	Les matrices décrivent la simulation à un haut niveau d'abstraction.
	Les conditions et actions des interactions décrivent la simulation à un niveau intermédiaire d'abstraction.
	Les primitives des agents décrivent la simulation à un niveau fin d'abstraction.
Déclaratif et procédural sont séparés.	La matrice d'interaction brute est une représentation abstraite des capacités des agents.
	La matrice d'interaction raffinée est une représentation abstraite du comportement des agents.
Les actions des agents sont représentées de manière générique et indépendante de leur comportement.	Il y a séparation entre matrice d'interaction brute et modèle de sélection d'interaction.
Les actions des agents sont représentées sous la forme de règles conditions/effets.	Les interactions sont construites selon ce principe.
Une interaction représente toute action impliquant simultanément plusieurs agents.	Les interactions sont construites selon ce principe.
Les interactions entre agents sont modélisées à l'aide d'un graphe ou d'une forme équivalente.	La matrice d'interaction brute est une forme équivalente au graphe.
Les interactions sont intégrées au comportement des agents.	Le modèle de sélection d'interaction se fonde sur une ligne de la matrice d'interaction brute.