Héritage et réutilisation de modèle dans IODA

Plan du chapitre :

La construction de simulations large échelle ne peut être envisagée sans profiter d'outils d'ingénierie logicielle similaires à l'héritage. En effet, ces outils permettraient d'éviter de fournir une spécification systématiquement exhaustive de la source et la cible de chaque interaction.

Dans ce chapitre, nous étudions sous quelles conditions il est possible d'intégrer des concepts proches de l'héritage dans l'approche IODA *afin de faciliter l'ingénierie des connaissances. A cette occasion, nous définissons une extension de* IODA *fondée sur la relation de spécialisation qui facilite l'ingénierie des connaissances dans une simulation. En effet, la spécialisation exprime une relation sémantique entre familles d'agents qui permet :*

- *de représenter des matrices d'interaction brutes exprimant un grand nombre d'interactions entre agents à l'aide d'une quantité réduite d'éléments ;*
- *d'exprimer qu'une famille d'agents est une extension d'une autre famille d'agents. Par exemple « une plante carnivore est une plante sachant de plus manger des insectes » ;*
- *d'exprimer qu'une famille d'agents est une exception à une autre famille d'agents. Par exemple « un client aveugle est un client ne sachant pas lire de panneaux publicitaires ».*

D'une manière générale, plus le modèle d'une simulation contient d'informations, plus on est susceptible d'y retrouver des redondances, *i.e.* des portions de modèle réutilisées en plusieurs endroits. L'un des principaux enjeux de la conception de telles simulations est alors de structurer le modèle afin de faciliter la spécification de ces portions du modèle et, si possible, à les réutiliser au maximum.

Une première approche favorisant la réutilisation dans le modèle consiste à le diviser en un maximum d'unités disjointes les plus indépendantes possible et à réifier chacune de ces unités en un élément logiciel. Cette séparation permet de constituer des bibliothèques d'éléments pouvant être réutilisés autant lors de la construction du modèle que lors de son implémentation. C'est par exemple le cas dans la plateforme Volcano [Dem95], où un agent est conçu comme une agrégation de briques logicielles décrivant respectivement : son comportement, comment il communique avec les autres agents, comment il communique avec l'environnement et quels sont ses liens sociaux avec les autres agents. Il en va de même pour la plateforme JADE [BPR99], où le comportement d'un agent est décomposé en activités, implémentées chacune à l'aide d'une instance de la classe Behavior. et où le comportement est décrit par l'utilisateur en spécifiant l'activation séquentielle, cyclique ou parallèle de ces activités. On retrouve aussi ce principe de décomposition dans la plateforme Jack [BHRH00], où les agents sont définis par des plans, des ensembles de croyances et des buts. Ces éléments sont tous représentés sous la forme d'une entité logicielle autonome (une classe) permettant ainsi leur réutilisation pour des agents différents. L'approche IODA décrite dans cette thèse se place aussi dans cette perspective, mais va plus loin en fournissant les outils permettant non seulement de réifier les interactions en unités logicielles autonomes, mais aussi de les décrire de manière générique et indépendante des spécificités des agents, ce qui favorise leur réutilisation dans des agents

aux spécificités très différentes.

Chaque unité élémentaire constituant le modèle représente une unité sémantique de base permettant de décrire le phénomène. Bien que ces unités soient décrites de manière disjointe, certaines peuvent être regroupées et concourir à la description d'un concept plus complexe. Par exemple :

- le fait de savoir chasser un autre agent et de le manger caractérise un prédateur ;
- le fait de vieillir et mourir caractérise un être vivant ;
- le fait de savoir se déplacer caractérise une entité mobile, *etc.*

Une deuxième approche favorisant la réutilisation dans le modèle consiste à décrire les agents à l'aide de ces regroupements sémantiques. Dans cette section, nous étudions cette question du point de vue du modèle et de l'implémentation d'une simulation, en nous appuyant sur les concepts développés dans IODA.

8.1 Héritage en simulation multi-agents

En génie logiciel, le concept s'approchant le plus de ce type de descriptions est l'héritage, utilisé en premier lieu en programmation orientée-objet. L'héritage est une relation binaire liant un objet "père" à un objet "fils", statuant que l'objet "fils" est une instance particulière de l'objet "père". En termes de réutilisation, cela implique que tous les attributs et toutes les méthodes figurant dans l'objet "père" doivent figurer dans l'objet "fils".

Un grand nombre de plateformes permettant d'implémenter des systèmes multi-agents reposent sur des langages orientés-objets. Dans ces plateformes, un agent est en général implémenté sous la forme d'une classe ou un objet. En conséquence, le concept d'héritage peut leur être appliqué, permettant ainsi d'une part de fournir une structure aux liens sémantiques existant entre les divers agents d'une simulation, mais aussi de permettre dans une certaine mesure leur réutilisation. Caractériser une telle relation entre deux agents au niveau du modèle s'avère plus problématique. En effet, l'héritage des langages orientés-objet s'exprime en termes de méthodes et attributs réutilisés, alors que dans le cas d'agent, on aimerait exprimer la réutilisation en termes de façon de percevoir, en termes de contenu de la mémoire, en termes d'interactions qu'un agent est capable d'initier, ou en termes de modèle de sélection d'interaction.

La construction de la sémantique d'un modèle commence inévitablement par l'identification des identifiants des familles d'agents, ainsi que les identifiants des interactions impliquées dans le phénomène. Lors de cette étape, les familles d'agents peuvent être utilisées pour exprimer des concepts du phénomène très différents :

1. une espèce, un genre, une famille, un ordre, une classe ou un embranchement, au sens des sciences du vivant. Par exemple `Végétaux`, `Animaux`, `Minéraux`, *etc.*
2. une catégorie particulière d'individus ayant des caractéristiques communes. Par exemple `AgentAvecInventaire`, qui représente l'ensemble des agents pouvant contenir quelque chose. Cette famille d'agents caractérise alors autant un `Facteur` qu'une `Remorque` ou une `Corbeille` ;
3. une catégorie particulière d'individus ayant des capacités communes. Par exemple, un `Être vivant` est capable de MOURIR et VIEILLIR. Cette famille d'agents caractérise alors autant un `Loup` qu'une `Personne` ou une `Plante` ;
4. une catégorie particulière d'individus ayant leur comportement en commun. Par exemple, un `Virus` a pour comportement de SE DÉPLACER dans un corps humain, PÉNÉTRER dans une `Cellule`, puis SE DUPLIQUER dans cette `Cellule`. Cette famille d'agents caractérise des virus pouvant avoir des effets très différents sur la cellule infectée, allant de l'INHIBITION de certaines de ses fonctions (par exemple dans le cas du `VIH`), à la FUSION de cellules (par exemple dans le cas du `Virus de Sendai`) ;
5. toute combinaison des quatre concepts précédents.

La notion de *capability* de la plateforme Jack [BHRH00] est particulièrement intéressante de ce point de vue. En effet, ce concept permet de structurer les éléments relatifs au raisonnement des agents (plans, croyances et buts) en blocs sémantiques. Un agent associé à une *capability* dispose de tous les plans, croyances et buts qu'elle contient. Une *capability* peut de plus hériter d'autres *capabilities* et donc disposer de tous les plans, croyances et buts qu'elle contient. Un agent peut alors être construit en décrivant dans un premier temps des fonctionnalités abstraites à l'aide de *capabilities*, puis en attribuant ces fonctionnalités

aux agents de la simulation. Ainsi, des librairies de *capabilities* réutilisables sont construites. Ces dernières simplifient la conception d'applications pouvant contenir un grand nombre d'agents.

Cette approche ne fait pas apparaître les interactions ayant lieu entre agents. La solution proposée est donc susceptible d'éluder un certain nombre de problèmes, que nous proposons d'étudier ci-après.

8.1.1 Nature des problèmes liés aux interactions

Deux types de problèmes de modélisation sont rencontrés lors de la spécification de grandes matrices d'interactions. L'un d'entre eux est lié à l'énumération exhaustive de chaque cible d'une interaction pour une famille d'agents source donnée, l'autre est lié à la re-spécification systématique des interactions pouvant être initiées par différentes familles d'agents sources ayant pourtant une souche commune. Dans cette section, nous illustrons en quoi consistent ces problèmes et fournissons l'idée générale de la solution que nous employons pour y remédier.

Problèmes liés aux cibles

Exprimer explicitement quelles sont les cibles d'une interaction induit un double problème. Lorsque qu'une famille d'agent source a la capacité d'initier une interaction individuelle ou de multicast avec un grand nombre de famille d'agents différentes pour cible, spécifier la matrice d'interaction devient fastidieux, puisque **l'interaction doit figurer dans la colonne de chacune des familles d'agents cibles**. De plus, cette approche est peu robuste à l'ajout de nouvelles familles d'agents devant elles aussi être la cible de l'interaction. Prenons l'exemple d'un `Lymphocyte`, dont le rôle dans le corps est de PHAGOCYTER le `Virus Rhume`, `Virus Grippe` et `Virus Varicelle`. Ces trois virus ont tous trois un comportement différent que nous supposons déjà spécifié. Une telle simulation est modélisée à l'aide de la matrice d'interaction brute décrite dans la figure 8.1(a). Si jamais une nouvelle famille d'agents VIRUS ROUGEOLE devait être ajoutée, alors la ligne de la matrice d'interaction brute du Lymphocyte doit être modifiée (voir figure 8.1(b)), bien que son comportement reste rigoureusement identique : la phagocytose des virus.

Source \ Cible	$\emptyset$	Virus Rhume	Virus Grippe	Virus Varicelle
Lymphocyte	(SE DÉPLACER)	(PHAGOCYTER, d = 0)	(PHAGOCYTER, d = 0)	(PHAGOCYTER, d = 0)

(a) Matrice décrivant la base d'un comportement de Lymphocyte

Source \ Cible	$\emptyset$	Virus Rhume	Virus Grippe	Virus Varicelle	Virus Rougeole
Lymphocyte	(SE DÉPLACER)	(PHAGOCYTER, d = 0)	(PHAGOCYTER, d = 0)	(PHAGOCYTER, d = 0)	(PHAGOCYTER, d = 0)

(b) Matrice décrivant l'extension du comportement de Lymphocyte à la phagocytose d'un nouveau virus

FIGURE 8.1 – Matrices d'interaction brutes décrivant d'une part le comportement d'un `Lymphocyte` vis-à-vis du `Virus Rhume`, du `Virus Grippe` et du `Virus Varicelle` (figure 8.1(a)) et d'autre part l'extension de ce comportement au `Virus Rougeole` (figure 8.1(b)).

Pour remédier à ce problème, nous étendons l'approche IODA en lui ajoutant la *relation de spécialisation* entre une famille d'agent "fille" et une famille d'agents "mère". Définir une famille d'agents "mère" en tant que cible d'une interaction dans la matrice d'interaction brute est alors un raccourci spécifiant que cette famille d'agents, ainsi que toutes les familles d'agents spécialisant cette famille d'agents "mère" peuvent être la cible de cette interaction. Dans l'exemple du `Lymphocyte`, la famille d'agents `Virus` est spécialisée par les familles d'agents `Virus Rhume`, `Virus Grippe`, `Virus Varicelle` et `Virus Rougeole`. Cette relation simplifie la spécification de la matrice d'interaction brute (voir figure 8.2) et la rend plus robuste aux modifications. En effet, l'ajout d'une nouvelle espèce de `Virus` ne nécessite pas la modification du comportement du `Lymphocyte`.

Problèmes liés aux sources

Dans IODA, une ligne de la matrice d'interaction brute doit être définie pour chaque famille d'agents ayant un comportement autonome. Certaines de ces familles peuvent avoir beaucoup en commun et

Source \ Cible	∅	Virus
Lymphocyte	(SE DÉPLACER)	(PHAGOCYTER, d = 0)

FIGURE 8.2 – Matrice d'interaction brute décrivant le comportement d'un `Lymphocyte` vis à vis d'une famille d'agents `Virus`, qui est spécialisée par les familles d'agents `Virus Rhume`, `Virus Grippe`, `Virus Varicelle` et `Virus Rougeole`. Cette matrice est équivalente à celle présentée dans la figure 8.1

l'adaptation d'un concept proche des *capabilities* de Jack permettrait de simplifier leur conception. Prenons l'exemple d'une simulation en éthologie où évoluent trois espèces animales et une espèce végétale, représentées respectivement à l'aide d'une famille d'agents `Aigle`, `Faucon`, `Rongeur` et `Végétation`. Cette simulation modélise les relation de proie et prédation entre les différentes familles d'agents constituant la simulation, où les entités peuvent de plus vieillir et mourir de vieillesse. La matrice d'interaction brute et la matrice de mise à jour d'une telle simulation sont résumées sur la figure 8.3. Dans ce modèle, il y a beaucoup de redondances : toutes les familles d'agents peuvent initier l'interaction MOURIR et l'interaction de mise à jour VIEILLIR, trois familles d'agents peuvent initier l'interaction de mise à jour AUGMENTER SENSATION FAIM ainsi que l'interaction SE DÉPLACER. Deux plus, deux familles d'agents représentent des espèces animales apparentées, qui partagent une partie de leur régime alimentaire.

Source \ Cible	∅	Aigle	Faucon	Rongeur	Végétation
Aigle	(SE DÉPLACER) (MOURIR)	(SE REPRODUIRE, d = 0)	(MANGER, d = 0)	(MANGER, d = 0)	
Faucon	(SE DÉPLACER) (MOURIR)		(SE REPRODUIRE, d = 0)	(MANGER, d = 0)	
Rongeur	(SE DÉPLACER) (MOURIR)			(SE REPRODUIRE, d = 0)	(MANGER, d = 0)
Végétation	(MOURIR) (SE PROPAGER)				

(a) Matrice d'interaction brute

	Aigle	Faucon	Rongeur	Végétation
Interactions de mise à jour	VIEILLIR AUGMENTER SENSATION FAIM	VIEILLIR AUGMENTER SENSATION FAIM	VIEILLIR AUGMENTER SENSATION FAIM	VIEILLIR

(b) Matrice de mise à jour

FIGURE 8.3 – Matrice d'interaction brute et matrice de mise à jour d'une simulation en éthologie contenant les familles d'agents AIGLE, FAUCON, RONGEUR et VÉGÉTATION.

Il serait donc possible de le structurer afin d'y favoriser l'ajout futur d'autres espèces animales. La relation de spécialisation mentionnée précédemment peut jouer ce rôle. Définir qu'une famille d'agents "mère" est capable d'initier une interaction est alors un raccourci spécifiant que toutes les familles d'agents spécialisant la famille d'agents "mère" sont aussi capables d'initier cette interaction. Dans l'exemple de l'écosystème mentionné ici, cela reviendrait par exemple à :

- créer une famille d'agents `Rapace` qui rassemble les interactions communes aux familles d'agents `Faucon` et `Aigle` ;
- créer une famille d'agents `Animal` qui rassemble les interactions communes aux familles d'agents `Aigle`, `Faucon` et `Rongeur` ;
- créer une famille d'agents `Vivant` qui rassemble les interactions communes aux familles d'agents `Aigle`, `Faucon`, `Rongeur` et `Végétation`.

Cela aboutirait alors à une matrice d'interaction similaire à celle présentée sur la figure 8.4.

Problèmes inhérents à la spécialisation

L'utilisation de la spécialisation telle que schématisée dans les deux sections précédentes allège la construction d'une matrice d'interaction brute. Elle favorise de plus la réutilisation logicielle de lignes de la matrice déjà spécifiées. Toutefois, ces notations induisent un certain nombre de problèmes qu'il est nécessaire de résoudre afin de pouvoir utiliser ce concept sans ambiguïtés.

Source \ Cible	∅	Aigle :: Rapace	Faucon :: Rapace	Rongeur :: Animal	Végétation :: Vivant
Vivant	(Mourir)				
Végétation :: Vivant	(Se Propager)				
Animal :: Vivant	(Se Déplacer)				
Rongeur :: Animal				(Se Reproduire, d = 0)	(Manger, d = 0)
Rapace :: Animal				(Manger, d = 0)	
Aigle :: Rapace		(Se Reproduire, d = 0)	(Manger, d = 0)		
Faucon :: Rapace			(Se Reproduire, d = 0)		

(a) Matrice d'interaction brute

	Vivant	Animal :: Vivant
Interactions de mise à jour	Vieillir	Augmenter Sensation Faim

(b) Matrice de mise à jour

Figure 8.4 – Matrice d'interaction brute et matrice de mise à jour d'une simulation en éthologie équivalente à celle décrite dans la figure 8.3. Toutefois, le modèle y est exprimé à l'aide de la spécialisation de familles d'agents, permettant l'ajout plus simple de nouvelles familles d'agents. Dans cette figure, la relation "::" spécifie que la famille d'agents présente dans le membre de gauche spécialise les familles d'agents présentes dans le membre de droite. "`Aigle` :: `Rapace`" signifie alors qu'un `Aigle` est une sorte de `Rapace` pouvant en plus Manger des `Faucons` et Se Reproduire avec d'autres `Aigles`.

Classes abstraites et méthodes abstraites Le concept de spécialisation pose d'abord problème au niveau des primitives à spécifier. En effet, certaines familles d'agents sont trop générales pour pouvoir fournir une spécification sensée des primitives abstraites des interactions qu'elle peut subir ou effectuer. C'est le cas de l'exemple fourni dans la section 8.1.1, où la famille `Virus` doit fournir une spécification aux primitives liées aux cibles de l'interaction Phagocyter, alors que ce code peut être spécifique à chacune des familles `Virus Rhume`, `Virus Grippe`, *etc.* Il faudrait donc pouvoir ne pas fournir de spécification à certaines primitives, ce qui n'est actuellement pas possible dans le modèle Ioda présenté dans le chapitre 3.

Héritage multiple Le second problème est aussi lié aux primitives. Il est rencontré lorsqu'une famille d'agents spécialise deux familles d'agents pouvant initier la même interaction, ou subir la même interaction. Dans cette situation, si les deux familles d'agents "mères" fournissent une spécification aux primitives, il faut alors pouvoir déterminer laquelle de ces spécifications doit être utilisée dans la famille d'agents "fille". Ce problème est similaire au problème du losange rencontré en conception orientée-objet. Il faut donc trouver un moyen permettant de décider quelles spécifications des primitives sont réutilisées par la famille d'agents "fille".

Dispersion de la connaissance Le troisième problème survient lorsqu'une famille d'agents spécialise un nombre élevé de familles d'agents. Dans de tels cas, la réutilisation de modèles existants est grandement favorisée, au prix de la facilité de compréhension du comportement des agents. En effet, les interactions que cette famille d'agents peut initier ou subir sont éparpillées dans des familles d'agents très différentes, si bien qu'il est difficile d'avoir une vue d'ensemble sur toutes les interactions auxquelles peut participer cette famille d'agents. Il faudrait donc trouver un compromis entre expressivité de la matrice d'interaction brute et possibilité de réutiliser des comportements déjà existants.

8.1.2 Héritage et ingénierie des connaissances

Une transposition naïve du concept d'héritage des langages orientés-objet à la simulation consiste à construire les familles d'agents en :

– récupérant l'ensemble des spécifications effectuées dans les familles d'agents "mères" ;
– complétant éventuellement ces spécifications par l'ajout de nouveaux éléments d'assignation et primitives.

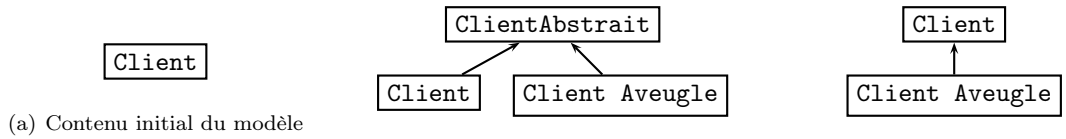
Selon cette approche, si deux familles d'agents ont en commun une partie des interactions qu'elles sont capables d'initier, alors une famille d'agents intermédiaire doit être créée. Son rôle est de définir le tronc commun de ces deux familles d'agents. Dans cette section, nous illustrons succinctement certains problèmes sous-jacents à cette approche et décrivons une approche différente qui permettrait de s'en détacher.

Considérons une simulation reproduisant le comportement (simplifié) de `Clients` dans un magasin. Ce comportement consiste à SE DÉPLACER, LIRE les informations affichées sur un `Panneau Publicitaire`, RAMASSER des `Articles` et PAYER à une `Caisse` les articles achetés. Dans cette simulation, les `Caisses` peuvent de plus SIGNALER LEUR OUVERTURE à un `Client`. La matrice d'interaction brute de cette simulation est résumée sur la figure 8.5.

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Client
Client	(SE DÉPLACER)	(LIRE, d = 5)	(RAMASSER, d = 0)	(PAYER, d = 0)	
Caisse					(SIGNALER OUVERTURE, d = 3)

FIGURE 8.5 – Matrice d'interaction brute décrivant ce qu'un `Client` et une `Caisse` sont capables de faire dans un magasin.

Supposons maintenant qu'un raffinement du modèle amène à l'ajout au modèle de `Clients Aveugles`. Ces derniers sont des clients ne pouvant pas LIRE de `Panneaux Publicitaires` et percevant leur voisinage dans des modalités différentes des clients ne souffrant pas de ce handicap. Dans la suite, nous décrivons deux approches différentes permettant de modéliser cette modification du modèle. Nous nous focalisons pour cela sur la relation de existant entre la famille d'agents `Client` et la famille d'agents `Client Aveugle`. Les idées principales de ces deux approches sont résumées sur la figure 8.6.



(a) Contenu initial du modèle

(b) Modification du modèle en utilisant l'approche classique, qui implique la modification de la famille d'agents `Client`

(c) Modification souhaitable du modèle, qui n'implique pas la modification de la famille d'agents `Client`

FIGURE 8.6 – Illustration du problème de modélisation lié à l'ajout d'une famille d'agents dont le comportement est une exception au comportement d'une autre famille d'agents. Ce problème est illustré dans le cas de la modélisation du comportement d'un `Client Aveugle` dans un magasin, dont le comportement est identique à celui d'un `Client`, hormis le fait qu'il n'est pas capable d'initier des interactions telles que LIRE avec un `Panneau Publicitaire` pour cible. Dans cette figure, une flèche allant de A vers B signifie que A réutilise tout (figure 8.6(b)) ou partie (figure 8.6(c)) du comportement et des interactions de B.

La pratique classique du génie logiciel voudrait la création d'une famille d'agents intermédiaire (par exemple `ClientAbstrait`) qui regroupe les capacités d'interagir communes aux familles d'agents `Client` et `Client Aveugle`. Il faudrait de plus s'assurer que les deux familles d'agents "héritent" de la famille `ClientAbstrait`. Cette façon d'organiser les familles d'agents est résumée sur la figure 8.6(b). Cette description aboutirait alors à la matrice d'interaction brute présentée sur la figure 8.7.

Dans cet exemple, l'ajout de la famille d'agents `Client Aveugle` implique :
– la modification de la ligne de la famille d'agents `Client` dans la matrice d'interaction ;
– la création d'une famille d'agents `ClientAbstrait` qui regroupe ces interactions communes à `Client` et `Client Aveugle`.

Cette modification a pour conséquence que les `Caisses` ne peuvent pas signaler leur ouverture aux `Clients Aveugles`.

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	ClientAbstrait
ClientAbstrait	(SE DÉPLACER)		(RAMASSER, d = 0)	(PAYER, d = 0)	
Client :: ClientAbstrait	—	(LIRE, d = 5)	—	—	
Client Aveugle :: ClientAbstrait					
Caisse					(SIGNALER OUVERTURE, d = 3)

FIGURE 8.7 – Matrice d'interaction brute décrivant ce qu'un `Client`, un `Client Aveugle` et une `Caisse` sont capables de faire dans un magasin. Cette matrice est construite en factorisant les capacités d'interaction communes aux familles d'agents `Client` et `Client Aveugle` au sein d'une troisième famille d'agents `ClientAbstrait`. Les modifications du modèle initial présenté dans la figure 8.5 apparaissent en rouge dans la présente figure.

Pour corriger ce problème, toute occurrence de la famille d'agents `Client` en tant que cible d'une interaction a du être remplacée par la nouvelle famille d'agents `ClientAbstraite`. Cela implique la modification des lignes de la matrice d'interaction associées à chaque famille d'agents pouvant auparavant interagir avec des `Clients`. Enfin, la création d'une famille d'agents supplémentaire et l'introduction de relations de spécialisation supplémentaires dispersent encore plus la connaissance des agents.

De telles situations surviennent dès que l'on cherche à définir le comportement d'un agent qui constitue une exception. Dans de tels cas, plutôt que de tenter de factoriser les points communs des deux familles d'agents, il peut s'avérer fructueux d'utiliser une approche inverse où l'on caractérise une famille d'agents par les interactions n'étant pas héritées de la famille d'agents mère. Dans l'exemple développé ici, cela reviendrait à mettre en exergue qu'un `Client Aveugle` est un `Client` n'étant pas capable de LIRE des `Panneaux Publicitaires`. Cette façon de modéliser se traduirait alors par une matrice d'interaction brute prenant la forme illustrée sur la figure 8.8. Cette façon d'organiser les familles d'agents est résumée sur la figure 8.6(c).

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Client
Client	(SE DÉPLACER)	(LIRE, d = 5)	(RAMASSER, d = 0)	(PAYER, d = 0)	
Client Aveugle :: Client		−(LIRE, d = 5)			
Caisse					(SIGNALER OUVERTURE, d = 3)

FIGURE 8.8 – Matrice d'interaction brute décrivant ce qu'un `Client`, un `Client Aveugle` et une `Caisse` sont capables de faire dans un magasin. Dans cette matrice, le signe − peut apparaître devant un élément d'assignation, ce qui signifie que l'interaction lui étant associée est retirée des interactions pouvant être initiées ou subies par la famille d'agents. Ainsi, la ligne associée à la famille d'agents `Client Aveugle` se lit « Un `Client Aveugle` est un `Client` particulier (*i.e.* Client Aveugle ::Client) » ne sachant pas LIRE de `Panneaux Publicitaires` (*i.e.* −(LIRE, d = 5)). Les modification du modèle initial présenté dans la figure 8.5 apparaissent en rouge dans la présente figure.

Ce changement de perspective se révèle particulièrement intéressant lorsque l'on souhaite concevoir des familles d'agents dont le comportement constitue une exception du comportement d'une autre famille d'agents, par exemple lorsqu'il s'agit de modéliser le comportement d'une `Personne Aveugle` à partir du comportement d'une `Personne` qui n'est pas aveugle, le comportement d'un `Animal Stérile` à partir du comportement d'un `Animal` qui n'est pas stérile *etc.* En effet, il ne nécessite ni l'ajout d'une nouvelle famille d'agents intermédiaire, ni la modification de la matrice d'interaction brute.

8.2 Spécialisation de familles d'agents dans IODA/JEDI

Pour répondre aux diverses problématiques mentionnées ci-avant, nous transposons dans un premier temps à IODA les principales notions liées à l'héritage dans les langages orientés-objets. Nous décrivons ensuite comment profiter de la spécification synthétique des agents permise par l'héritage sans pour autant souffrir de la dispersion des connaissances. Cette approche permet ainsi concilier ingénierie des

connaissances et ingénierie logicielle. Enfin, nous décrivons quels concepts permettent de spécifier les familles d'agents selon une approche radicalement opposée à l'héritage des langages orientés-objets : concevoir des exceptions à certaines familles d'agents, en modifiant ou en supprimant certaines de leurs capacités à interagir.

8.2.1 Spécialisation et héritage

Dans cette extension du corps de l'approche IODA, la relation d'héritage des langages orientés-objets est transposée en une relation que nous appelons *spécialisation*.

Définition 48. ***Spécialisation***
La **spécialisation** est une relation binaire entre une famille d'agents appelée **fille** et une famille d'agents appelée **mère**. Elle exprime le fait que la famille d'agents fille constitue un sous-ensemble particulier de la famille d'agents mère.

Cette relation est exprimée dans la méthodologie IODA lors de la construction de l'ensemble des identifiants des familles d'agents. En plus de la construction de la liste des identifiants, cette étape établit les relations de spécialisation entre familles d'agents sous la forme d'un graphe similaire à un diagramme de classes : les noeuds y sont des identifiants de familles d'agents et les arcs représentent une relation de spécialisation. Les arcs ont pour origine une famille d'agents fille et pointent vers une famille d'agents mère. Un tel graphe est illustré sur la figure 8.2.1 page 228 pour une simulation d'un écosystème dont la matrice d'interaction est décrite dans la figure 8.4.

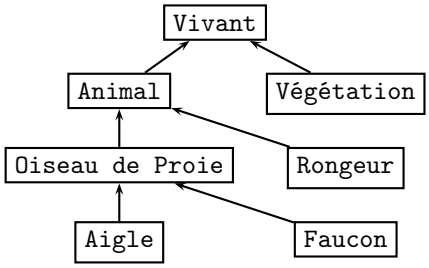


FIGURE 8.9 – Graphe dirigé résumant le lien de spécialisation entre familles d'agents, dans le cas de la simulation d'un écosystème déjà décrit sous la forme d'une matrice d'interaction brute dans la figure 8.2.1 page 228. Une flèche y représente une relation de spécialisation entre deux familles d'agents, qui pointe vers la famille d'agent mère de la relation.

La spécialisation s'exprime aussi dans la matrice d'interaction brute, dans la matrice de mise à jour, et dans la matrice de mise à jour ordonnée, en postfixant le nom de chaque famille d'agents par les caractères "::", suivis d'une liste de familles d'agents mères. Par exemple, dans la figure 8.4(b), l'expression « `Animal`::`Vivant` » spécifie que la famille d'agents `Animal` spécialise la famille d'agents mère `Vivant`. Si jamais la famille d'agents ne spécialise aucune famille d'agents particulière, les caractères "::" n'apparaissent pas. La relation de spécialisation est transitive. Par conséquent, pour éviter de surcharger le contenu de la matrice d'interaction brute, seules les familles d'agents mères étant directement en relation avec la famille d'agents fille sont mentionnées dans l'entête de chaque ligne et de chaque colonne de la matrice.

Cette relation a deux conséquences dans la matrice d'interaction. Premièrement, **une famille d'agents *fille* peut subir toutes les interactions que la famille d'agents *mère* peut subir**. Des regroupements sémantiques de familles d'agents sont donc créés. Ils permettent de désigner plus simplement un ensemble de familles d'agents cibles à l'aide d'un seul élément d'assignation. Cela allège d'une part la description du contenu de la matrice d'interaction brute et cela facilite d'autre part l'intégration de nouvelles familles d'agents au modèle sans avoir à modifier les familles déjà existantes. Cette première utilisation de la spécialisation est illustré dans la section 8.1.1 pour une simulation où un `Lymphocyte` peut PHAGOCYTER des `Virus` tels que le `Virus Rougeole` ou le `Virus Rhume`.

La seconde conséquence de cette relation est qu'**une famille d'agents *fille* récupère par défaut toutes les interactions de la matrice d'interaction brute que la famille d'agents *mère* est capable d'initier**. Le même principe s'applique aux interactions présentes dans la matrice de mise à jour. Cette seconde utilisation de la spécialisation est illustrée dans la section 8.1.1 pour la simulation d'un écosystème.

Dans les deux cas, la famille d'agents fille doit spécifier les primitives issues des interactions que la famille d'agents mère peut subir ou effectuer. Dans le cas le plus simple, la spécification de ces primitives consiste à réutiliser les spécifications fournies dans la famille d'agents mère ou, s'il en est besoin, de surcharger certaines primitives pour leur fournir un comportement plus spécifique. En pratique, deux cas récurrents échappent à ces principes simples de spécification.

Familles d'agents abstraites

Le premier cas posant problème dans la relation de spécialisation est rencontré si la famille d'agent mère ne peut pas fournir de spécification pertinente à toutes les primitives, car elle représente un groupement trop abstrait d'agents (il s'agit du premier problème inhérent à la spécialisation mentionné dans la section 8.1.1). Pour traiter ce genre de cas, nous introduisons le concept de *famille d'agents abstraite*, où certaines primitives peuvent ne pas être spécifiées.

Définition 49. ***Famille d'agents abstraite***
Une **Famille d'agents abstraite** est une famille d'agents ne fournissant pas de spécifications à au moins l'une de ses primitives. Ce type de famille d'agents ne peut être instanciée directement.

Dans la méthodologie IODA, déterminer si une famille d'agents est abstraite ou non se fait lors de l'étape de spécification de leurs primitives. Pendant cette étape, le concepteur peut décider de ne pas fournir de spécifications à une primitive, dans quel cas la famille d'agents devient obligatoirement abstraite. Si jamais le concepteur spécifie toutes les primitives de la famille d'agents, alors cette famille n'est plus abstraite.

Problème de l'héritage multiple

Le second cas posant problème dans la relation de spécialisation est rencontré lorsque la famille d'agents fille spécialise plus d'une famille d'agents, et que ces familles fournissent des spécifications différentes à une même primitive (il s'agit du deuxième problème inhérent à la spécialisation mentionné dans la section 8.1.1). Ce problème, aussi connu sous le nom de l'héritage multiple en programmation orientée-objet, consiste à déterminer quel moyen utiliser pour identifier quelle spécification fournir aux primitives d'une famille d'agents fille. De nombreuses techniques permettent de résoudre ce problème en programmation orientée-objet.

La première est trouvée par exemple dans le langage Perl [Wal10]. Elle consiste à ordonner l'ensemble des classes mères lors de la déclaration de l'objet et à utiliser la première spécification de la primitive rencontrée en itérant sur cet ensemble. Une autre spécification, retrouvée par exemple dans Scala [LAM10] ou Python [Fou10], consiste à remonter dans l'arbre d'héritage en effectuant un parcours en largeur d'abord et à utiliser la spécification de la première primitive rencontrée.

Dans le cas de la simulation, de telles sélections automatisées sont à éviter. En effet, elles reviennent à définir de manière arbitraire la sémantique du modèle. La solution considérée dans IODA consiste à faire ce choix manuellement dans la méthodologie de conception lors de l'étape de spécification des primitives. Ce choix de conception se retrouve par exemple dans le langage C++ [SKM02].

Puisque certaines familles d'agents mères peuvent être abstraites, elles peuvent ne pas fournir de spécification à certaines primitives. Par conséquent, avant de pouvoir choisir la spécification d'une primitive dans une famille d'agents fille, il faut au préalable effectuer l'étape de spécification des primitives de toutes ses familles d'agents mère. Pour cela, nous imposons un ordre dans lequel les agents se voient spécifier leurs primitives. Il s'agit d'un parcours en largeur d'abord des liens de spécialisation, dont le point de départ sont les familles d'agents ne spécialisant aucune autre famille d'agents.

Synthèse

Dans cette section, nous avons caractérisé la spécialisation comme un moyen permettant de construire les familles d'agents à l'aide d'un moyen proche de l'héritage dans le systèmes multi-agents, que l'on retrouve par exemple avec la notion de capability dans Jack. Cette caractérisation de la spécialisation répond aux problèmes relatifs aux sources et aux cibles ainsi qu'aux deux premiers problèmes inhérents à la spécialisation mentionnés dans la section 8.1.1.

8.2.2 Forme synthétique et forme étendue

La spécialisation permet d'exprimer sous forme synthétique l'ensemble des interactions que des agents sont capables d'initier ou subir ainsi que l'ensemble de leurs interactions de mise à jour. Cette relation facilite ainsi la conception de la matrice d'interaction brute et la factorisation de code dans les agents. Toutefois, cet apport logiciel a un coût du point de vue de l'ingénierie des connaissances : la matrice d'interaction brute devient moins facile à interpréter. En effet, les interactions pouvant être initiées ou subies par une famille d'agents sont éparpillées parmi ses différentes familles d'agents mères. Ingénierie des connaissances et ingénierie logicielle sont toutes deux fondamentales pour la conception de simulations contenant un grand nombre d'informations. Par conséquent, la méthodologie de conception doit fournir le meilleur compromis possible entre l'utilisation de la relation de spécialisation et la possibilité d'interpréter facilement le modèle construit.

Dans IODA, nous choisissons de reposer sur deux représentations équivalentes d'une matrice d'interaction brute, se focalisant respectivement sur l'aspect représentation des connaissances et l'aspect ingénierie logicielle de la simulation. Nous appelons ces deux formes des matrices d'interaction *forme synthétique* et *forme étendue*.

Principes

La spécialisation est un moyen permettant de construire facilement de grandes matrices d'interaction brutes à l'aide d'un faible nombre d'éléments d'assignation. Elle transpose à une approche centrée-interactions la notion d'héritage telle qu'on la rencontre en programmation orientée-objets. Les matrices d'interaction reposant sur cette relation afin de réduire le nombre d'éléments d'assignations qu'elles contiennent sont dites sous *forme synthétique*. Une matrice d'interaction brute sous une telle forme est illustrée sur la figure 8.4(a) de la page 225 pour une simulation simple d'un écosystème.

Définition 50. ***Forme synthétique***
Une matrice d'interaction brute est dite sous **forme synthétique** si elle fait apparaître des liens de spécialisation entre familles d'agents et le moins possible d'interactions.

L'interprétation d'une matrice d'interaction n'est pas aisée lorsqu'un nombre important de relations de spécialisation entre familles d'agents apparaissent. En effet, les interactions pouvant être effectuées ou subies par un agent sont éparpillées entre les très nombreuses familles d'agents mères. Afin de faciliter cette interprétation, nous considérons dans IODA une forme équivalente à la matrice d'interaction brute sous forme synthétique, appelée *forme étendue*. Cette forme fait apparaître de manière exhaustive les interactions pouvant survenir entre toutes les familles d'agents de la simulation. La forme étendue de la simulation d'un écosystème simple mentionnée dans le paragraphe précédent est illustrée sur la figure 8.10 de la page 231. Dans le cas de la simulation de la phagocytose de virus dont la matrice d'interaction brute sous forme synthétique est décrite dans la figure 8.2, la forme étendue correspond à la matrice présente sur la figure 8.11.

Définition 51. ***Forme étendue***
La **Forme étendue** d'une matrice d'interaction brute sous forme synthétique est une matrice faisant état de manière exhaustive de toutes les interactions pouvant avoir lieu entre les différentes familles d'agents d'une simulation.

Dans la méthodologie IODA, nous prenons le parti de construire les matrices d'interaction brutes en utilisant la forme synthétique. La forme étendue y est utilisée ponctuellement afin de vérifier que les capacités d'interaction sont héritées correctement, et décrivent bien le modèle voulu. Reposer sur

Source \ Cible	∅	Aigle :: Oiseau de Proie	Faucon :: Oiseau de Proie	Rongeur :: Animal	Végétation :: Vivant
Vivant	**(Mourir)**				
Végétation :: Vivant	*(*Mourir*)* **(Se Propager)**				
Animal :: Vivant	*(*Mourir*)* **(Se Déplacer)**				
Rongeur :: Animal	*(*Mourir*)* *(*Se Déplacer*)*			**(Se Reproduire, d = 0)**	**(Manger, d = 0)**
Oiseau de Proie :: Animal	*(*Mourir*)* *(*Se Déplacer*)*			**(Manger, d = 0)**	
Aigle :: Rapace	*(*Mourir*)* *(*Se Déplacer*)*	**(Se Reproduire, d = 0)**	**(Manger, d = 0)**	*(*Manger, *d = 0)*	
Faucon :: Rapace	*(*Mourir*)* *(*Se Déplacer*)*		**(Se Reproduire, d = 0)**	*(*Manger, *d = 0)*	

FIGURE 8.10 – Matrice d'interaction brute sous forme étendue d'une simulation d'un écosystème contenant des `Aigles`, des `Faucons`, des `Rongeurs` et de la `Végétation`, dont la matrice d'interaction brute sous forme synthétique est définie sur la figure 8.4(a). Dans cette matrice, les éléments d'assignation présents explicitement dans la forme synthétique sont affichés en gras. Ceux ajoutés afin de compléter la forme étendue apparaissent en italique.

Source \ Cible	∅	Virus	Virus Rhume
Lymphocyte	**(Se Déplacer)**	**(Phagocyter, d = 0)**	*(*Phagocyter, *d = 0)*

Source \ Cible	Virus Grippe	Virus Varicelle	Virus Rougeole
Lymphocyte	*(*Phagocyter, *d = 0)*	*(*Phagocyter, *d = 0)*	*(*Phagocyter, *d = 0)*

FIGURE 8.11 – Matrice d'interaction brute sous forme étendue de la matrice d'interaction brute sous forme synthétique décrite sur la figure 8.2. Dans cette simulation, un `Lymphocyte` peut phagocyter des `Virus` tels que`Virus Rhume`, le `Virus Grippe`, le `Virus Varicelle` ou le `Virus Rougeole`. Dans la forme étendue, les éléments d'assignation présents explicitement dans la forme synthétique sont affichés en gras. Ceux ajoutés afin de compléter la forme étendue appariassent en italique. Pour des raisons de lisibilité, cette matrice d'interactions a été divisée en deux morceaux.

cette dualité dans la méthodologie permet alors à la fois de profiter du génie logiciel et de la facilité de compréhension du modèle ainsi construit. Pour profiter de cette dualité, il faut pouvoir construire la forme étendue d'une matrice d'interaction brute à partir de sa forme synthétique. Dans la section qui suit, nous décrivons le procédé général permettant d'y parvenir.

Principes de la construction d'une forme étendue

La forme étendue d'une matrice d'interaction brute est composée de deux types d'éléments d'assignations. Les premiers, qui apparaissent en gras sur la forme étendue donnée en exemple dans la figure 8.11, sont les éléments que l'on retrouve dans la forme synthétique. Nous qualifions ces éléments d'assignation d'*absolus*. Les seconds, qui apparaissent en italique sur la forme étendue donnée en exemple, sont les éléments d'assignation qui ont été déduits des éléments d'assignation *absolus*. Nous les appelons *éléments d'assignation relatifs*. Dans cette section, nous illustrons comment construire la forme étendue de la matrice d'interaction brute décrite sous forme synthétique dans la figure 8.12.

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Mouton :: Herbivore
Vivant				
Carnivore :: Vivant			(MANGER, d=1)	
Loup :: Carnivore				
Louveteau :: Loup				

FIGURE 8.12 – Forme synthétique de la matrice d'interaction brute servant d'exemple pour illustrer la construction de sa forme étendue. Pour simplifier la compréhension de l'exemple, nous ne faisons apparaître qu'un seul élément d'assignation.

La construction de la forme étendue d'une matrice d'interaction brute est un procédé itératif pouvant être automatisé, lors duquel la présence de chaque élément d'assignation dans la forme synthétique de la matrice d'interaction brute est interprétée, afin de déterminer de manière exhaustive toutes les interactions entre une famille d'agents source et une famille d'agents cible qu'elle désigne. Ce procédé consiste donc à traiter séquentiellement chaque élément d'assignation a présent dans la forme synthétique de la matrice d'interaction brute. Le traitement de a est alors formé des trois étapes qui suivent :

1. La première étape consiste à ajouter a dans la forme étendue de la matrice d'interaction brute, dans la même ligne L et la même colonne C que dans la forme synthétique. Cette étape est illustrée dans la figure 8.13 sous la forme de flèches à double traits marquées par le chiffre 1.
2. Si la colonne C est associée à une famille d'agents ayant une ou plusieurs filles, alors a est aussi ajouté à l'intersection de la ligne L et de la colonne de toutes ces familles d'agents filles. Cette étape est illustrée dans la figure 8.13 sous la forme de flèches à double traits marquées par le chiffre 2.
3. Si la ligne L est associée à une famille d'agents ayant une ou plusieurs filles, alors chaque élément d'assignation ajouté lors des étapes 1 et 2 dans la ligne L sont aussi ajoutés dans la ligne associée à chacune de ces familles d'agents filles. Cette étape est effectuée récursivement pour toutes les familles d'agents filles rencontrées. Elle est illustrée dans la figure 8.13 sous la forme de flèches à trait simple marquées par le chiffre 3.

Par exemple, dans le cas où la forme synthétique de la matrice d'interaction correspond à ce qui est présenté sur la figure 8.12, le procédé permettant de construire sa forme étendue est celui décrit sur la figure 8.13.

8.2.3 Opérateurs de spécialisation

La relation de spécialisation que nous caractérisons dans cette section ne se limite pas au simple héritage des capacités d'interaction ou de mise à jour. Nous y permettons de plus de rendre ces connaissances

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Mouton :: Herbivore
Vivant				
Carnivore :: Vivant				
Loup :: Carnivore				
Louveteau :: Loup				

(a) Première étape : construction de la ligne associée à `Vivant`

Source \ Cible	∅	Chèvre :: Herbivore	1 Herbivore	Mouton :: Herbivore
Vivant				
Carnivore :: Vivant		*(*MANGER*, d=1)*	**(MANGER, d=1)**	*(*MANGER*, d=1)*
Loup :: Carnivore		2		2
Louveteau :: Loup				

(b) Deuxième étape : construction de la ligne associée à `Carnivore`

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Mouton :: Herbivore
Vivant				
Carnivore :: Vivant		*(*MANGER*, d=1)* 3	**(MANGER, d=1)** 3	*(*MANGER*, d=1)* 3
Loup :: Carnivore		*(*MANGER*, d=1)*	*(*MANGER*, d=1)*	*(*MANGER*, d=1)*
Louveteau :: Loup				

(c) Troisième étape : construction de la ligne associée à `Loup`

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Mouton :: Herbivore
Vivant				
Carnivore :: Vivant		*(*MANGER*, d=1)*	**(MANGER, d=1)**	*(*MANGER*, d=1)*
Loup :: Carnivore		*(*MANGER*, d=1)* 3	*(*MANGER*, d=1)* 3	*(*MANGER*, d=1)* 3
Louveteau :: Loup		*(*MANGER*, d=1)*	*(*MANGER*, d=1)*	*(*MANGER*, d=1)*

(d) Quatrième étape : construction de la ligne associée à `Louveteau`

FIGURE 8.13 – Illustration du procédé permettant de construire la forme étendue d'une matrice d'interaction brute, à partir de la forme synthétique présentée sur la figure 8.12. Dans cette figure, chaque flèche décrit l'effet d'une des étapes du processus de construction de la forme étendue. Le nombre associé à chaque flèche correspond à l'étape de ce procédé ayant produit un tel résultat.

plus spécifiques à la famille d'agents fille, en autorisant d'autres opérations :
- la modification de la garde de distance d'un élément d'assignation ;
- le retrait d'éléments d'assignation permettant de retirer à un agent la capacité d'initier une interaction avec une cible particulière.

Ces deux opérateurs peuvent être utilisés pour altérer ou supprimer tout élément d'assignation présent dans la forme étendue de la matrice d'interaction.

Caractérisation des opérateurs de spécialisation

Un *opérateur de spécialisation* est un élément placé dans la forme synthétique de la matrice d'interaction brute. Il caractérise l'ajout d'un élément d'assignation (on parle alors d'opérateur d'ajout), la modification de la garde de distance d'un élément d'assignation (on parle alors d'opérateur de modification), ou la suppression pure et simple d'un élément d'assignation (on parle alors d'opérateur de retrait).

L'*opérateur d'ajout* permet d'ajouter un nouvel élément d'assignation dans la forme synthétique de la matrice d'interaction brute. Cet opérateur est utilisé implicitement dans toutes les matrices d'interaction brute sous forme synthétique que nous avons définies jusqu'à présent.

Définition 52. ***Opérateur d'ajout***
Un **opérateur d'ajout** est un opérateur figurant dans une cellule de la matrice d'interaction brute sous forme synthétique, qui caractérise l'ajout d'un élément d'assignation. Il permet donc de décrire le fait qu'un agent gagne la capacité d'initier une interaction.

Lorsqu'il est placé dans la colonne $\emptyset$ de la matrice, l'opérateur d'ajout est noté $'+(I)'$ ou $'(I)'$, où I est l'identifiant d'une interaction. Dans le cas contraire, il est noté $'+(I, d = \delta)'$ ou $'(I, d = \delta)'$, avec δ un nombre réel positif.

L'*opérateur de retrait* permet de retirer à un agent la capacité d'initier une interaction avec une cible particulière. Pour cela, cet opérateur permet de retirer tout élément d'assignation apparaissant dans la forme étendue de la matrice d'interaction brute.

Définition 53. ***Opérateur de retrait***
Un **opérateur de retrait** est un opérateur figurant dans une cellule de la matrice d'interaction brute sous forme synthétique, qui caractérise le retrait d'un élément d'assignation. Il permet donc de décrire le fait qu'un agent ne possède plus la capacité d'interagir avec une cible particulière.

Lorsqu'il est placé dans la colonne $\emptyset$ de la matrice, l'opérateur de retrait est noté $'-(I)'$, où I est l'identifiant d'une interaction. Dans le cas contraire, il est noté $'-(I, d = \delta)'$, avec δ un nombre réel positif. Bien entendu, on ne peut retirer à un agent la capacité d'initier une interaction avec une cible spécifique que si ce dernier possédait cette capacité. Par conséquent, un tel opérateur ne peut être ajouté dans une cellule de la matrice d'interaction brute sous forme synthétique que si un élément d'assignation de la forme $(I, d = \delta)$ (ou (I)) est présent dans la même cellule dans la matrice d'interaction brute sous forme étendue. Le retrait peut être utilisé de deux façons : retirer purement et simplement à un agent sa capacité à interagir avec une cible, ou restreindre l'ensemble des cibles pouvant subir une interaction particulière. Par exemple, la figure 8.14 montre comment utiliser l'opérateur de retrait pour exprimer qu'un loup peut manger tout type d'herbivore hormis les chèvres.

Source \ Cible	$\emptyset$	Herbivore	Mouton :: Herbivore	Chèvre :: Herbivore	Cerf :: Herbivore
Loup		+(MANGER, d = 1)		-(MANGER, d = 1)	

FIGURE 8.14 – Matrice d'interaction exprimant qu'un `Loup` est capable de MANGER tout `Herbivore` mis à part les `Chèvres`.

Le dernier opérateur est l'*opérateur de modification*. Il permet de modifier la garde de distance de n'importe quel élément d'assignation apparaissant dans la forme étendue de la matrice d'interaction.

Définition 54. ***Opérateur de modification***
Un **opérateur de modification** est un opérateur figurant dans une cellule de la matrice d'interaction brute sous forme synthétique, qui caractérise la modification de la garde de distance d'un élément d'assignation.

Cet opérateur ne peut pas apparaître dans la colonne $\emptyset$, puisqu'aucune garde de distance ne peut y être définie. Nous notons cet opérateur $' * (I, d = \delta \rightarrow \delta')'$, où δ' est la nouvelle garde de distance. Bien entendu, on ne peut modifier la capacité d'un agent à initier une interaction avec une cible spécifique que si ce dernier possédait cette capacité. Par conséquent, un tel opérateur ne peut être ajouté dans une cellule de la matrice d'interaction brute sous forme synthétique que si un élément d'assignation de la forme $(I, d = \delta)$ est présent dans la même cellule dans la matrice d'interaction brute sous forme étendue.

L'utilisation conjointe de ces trois opérateurs permet de spécifier des simulations non seulement par héritage, mais aussi par spécialisation. Il devient alors possible d'utiliser le schéma de conception décrit dans la figure 8.6(c) page 226. Nous illustrons ici comment pratiquer cette autre approche de la conception pour construire la matrice d'interaction brute d'une simulation reproduisant un comportement basique de clients dans un magasin.

Exemple.
Considérons une simulation reproduisant le comportement de `Clients` *et de* `Clients Aveugles` *dans un magasin. Un* `Client` *y est une entité pouvant* SE DÉPLACER, LIRE *les informations affichées sur un* `Panneau Publicitaire`, LIRE *l'étiquette d'un* `Article`, RAMASSER *un* `Article` *et* PAYER *à une* `Caisse` *les articles achetés. Dans cette simulation, les* `Clients Aveugles` *se comportent comme des* `Clients`*, hormis le fait qu'il sont incapables de* LIRE *les informations d'un* `Panneau Publicitaire`*. De plus, il ne peuvent* LIRE *les informations concernant un* `Article` *qu'en braille et donc à une distance de 0. La spécification de la matrice d'interaction brute de cette simulation se fait en deux phases.*

La première phase consiste à :

1. *établir la ligne de la matrice d'interaction brute associée la famille d'agents* `Client` *à l'aide d'opérateurs d'ajout ;*
2. *déclarer que la famille d'agents* `Client Aveugle` *spécialise la famille d'agents* `Client`*.*

La forme étendue de cette matrice (voir figure 8.15(b)) permet alors de déterminer quelles interactions peuvent être initiées par la famille d'agents `Client Aveugle`*. Cette première étape est aussi l'occasion d'identifier les éléments d'assignation devant être modifiés ou supprimés pour que les* `Clients Aveugles` *correspondent bien aux descriptions de l'énoncé.*

Source \ Cible	$\emptyset$	Panneau Publicitaire	Article	Caisse
Client	+(SE DÉPLACER)	+(LIRE, d = 5)	+(LIRE, d = 2) +(RAMASSER, d = 0)	+(PAYER, d = 0)
Client Aveugle :: Client				

(a) Forme synthétique de la matrice d'interaction brute

Source \ Cible	$\emptyset$	Panneau Publicitaire	Article	Caisse
Client	**(SE DÉPLACER)**	**(LIRE, d = 5)**	**(LIRE, d = 2)** **(RAMASSER, d = 0)**	**(PAYER, d = 0)**
Client Aveugle :: Client	*(SE DÉPLACER)*	*(LIRE, d = 5)*	*(LIRE, d = 2)* *(RAMASSER, d = 0)*	*(PAYER, d = 0)*

(b) Forme étendue de la matrice d'interaction brute

FIGURE 8.15 – Matrice d'interaction brute sous forme synthétique et sous forme étendue décrivant une simulation reproduisant le comportement de `Clients` et de `Clients Aveugles` dans un magasin. Dans cette matrice, le comportement spécifique des `Clients Aveugles` n'est pas encore décrit.

Dans une seconde phase, des opérateurs de modification et de retrait sont utilisés afin de caractériser la différence entre les familles `Client Aveugle` *et* `Client`*. Dans cet exemple, cela consiste à lui retirer la*

capacité de LIRE *des* `Panneaux Publicitaires` *en utilisant l'opérateur de retrait sur l'élément d'assignation* (LIRE, $d = 5$). *Cela consiste d'autre part à modifier la distance à partir de laquelle il est capable de* LIRE *l'étiquette des* `Articles`, *en modifiant la garde de distance de l'élément d'assignation* (LIRE, $d = 2$). *On aboutit alors à la matrice d'interaction sous forme synthétique de la figure 8.16(a). Sa forme étendue (voir figure 8.16(b)) représente bien les* `Clients Aveugles` *tels que décrits au début de cet exemple.*

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse
Client	+(SE DÉPLACER)	+(LIRE, d = 5)	+(LIRE, d = 2) +(RAMASSER, d = 0)	+(PAYER, d = 0)
Client Aveugle :: Client		-(LIRE, d = 5)	*(LIRE, d = 2 → 0)	

(a) Forme synthétique de la matrice d'interaction brute

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse
Client	**(SE DÉPLACER)**	**(LIRE, d = 5)**	**(LIRE, d = 2)** **(RAMASSER, d = 0)**	**(PAYER, d = 0)**
Client Aveugle :: Client	*(SE DÉPLACER)*		**(LIRE, d = 0)** *(RAMASSER, d = 0)*	*(PAYER, d = 0)*

(b) Forme étendue de la matrice d'interaction brute

FIGURE 8.16 – Matrice d'interaction brute sous forme synthétique et sous forme étendue décrivant une simulation reproduisant le comportement de `Clients` et de `Clients Aveugles` dans un magasin. Les opérateurs figurant dans la ligne associée à la famille d'agents `Client Aveugle` ont été déterminés en identifiant les modifications à apporter à la ligne associée à la famille d'agents `Client Aveugle` dans la figure 8.15(b) afin de parvenir à la ligne représentant les capacités d'interaction attendues affichées dans la figure 8.16(b).

Nous avons décrit ici l'usage des opérateurs de spécialisation dans un cas simple ne faisant intervenir que deux familles d'agents et une seule relation de spécialisation. Lorsque plusieurs relations de spécialisation apparaissent, la construction de la forme étendue de la matrice d'interaction brute doit prendre en compte non seulement les modifications ayant lieu dans la cellule où a été placé l'opérateur de retrait ou de modification, mais aussi la propagation de leur effet aux autres cellules de la matrice. Ce point nécessite une attention particulière, car selon la façon dont sont propagées les effets des différents opérateurs, des formes étendues très différentes peuvent être obtenues.

Propagation de l'effet des opérateurs de spécialisation

L'effet d'un opérateur de spécialisation n'est pas restreint à la cellule de la matrice d'interaction brute où il est placé : il se propage aux familles d'agents filles de sa source et de sa cible. Dans le cas de l'opérateur d'ajout, cette propagation consiste à ajouter des éléments d'assignation dans la forme étendue de la matrice d'interaction.

Tout comme l'opérateur d'ajout, l'opérateur de suppression et l'opérateur de modification ont un effet se propageant parmi les familles d'agents filles. Le traitement de ces opérateurs pose toutefois problème, car l'ordre dans lequel ils sont évalués influe grandement sur la forme étendue obtenue de la matrice d'interaction brute et donc sur la sémantique du modèle créé. Pour préserver la cohérence du modèle, le procédé permettant de propager l'effet des opérateurs doit en particulier s'assurer du traitement de deux problèmes, décrits par les matrices d'interaction brutes sous forme synthétique décrites dans les figures 8.17(a) et 8.17(b).

Caractérisation de problèmes rencontrés. Lorsque plusieurs familles d'agents entretiennent une relation de spécialisation, certains opérateurs peuvent avoir un effet entrant en concurrence. Par exemple, la figure 8.17(a) décrit qu'un `Carnivore` est capable de MANGER des `Herbivores` à une distance de 0, sauf dans le cas des `Herbivores Malade` dont la faible mobilité permet aux `Carnivores` de les chasser puis les MANGER à plus grande distance (une distance de 5). Dans cette matrice, un LOUP est un `Carnivore`

Source \ Cible	∅	Herbivore	Herbivore Malade :: Herbivore
Carnivore		+(MANGER, d = 0)	*(MANGER, d=0 → 5)
Loup :: Carnivore		*(MANGER, d=0 → 1)	?

(a) Premier problème rencontré : la collision de modifications. L'opérateur "*" est utilisé afin de modifier la garde de distance d'un élément d'assignation. Cette modification est valable pour tous les agents spécialisant la famille d'agent source et la famille d'agent cible de l'élément d'assignation modifié. Le problème est ici de déterminer si les `Loups` mangent les `Herbivores malades` à une distance de 1 ou à une distance de 5.

Source \ Cible	∅	Loup
Loup	+(SE DÉPLACER) +(MOURIR)	+(SE REPRODUIRE, d=0)
Loup Agressif :: Loup		+(ATTAQUER, d=1) *(SE REPRODUIRE, d=4)
Loup Stérile :: Loup		-(SE REPRODUIRE, d=0)
Loup Agressif et Stérile :: Loup Agressif Loup Stérile		?

(b) Deuxième problème rencontré : le retrait d'une interaction dans une famille d'agents mère qui est présente dans une autre famille d'agents mère. Dans cette figure, le problème consiste à déterminer si l'interaction SE REPRODUIRE est présente ou non dans la ligne associée à la famille d'agents `Loup Agressif et Stérile`

FIGURE 8.17 – Illustration de deux problèmes de cohérence devant être pris en compte lors de la propagation de l'effet des opérateurs de spécialisation.

particulier, suffisamment agile et rapide pour chasser et MANGER des `Herbivores` à une plus grande distance que la plupart des `Carnivores` (une distance de 1). Dans ce cas, deux valeurs peuvent être attribuées à la distance à laquelle un LOUP peut MANGER des `Herbivores Malades`. Se pose alors le problème du choix de la valeur à prendre en compte.

La solution à ce problème se trouve dans l'interprétation de la relation de spécialisation. La ligne de la matrice d'interaction associée à la famille d'agents CARNIVORE s'interprète en « Un carnivore est un agent qui est capable de manger des herbivores à une distance de 0, sauf les herbivores malades, qu'il est capable de manger à une distance d'au plus 5 ». L'opérateur de modification $*(Manger, d = 0 \rightarrow 1)$ situé à l'intersection de la ligne associée à LOUP et de la colonne associée à HERBIVORE se lit « Un loup est un carnivore particulier qui mange tout herbivore à une distance d'au plus 1 ». **Cette seconde interprétation est plus spécifique que celle fournie pour les CARNIVORES.** La solution nous étant apparue comme la plus appropriée est alors :

- d'ignorer la modification induite par l'opérateur $*(Manger, d = 0 \rightarrow 5)$ situé dans la ligne associée à la famille d'agents HERBIVORE ;
- de prendre en compte la modification la plus spécifique à la famille d'agents LOUP.

Dans ce cas, l'élément d'assignation présent implicitement dans la cellule marquée par point d'interrogation dans la figure 8.17(a) est donc $(\text{MANGER}, d = 1)$.

Le second problème est aussi relatif à la collision de la propagation de deux opérateurs de spécialisation. Il est lié au fait qu'un opérateur de retrait et un opérateur de modification peuvent avoir un effet sur un même élément d'assignation dans deux familles d'agents différentes. Dans ce cas, si ces deux familles d'agents sont spécialisées par une même famille d'agents fille, il faut pouvoir déterminer si l'élément d'assignation est présent ou pas dans la ligne associée à la famille d'agents fille. La figure 8.17(b) fournit un exemple de ce problème à l'aide d'une matrice d'interaction brute sous forme synthétique. Cette matrice décrit trois déclinaisons différentes du comportement d'un `Loup` : les `Loups Agressifs`, les `Loups Stériles` et les `Loup Agressifs et Stériles`. Les `Loups Agressifs` sont des `Loups` particuliers

pouvant ATTAQUER d'autres `Loups` et chercher plus loin leurs partenaires de reproduction. Les `Loups Stériles` sont des `Loups` particuliers n'étant pas capables de se reproduire. Enfin, les `Loups Agressifs et Stériles` cumulent les spécificités des `Loups Agressifs` et des `Loups Stériles`. Ce diagramme d'héritage amène à un paradoxe : un `Loup Agressif et Stérile` est un `Loup Agressif` particulier, et, à ce titre, il doit être capable de SE REPRODUIRE à une distance d'au plus 4. Il est toutefois aussi un `Loup Stérile` particulier et doit donc ne pas pouvoir SE REPRODUIRE.

La solution la plus appropriée à ce paradoxe pour cet exemple consiste à ignorer les modifications de garde de distance induites par la famille d'agents `Loup Agressif` et à ne prendre en compte que le retrait effectué par la famille d'agents `Loup Stérile`. **L'opérateur de suppression serait donc plus prioritaire que l'opérateur de modification.**

Les solutions envisagées ici ne dépendent pas de la sémantique de l'interaction, mais de la position des interaction dans la forme étendue de la matrice d'interaction brute, ainsi que de la relation d'ordre établie entre les opérateurs. La construction de la forme étendue d'une matrice d'interaction brute sous forme synthétique peut donc être automatisée.

Principes de la propagation de l'effet des opérateurs de spécialisation. Nous proposons dans cette section un procédé automatique permettant de passer d'une forme synthétique à une forme étendue d'une matrice d'interaction brute. Pour cela, nous faisons le choix de suivre les deux principes évoqués dans le paragraphe précédent :

1. **toujours prendre en compte l'effet de l'opérateur de modification le plus spécifique à la famille d'agents F considérée.** Il consiste à préférer les modifications apportées directement par un opérateur de modification figurant dans la ligne associée à F (par exemple l'opérateur $*(\text{MANGER}, d = 0 \rightarrow 1)$ dans la figure 8.17(b) si F est la famille d'agents `Loup`) plutôt que les modifications apportées par un opérateur de modification présent dans la ligne d'une famille d'agents mère de F (par exemple l'opérateur $*(\text{MANGER}, d = 0 \rightarrow 5)$ présent dans la ligne associée à la famille d'agents `Carnivore`). Il consiste aussi à préférer les modifications apportées par un opérateur de modification prenant directement pour cible une famille d'agents C plutôt que les modifications induites par un opérateur de modification prenant pour cible une famille d'agent mère de C.
2. **toujours préférer les modifications induites par un opérateur de retrait plutôt que les modifications induites par un opérateur de modification.** L'effet des opérateurs de retrait doit donc être appliqué en dernier.

Automatisation de la construction d'une forme étendue. L'automatisation du procédé permettant de construire la forme étendue d'une matrice d'interaction à partir de sa forme synthétique repose en premier lieu sur le principe de la propagation verticale et la propagation horizontale de l'effet d'un opérateur de spécialisation. Dans ce paragraphe, nous posons :

- a un élément d'assignation ;
- *op* un opérateur de spécialisation permettant d'ajouter, de modifier, ou de supprimer un élément d'assignation a ;
- S la famille d'agents associée à la ligne de la matrice d'interaction brute sous forme synthétique où figure *op* ;
- C la famille d'agents associée à la colonne où se situe a.

La *propagation horizontale* d'un opérateur *op* consiste à appliquer son effet uniquement dans la ligne associée à S dans la forme étendue de la matrice d'interaction brute. Ce type de propagation apparaît sous la forme de flèches à double trait dans la figure 8.19. Si *op* est un opérateur d'ajout, l'effet consiste à ajouter l'élément d'assignation a à l'intersection de la ligne associée à S et de la colonne associée à C . De plus, si a n'est pas un élément d'assignation dégénéré, la propagation horizontale de son effet consiste à dupliquer l'élément d'assignation a dans chaque colonne associée à une famille d'agents spécialisant C. Si *op* est un opérateur de retrait, son effet consiste à retirer l'élément d'assignation a de la cellule située à l'intersection de la ligne associée à S et de la colonne associée à C . De plus, si a n'est pas un élément d'assignation dégénéré, la propagation horizontale de son effet consiste à aussi retirer l'élément d'assignation a dans chaque colonne associée à toute famille d'agents spécialisant C. Enfin, si *op* est un opérateur de modification, son effet consiste à modifier la garde de distance de l'élément d'assignation

a de la cellule située à l'intersection de la ligne associée à S et de la colonne associée à C, ainsi que de toutes les occurrences de a dans une colonne associée à une famille d'agents spécialisant C.

Dans les trois cas, nous qualifions de dépendances de *op* l'ensemble des éléments d'assignation ayant été ajoutés, modifiés ou supprimés lors de la propagation horizontale de l'opérateur *op*.

La *propagation verticale* d'un opérateur *op* consiste à appliquer l'effet de *op* dans la ligne associée à chaque famille d'agents spécialisant S, ainsi que dans la ligne de toutes les familles d'agents spécialisant transitivement S. La propagation verticale de l'effet de *op* dans une de ces lignes consiste à y reproduire l'effet qu'a eu *op* dans la ligne associée à S. Dans le cas d'un opérateur d'ajout, cela consiste à ajouter une copie dans la ligne de chaque élément d'assignation apparaissant dans les dépendances de *op*. Dans le cas d'un opérateur de retrait, cela consiste à retirer de la ligne tous les éléments d'assignation présents dans les dépendances de *op*. Enfin, dans le cas d'un opérateur de modification, cela consiste à modifier la garde de distance de tous les éléments d'assignation de la ligne présents dans les dépendances de *op*. Ce type de propagation apparaît sous le forme de flèches à simple trait dans la figure 8.19.

La construction de la forme étendue de la matrice d'interaction se fait alors en trois phases :

1. **propager l'effet de tous les opérateurs d'ajout contenus dans la forme synthétique de la matrice d'interaction brute.** Cette propagation est illustrée plus en détails dans la section 8.2.2.
2. **propager l'effet des opérateurs de modification dans un ordre précis, afin que la modification la plus spécifique soit prise en compte.** Pour cela, l'effet d'un opérateur de modification présent dans une ligne associée à une famille d'agents F n'est propagé que si l'effet des opérateurs de modification présents dans la ligne associée aux les familles d'agents mères de F a déjà été propagé. La propagation de l'effet de la modification repose sur le même principe que le retrait : la propagation horizontale dans la ligne où l'opérateur apparaît, puis la modification par propagation verticale de tous les éléments d'assignation ayant été modifiés dans cette ligne.
3. **propager l'effet de tous les opérateurs de retrait**, en effectuant dans un premier temps une propagation horizontale dans la ligne où l'opérateur apparaît, puis par le retrait par propagation verticale de tous les éléments d'assignation ayant été retirés de cette ligne.

Nous illustrons ce procédé de construction dans la figure 8.19, pour une matrice d'interaction brute sous forme synthétique telle que décrite dans la figure 8.18.

Cible / Source	∅	Chèvre :: Herbivore	Herbivore	Ovins :: Herbivore	Mouton :: Ovins
Carnivore			+(MANGER, d=1)		
Loup :: Carnivore			*(MANGER, *d=1→2*)	-(MANGER, d=1)	
Louveteau :: Loup					

FIGURE 8.18 – Matrice d'interaction brute sous forme synthétique utilisée pour illustrer la propagation de l'effet des opérateurs. On exprime ici que les `Carnivores` peuvent manger des `Herbivores`. Il est de plus exprimé que les `Loups` peuvent manger des `Herbivores` à une distance plus grande que les autres `Carnivores` et qu'ils ne peuvent pas manger d'`Ovins`.

Limites de l'automatisation de la construction d'une forme étendue. Les principes que nous décrivons ici permettent d'automatiser la transformation d'une forme synthétique vers une forme étendue. Cette transformation ne peut toutefois pas être automatique dans tous les cas. En effet, puisque la spécialisation permet de pratiquer l'héritage "classique" des systèmes multi-agents, elle souffre aussi des mêmes limites. L'opérateur de modification permet de "surcharger" la garde de distance d'un élément d'assignation. Il est donc sujet au même problème que le polymorphisme des méthodes dans le contexte de l'héritage multiple : si une famille d'agents spécialise deux familles d'agents différentes, appliquant toutes deux un opérateur de modification au même élément d'assignation, une ambiguïté survient dans le modèle. La figure 8.20 illustre ce type d'ambiguïté.

+(MANGER, d=1)

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Ovins :: Herbivore	Mouton :: Ovins
Carnivore		*(MANGER, d=1)*	**(MANGER, d=1)**	*(MANGER, d=1)*	*(MANGER, d=1)*
Loup :: Carnivore		*(MANGER, d=1)*	*(MANGER, d=1)*	*(MANGER, d=1)*	*(MANGER, d=1)*
Louveteau :: Loup		*(MANGER, d=1)*	*(MANGER, d=1)*	*(MANGER, d=1)*	*(MANGER, d=1)*

(a) Première étape : application de l'effet des opérateurs d'ajout

***(MANGER, d=1→2)**

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Ovins :: Herbivore	Mouton :: Ovins
Carnivore		*(MANGER, d=1)*	**(MANGER, d=1)**	*(MANGER, d=1)*	*(MANGER, d=1)*
Loup :: Carnivore		~~*(MANGER, d=1)*~~ *(MANGER, d=2)*	~~*(MANGER, d=1)*~~ **(MANGER, d=2)**	~~*(MANGER, d=1)*~~ *(MANGER, d=2)*	~~*(MANGER, d=1)*~~ *(MANGER, d=2)*
Louveteau :: Loup		~~*(MANGER, d=1)*~~ *(MANGER, d=2)*	~~*(MANGER, d=1)*~~ *(MANGER, d=2)*	~~*(MANGER, d=1)*~~ *(MANGER, d=2)*	~~*(MANGER, d=1)*~~ *(MANGER, d=2)*

(b) Deuxième étape : application de l'effet des opérateurs de modification

-(MANGER, d=2)

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Ovins :: Herbivore	Mouton :: Ovins
Carnivore		*(MANGER, d=1)*	**(MANGER, d=1)**	*(MANGER, d=1)*	*(MANGER, d=1)*
Loup :: Carnivore		*(MANGER, d=2)*	**(MANGER, d=2)**	~~*(MANGER, d=2)*~~	~~*(MANGER, d=2)*~~
Louveteau :: Loup		*(MANGER, d=2)*	*(MANGER, d=2)*	~~*(MANGER, d=2)*~~	~~*(MANGER, d=2)*~~

(c) Troisième étape : application de l'effet des opérateurs de retrait

Source \ Cible	∅	Chèvre :: Herbivore	Herbivore	Ovins :: Herbivore	Mouton :: Ovins
Carnivore		*(MANGER, d=1)*	**(MANGER, d=1)**	*(MANGER, d=1)*	*(MANGER, d=1)*
Loup :: Carnivore		*(MANGER, d=2)*	**(MANGER, d=2)**		
Louveteau :: Loup		*(MANGER, d=2)*	*(MANGER, d=2)*		

(d) Apparence finale de la forme étendue de la matrice d'interaction brute

FIGURE 8.19 – Illustration des trois étapes du procédé permettant de propager l'effet d'un opérateur de modification dans une matrice d'interaction brute, ainsi que du résultat de la propagation. La flèche à double trait (resp. trait simple) représente la propagation horizontale (resp. verticale) de l'effet de l'opérateur.

Source \ Cible	∅	E
A		$+(\mathrm{I}, d = \delta)$
B :: A		$*(\mathrm{I}, d = \delta \rightarrow \delta'_1)$
C :: A		$*(\mathrm{I}, d = \delta \rightarrow \delta'_2)$
D :: B, C		?

FIGURE 8.20 – Illustration d'un problème lié à la spécialisation de plusieurs familles d'agents issu du problème du losange en héritage multiple. Dans cette figure, la détermination de la garde de distance dans la ligne associée à la famille d'agents `D` ne peut être automatisée. Elle requiert l'intervention du concepteur.

Contrairement aux deux autres problèmes mentionnés dans le paragraphe intitulé « Caractérisation de problèmes rencontrés » dans la page 236, ce dernier problème ne peut être résolu automatiquement. En effet, la solution des deux problèmes évoqués précédemment se fondent soit sur l'interprétation d'opérateurs de même nature se trouvant dans des situations différentes (voir figure 8.17(a)), soit sur des opérateurs de natures différentes se trouvant dans une situation similaire (voir figure 8.17(b)). Dans ce dernier cas, il n'y a aucune différence d'opérateur, ou de situation.

Une solution de ce problème lié à l'héritage multiple serait de **détecter automatiquement les collisions** dans l'implémentation de la méthodologie (par exemple JEDI-BUILDER) et **demander au concepteur de choisir** parmi différentes alternatives la garde de distance utilisée. Ce cas particulier de la spécialisation de plusieurs familles d'agents, où des modifications contradictoires sont fournies à un même élément d'assignation, n'est pas étudié dans le cadre de cette thèse.

Exploitation de la spécialisation dans le simulateur

Jusqu'à présent, nous nous sommes focalisés sur la spécification de la relation de spécialisation entre les familles d'agents d'une simulation et sur les moyens permettant de réutiliser tout ou partie du modèle d'une famille d'agents. Puisque l'introduction de la relation de spécialisation a amené à changer la structure de la matrice d'interaction brute, les algorithmes de simulation décrits hors du cadre de la spécialisation peuvent être devenus obsolètes. Dans cette section, nous étudions cette question et nous déterminons différents moyens permettant d'implémenter la relation de spécialisation.

La différence principale entre les algorithmes présentés dans le chapitre 4 et les algorithmes prenant en compte la relation de spécialisation réside dans la façon de mesurer le potentiel d'interaction d'un agent. En effet, là on n'avions que la matrice d'interaction brute, nous avons maintenant deux formes pour cette matrice : une forme synthétique et une forme étendue. Deux problèmes doivent alors être résolus :

1. déterminer comment mesurer le potentiel d'interaction d'un agent (*i.e.* l'ensemble des interactions qu'il est susceptible d'initier avec les agents de son voisinage) à l'aide des formes étendue et synthétique d'une matrice d'interaction brute ;
2. déterminer :
   - à quelle forme correspond la ligne de la matrice d'interaction brute du modèle d'un agent ;
   - comment cette ligne est en pratique implémentée dans un simulateur.

Mesure du potentiel d'interaction La forme synthétique de la matrice d'interaction permet de factoriser le code des agents. Toutefois, pour qu'il puisse exécuter son processus comportemental, un agent doit avoir la connaissance de toutes les interactions qu'il est susceptible d'initier, afin de pouvoir construire son potentiel d'interaction. La forme étendue de la matrice d'interaction brute fait état de toutes ces interactions. Elle peut donc être utilisée pour construire le comportement des agents, selon les principes énoncés dans l'algorithme 20. Cet algorithme fonctionne de la même manière que l'algorithme décrit dans le chapitre 4 en se basant toutefois sur la forme étendue de la matrice d'interaction brute.

Implémentation d'une ligne de la matrice d'interaction brute. L'algorithme présenté précédemment repose sur la forme étendue de la matrice d'interaction, alors que la spécification d'une simulation repose sur sa forme synthétique. Trois approches peuvent implémenter ces matrices et en attribuer les lignes aux familles d'agents.

La première approche consiste à attribuer à chaque famille d'agents leur ligne de la forme synthétique de la matrice d'interaction. À chaque fois que l'ordonnanceur de la simulation donne la parole à l'agent, la forme étendue de la matrice d'interaction est déduite de la ligne associée à la famille d'agents, ainsi que de la ligne associée à ses diverses familles d'agents mères. Cette approche est peu efficace en termes de temps de calculs, mais permet de profiter de la factorisation de la description des lignes de la matrice.

La seconde approche consiste à attribuer à chaque famille d'agents leur ligne de la forme étendue de la matrice d'interaction. Dans ce cas, la simulation fonctionne exactement comme énoncé dans le chapitre 4. Cette approche est très efficace en termes de temps de calculs, mais la factorisation de la description des différentes lignes de la matrice d'interaction brute disparaît. La matrice d'interaction brute sous forme synthétique n'est alors qu'un moyen utilisé dans la méthodologie afin de faciliter la

Algorithme 20 : Algorithme de calcul du potentiel d'interaction d'un agent nommé x, dans le cas où la relation de spécialisation peut être utilisée. Dans cet algorithme, nous notons $\mathcal{M}^{etendue}$ la matrice d'interaction sous forme étendue et $\mathcal{M}^{etendue}(S, C)$ l'ensemble des éléments d'assignation ayant la famille d'agents S pour source et la famille d'agents C pour cible.

potentiel d'interaction(x)
début
  $\mathcal{R} \Leftarrow \emptyset$;
  $\mathcal{S} \Leftarrow$ famille(x);
  % Recensement des tuples contenant des interactions de multicast
  pour tous les $\mathcal{F} \in \mathbb{F}$ **faire**
    pour tous les $a \in \mathcal{M}^{etendue}(\mathcal{S}, \mathcal{F})$ **faire**
      si $card(\mathcal{I}(a)) = (1, *)$ **alors**
        $\mathcal{T} \Leftarrow \emptyset$;
        pour tous les $y \in \mathcal{V}(x)$ **faire**
          si $accepterCible\big(\mathcal{I}(a)\big)\big(x, y\big)$ ***et*** $environnement.distance(x, y) \leq dist(a)$ **alors**
            $\mathcal{T} \Leftarrow \mathcal{T} \cup \{y\}$;
        si $(a, x, \mathcal{T})$ *est réalisable* **alors**
          $\mathcal{R} \Leftarrow \mathcal{R} \cup \{(a, x, \mathcal{T})\}$;
  % Recensement des tuples contenant des interactions individuelles
  pour tous les $y \in \mathcal{V}(x)$ **faire**
    pour tous les $a \in \mathcal{M}^{etendue}\Big(famille(x), famille(y)\Big)$ **faire**
      si (a, x, y) *est réalisable* **alors**
        $\mathcal{R} \Leftarrow \mathcal{R} \cup \{(a, x, y)\}$;
  % Recensement des tuples contenant des interactions dégénérées
  pour tous les $a \in \mathcal{M}\Big(famille(x), \emptyset\Big)$ **faire**
    si (a, x) *est réalisable* **alors**
      $\mathcal{R} \Leftarrow \mathcal{R} \cup \{a, x\}$;
  retourner $\mathcal{R}$;
fin

conception de matrices de taille conséquente. Elle n'apparaît pas dans l'implémentation. Il s'agit de l'approche implémentée actuellement dans JEDI-BUILDER.

La dernière approche consiste à attribuer à chaque famille d'agents leur ligne de la forme synthétique de la matrice d'interaction. Toutefois, au lieu de réévaluer la ligne de la forme étendue de la matrice d'interaction brute à chaque sollicitation du processus comportemental de l'agent, cette approche construit la ligne de la forme étendue de la matrice d'interaction brute lors de l'initialisation de la simulation. Cette dernière approche fournit un compromis cumulant à la fois généricité et efficacité en termes de temps de calcul, et constitue donc l'approche à privilégier lors de l'implémentation de la méthodologie. Actuellement, cette approche n'est pas implémentée dans JEDI et JEDI-BUILDER par manque de temps.

8.3 Spécialisation et modèle de sélection d'interaction

La section précédente a décrit comment utiliser la relation de spécialisation afin de construire la matrice d'interaction brute et la matrice de mise à jour. Un agent ne se limite toutefois pas à ces deux matrices. Pour qu'il puisse être implémenté et qu'il puisse participer à une simulation, un agent doit aussi définir entre autres son modèle de sélection d'interaction et sa matrice de mise à jour ordonnée. La matrice d'interaction raffinée et la matrice de mise à jour ordonnée ont des structures similaires. Les propos que nous tenons dans cette section concernant la matrice d'interaction raffinée s'appliquent donc aussi à la matrice de mise à jour ordonnée.

Dans cette section, nous nous intéressons plus particulièrement au modèle réactif de sélection d'interaction de IODA.

8.3.1 Principes

Le modèle réactif de sélection d'interaction que nous considérons ici repose sur le concept de matrice d'interaction raffinée défini dans le chapitre 3. Cette matrice attribue une priorité à chaque élément d'assignation présent dans la matrice d'interaction brute. Si l'on se place dans le cas où la spécialisation peut être utilisée, la matrice d'interaction peut prendre soit une forme synthétique, soit une forme étendue. Afin de prendre en compte la spécialisation dans le modèle réactif de sélection d'interaction, il faut donc, dans un premier temps, identifier sur quelle forme de la matrice d'interaction brute se baser.

Dans le modèle réactif, les priorités sont utilisées pour déterminer quel tuple présent dans le potentiel d'interaction représente l'interaction initiée par l'agent. Puisque, dans le cas où l'on utilise la spécialisation, la mesure du potentiel d'interaction se base sur la forme étendue de la matrice d'interaction brute, la spécification minimale permettant de définir le comportement d'un agent serait d'associer une priorité à chaque élément d'assignation présent dans la forme étendue de la matrice d'interaction brute. Toutefois, attribuer des priorités dans une matrice faisant apparaître de manière exhaustive toutes les interactions pouvant survenir dans une simulation peut s'avérer fastidieux.

Pour remédier à ce problème, la matrice d'interaction raffinée possède elle aussi une forme synthétique et une forme étendue.

Formes synthétique et étendue La forme étendue de la matrice d'interaction raffinée attribue une priorité à chaque élément d'assignation présent dans la forme étendue de la matrice d'interaction brute. Tout comme dans le cas de la matrice d'interaction brute, l'intérêt de la forme étendue de la matrice d'interaction raffinée est double :

- Elle permet d'interpréter la forme synthétique de la matrice d'interaction raffinée. Elle pallie donc au problème de dispersion des interactions dans les familles d'agents mères.
- Elle sert de support à l'implémentation. En effet, la forme étendue correspond à la matrice d'interaction raffinée utilisée dans les algorithmes du cœur de IODA décrits dans le chapitre 4. Il est donc possible de réutiliser les algorithmes de simulation du cœur de IODA sans interprétation spécifique à la spécialisation.

La forme synthétique de la matrice d'interaction raffinée repose sur des principes identiques à la forme synthétique d'une matrice d'interaction brute. Ses cellules contiennent des opérateurs décrivant chacun l'attribution d'une priorité à un élément d'assignation pour une famille d'agents source et cible

particulières. L'effet de cette attribution se propage alors de la même manière que l'effet d'un opérateur de modification dans la matrice d'interaction brute.

Opérateur d'attribution de priorité Dans le cas d'une matrice d'interaction raffinée, seul un opérateur appelé *opérateur d'attribution de priorité* est utilisé. Cet opérateur permet d'associer une priorité à un élément d'assignation présent dans la forme étendue de la matrice d'interaction brute.

Définition 55. ***Opérateur d'attribution de priorité***
Un **opérateur d'attribution de priorité** est un opérateur de spécialisation figurant dans une cellule de la forme synthétique de la matrice d'interaction raffinée, qui caractérise l'attribution d'une priorité à un élément d'assignation.

Lorsqu'il est placé dans la colonne $\emptyset$ de la matrice, il est noté $!(I, p = \rho)$ ou $(I, p = \rho)$, où ρ est la priorité attribuée à l'élément d'assignation (I) et I est l'identifiant d'une interaction dégénérée. Dans le cas contraire, il est noté $!(I, d = \delta, p = \rho)$ ou $(I, d = \delta, p = \rho)$, où ρ est la priorité attribuée à l'élément d'assignation $(I, d = \delta)$, I est l'identifiant d'une interaction qui n'est pas dégénérée et δ est un nombre réel positif.

Pour qu'un opérateur puisse figurer dans la forme synthétique de la matrice d'interaction raffinée, l'opérateur doit attribuer une priorité à un élément d'assignation existant dans la matrice d'interaction brute. Par conséquent, un opérateur $!(I, d = \delta, p = \rho)$ se situe à l'intersection d'une ligne associée à une famille d'agents S et de la colonne associée à une famille d'agents C dans la forme synthétique de la matrice d'interaction raffinée uniquement si l'élément d'assignation $(I, d = \delta)$ est présent dans la cellule à l'intersection de la ligne associée à S et de la colonne associée à C dans la forme étendue de la matrice d'interaction brute. Il en va de même pour l'opérateur $!(I, p = \rho)$ dans la colonne $\emptyset$.

L'utilisation de cet opérateur est suffisante pour construire une matrice d'interaction raffinée supportant les schémas de conception décrits dans la figure 8.6. Nous illustrons ci-après comment pratiquer cette approche de conception pour construire une matrice d'interaction raffinée pour une simulation reproduisant le comportement basique de clients dans un magasin, dont la matrice d'interaction brute est décrite dans la figure 8.15 page 235.

Exemple.
Considérons à nouveau une simulation reproduisant le comportement de `Clients` *et* `Clients Aveugles` *dans un magasin décrite dans la figure 8.15 page 235. Pour illustrer l'usage des priorités, nous introduisons dans cet exemple une nouvelle famille d'agents* `Caisse Invalides`*, qui est une* `Caisse` *particulière refusant d'encaisser le paiement d'un* `Client` *non aveugle, si jamais un* `Client Aveugle` *se situe à proximité.*

Dans cette simulation, si un `Client` *a fini ses achats, alors il cherche ne priorité à* PAYER *ses achats en* `Caisse`*. On attribue donc à l'élément d'assignation* (PAYER, $d = 0$) *la priorité maximale dans la ligne de* `Client` *(en l'occurrence 10 dans cet exemple). Si ses achats ne sont pas terminés, alors le* `Client` SE DÉPLACE *dans le magasin tant qu'il ne rencontre pas de* `Panneaux Publicitaires` *ou d'*`Articles`*. On attribue donc une priorité minimale à l'élément d'assignation* (SE DÉPLACER) *(en l'occurrence 0). Si le* `Client` *rencontre un* `Panneau Publicitaire`*, alors il prend connaissance des informations qu'il affiche en effectuant l'interaction* LIRE*. Si le* `Client` *a déjà lu ce* `Panneau Publicitaire` *et qu'il aperçoit un* `Article`*, alors il* LIT *l'étiquette de l'*`Article` *s'il ne l'a pas encore lue. Il rassemble ainsi des informations sur l'article, par exemple si l'*`Article` *fait partie d'une promotion. Dans le cas contraire, il peut décider de* RAMASSER *l'*`Article` *selon des critères qui lui sont propres, définis dans le déclencheur de l'interaction. On aboutit donc à la relation d'ordre :* (LIRE, $d = 5$) *prenant pour cible* `Panneau Publicitaire` $>$ (LIRE, $d = 2$) *prenant pour cible* `Article` $>$ (RAMASSER, $d = 0$) *prenant pour cible* `Article` *. Nous leur attribuons donc respectivement les priorités 3, 2 et 1. La forme synthétique de la matrice d'interaction raffinée faisant état de ces priorités est résumée sur la figure 8.21(a).*

Dans cette simulation, un `Client Aveugle` *se comporte comme un* `Client`*, mis à part le fait qu'il préférera aller* PAYER *ses achats dans une* `Caisse Invalides` *plutôt que dans une* `Caisse` *classique. Une priorité* 11 *est donc attribuée à l'élément d'assignation* (PAYER, $d = 0$) *se situant dans la colonne associée à la famille d'agents* `Caisse Invalides`*. La forme synthétique de la matrice d'interaction raffinée faisant état de ces priorités est résumée sur la figure 8.22(a).*

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Caisse Invalides :: Caisse
Client	!(Se Déplacer, p = 0)	!(Lire, d = 5, p = 3)	!(Lire, d = 2, p = 2) !(Ramasser, d = 0, p = 1)	!(Payer, d = 0, p = 10)	
Client Aveugle :: Client					

(a) Forme synthétique de la matrice d'interaction raffinée

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Caisse Invalides :: Caisse
Client	**(Se Déplacer, p = 0)**	**(Lire, d = 5, p = 3)**	**(Lire, d = 2, p = 2)** **(Ramasser, d = 0, p = 1)**	**(Payer, d = 0, p = 10)**	*(Payer, d = 0, p = 10)*
Client Aveugle :: Client	*(Se Déplacer, p = 0)*		*(Lire, d = 0, p = ?)* *(Ramasser, d = 0, p = 1)*	*(Payer, d = 0, p = 10)*	*(Payer, d = 0, p = 10)*

(b) Forme étendue de la matrice d'interaction raffinée

FIGURE 8.21 – Matrice d'interaction raffinée sous forme synthétique (a) et sous forme étendue (b) d'une simulation de magasin. Cette simulation reproduit le comportement de `Clients` et de `Clients Aveugles`. Dans ces matrices, le comportement spécifique des `Clients Aveugles` n'est pas encore décrit.
L'opérateur !(Lire, $d = 2, p = 2$) défini pour la famille `Client` ne peut modifier la priorité que de l'élément d'assignation (Lire, $d = 2$). Or, dans la matrice d'interaction brute de cette simulation, un opérateur de modification est utilisé pour faire passer la garde de distance de l'élément d'assignation (Lire, $d = 2$) à 0. Aucune priorité n'est donc attribuée à l'élément d'assignation (Lire, $d = 0$) de la ligne associée à la famille d'agents `Clients Aveugles`. Sa priorité est donc non définie et est représentée par un point d'interrogation.

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Caisse Invalides :: Caisse
Client	!(Se Déplacer, p = 0)	!(Lire, d = 5, p = 3)	!(Lire, d = 2, p = 2) !(Ramasser, d = 0, p = 1)	!(Payer, d = 0, p = 10)	
Client Aveugle :: Client			!(Lire, d = 0, p = 2)		!(Payer, d = 0, p = 11)

(a) Forme synthétique de la matrice d'interaction raffinée

Source \ Cible	∅	Panneau Publicitaire	Article	Caisse	Caisse Invalides :: Caisse
Client	**(Se Déplacer, p = 0)**	**(Lire, d = 5, p = 3)**	**(Lire, d = 2, p = 2)** **(Ramasser, d = 0, p = 1)**	**(Payer, d = 0, p = 10)**	*(Payer, d = 0, p = 10)*
Client Aveugle :: Client	*(Se Déplacer, p = 0)*		*(Lire, d = 0, p = 2)* *(Ramasser, d = 0, p = 1)*	*(Payer, d = 0, p = 10)*	**(Payer, d = 0, p = 11)**

(b) Forme étendue de la matrice d'interaction raffinée

FIGURE 8.22 – Matrice d'interaction brute sous forme synthétique et sous forme étendue décrivant le comportement de `Clients` et de `Clients Aveugles` dans un magasin.

La forme étendue de ces deux matrices est fournie dans les figures 8.21(b) et 8.22(b).

La spécification minimale d'une telle matrice consiste à ajouter un opérateur d'attribution de priorité pour chaque opérateur d'ajout apparaissant dans la forme synthétique de la matrice d'interaction brute, afin de caractériser la priorité par défaut d'une première moitié des éléments d'assignation. Ce principe est illustré par les opérateurs d'attribution de priorité présents dans la ligne associée à la famille d'agents `Client` dans la figure 8.22(a). La seconde moitié des priorités par défaut est attribuée aux éléments d'assignation en ajoutant un opérateur d'attribution de priorité pour chaque opérateur de modification appraissant dans la forme synthétique de la matrice d'interaction brute. Ce principe a abouti à l'ajout de l'opérateur !(LIRE, $d = 0, p = 2$) dans la ligne associée à la famille d'agents `Client Aveugle` dans la figure 8.22(a). Ensuite, d'autres opérateurs peuvent être ajoutés afin de rendre le comportement de l'agent plus spécifique. Ce principe est par exemple utilisé dans la figure 8.22(a) pour spécifier que PAYER a une priorité plus forte si l'interaction prend pour cible une `Caisse Invalides`.

8.3.2 Construction d'une forme étendue et implémentation du simulateur

La construction de la forme étendue d'une matrice d'interaction raffinée à partir de sa forme synthétique repose sur le même principe que l'application de l'effet des opérateurs de modification dans la matrice d'interaction brute.

La construction de la forme étendue d'une matrice d'interaction raffinée est un procédé itératif, lors duquel la présence de chaque opérateur dans la forme synthétique de la matrice d'interaction raffinée est interprétée, afin de déterminer de manière exhaustive toutes les interactions entre une famille d'agents source et une famille d'agents cible concernées par l'attribution de priorité. Ce procédé consiste dans un premier temps à remplir la forme étendue de la matrice d'interaction raffinée d'éléments de valeurs par défaut. Cette première étape consiste à :

- ajouter un élément $(I, p =?)$ dans la forme étendue de la matrice d'interaction raffinée pour chaque élément d'assignation (I) apparaissant dans la forme étendue de la matrice d'interaction brute ;
- ajouter dans la forme étendue de la matrice d'interaction raffinée un élément $(I, d = \delta, p =?)$ pour chaque élément d'assignation (I) apparaissant dans la forme étendue de la matrice d'interaction brute.

La seconde étape de ce procédé consiste à traiter séquentiellement chaque opérateur d'attribution de priorité en commençant la ligne des familles d'agents ne spécialisant aucune autre famille d'agents et en suivant progressivement leurs relations de spécialisation. Le traitement de l'attribution d'une priorité ρ à un élément d'assignation $(I, d = \delta)$ est alors formé des trois étapes qui suivent :

1. Soit L la ligne et C la colonne où se situe l'opérateur d'attribution de priorité dans la forme synthétique de la matrice d'interaction. La première étape est de remplacer l'élément $(I, d = \delta, p =?)$ situé à l'intersection de L et C dans la forme étendue par $(I, d = \delta, p = \rho)$.

2. Si la colonne C est associée à une famille d'agents ayant une ou plusieurs filles, alors le remplacement est aussi effectué à l'intersection de la ligne L et de la colonne de toutes ces familles d'agents filles.

3. Si la ligne L est associée à une famille d'agents ayant une ou plusieurs filles, alors chaque élément remplacé lors des étapes 1 et 2 dans la ligne L sont aussi remplacés dans la ligne associée à chacune de ces familles d'agents filles. Cette étape est effectuée récursivement pour toutes les familles d'agents filles rencontrées.

Ce procédé permet d'automatiser la construction de la forme étendue de la matrice d'interaction raffinée à partir de sa forme synthétique.

Tout comme la matrice d'interaction brute, la matrice d'interaction raffinée repose sur une forme synthétique et une forme étendue. Par conséquent, les principes suivis pour l'implémenter sont similaires à ceux énoncés dans le paragraphe intitulé « Implémentation d'une ligne de la matrice d'interaction brute. » à la page 241.

8.3.3 Limites de la spécialisation dans un modèle réactif de sélection d'interaction

L'attribution de priorités dans une matrice d'interaction raffinée sous forme synthétique repose sur le principe de la propagation de l'effet de l'opérateur d'attribution des priorités.

Dans le cas général, cette propagation permet de construire une matrice d'interaction raffinée en utilisant un nombre d'opérateurs bien moins important que le nombre d'éléments d'assignation auxquels il faut attribuer une priorité. Toutefois, dans certains cas, réutiliser les priorités définies dans des familles d'agents mères peut poser problème, voire s'avérer inutile.

Dans le cas d'une spécialisation de familles d'agents prenant la forme d'un losange, un même élément d'assignation peut se voir attribuer des priorités différentes dans deux familles d'agents mères. On se retrouve alors dans un cas similaire à celui décrit dans la figure 8.20 page 8.20, qui se résout de la même façon : par un choix manuel effectué par le concepteur.

Un second problème survient aussi dans le cas où une famille d'agents spécialise plusieurs familles d'agents mères. Ce problème est lié à différence sémantique existant dans l'utilisation de la spécialisation dans la matrice d'interaction brute et dans la matrice d'interaction raffinée. La matrice d'interaction brute définit les interactions que les agents sont capables d'initier ou de subir. Dans la forme étendue de cette matrice, le comportement par défaut de la spécialisation consiste à construire chaque cellule de la ligne associée à la famille d'agents fille en lui ajoutant les éléments d'assignation présents dans la ligne associée à ses familles d'agents mères. Ce procédé permet d'exprimer que si une famille d'agents mère est capable d'initier une interaction, alors ses familles d'agents filles en sont aussi capables par défaut. Il peut donc être automatisé.

La matrice d'interaction raffinée utilise les priorités afin d'établir une relation d'ordre entre des éléments d'assignation *situés sur une même ligne*. Par conséquent, lorsqu'une famille d'agents spécialise deux familles d'agents mères pouvant initier au moins une interaction, elle doit mettre en commun les relations d'ordres définies pour la ligne de chacune de ses familles d'agents mères. Dans un tel cas, rien ne garantit que les priorités utilisées dans les familles d'agents mères peuvent être réutilisées directement dans la familles d'agents fille afin d'établir une relation d'ordre valide parmi ses éléments d'assignation. Illustrons ce point sur un exemple.

Exemple.
Considérons la matrice d'interaction raffinée définie dans les figures 8.23(a) et 8.23(b). Dans cette matrice, un `Loup` *est à la fois un agent* `Mobile` *et un être* `Vivant`*. Un être* `Vivant` *est caractérisé par le fait qu'il peut* Mourir*. Puisqu'il s'agit de la seule interaction apparaissant dans cette ligne, nous faisons le choix de lui attribuer la priorité* 0*. Un agent* `Mobile` *est quant à lui caractérisé par le fait qu'il peut* Se Déplacer *dans l'environnement et, si ça n'est pas possible, d'*Attendre *dans l'environnement qu'il puisse à nouveau se déplacer. La relation d'ordre entre ces deux interactions est alors* Se Déplacer > Attendre*, ce que nous traduisons par deux priorités* 1 *et* 0*. Ces descriptions sont résumées par la forme synthétique de la matrice d'interaction raffinée présente dans la figure 8.23(a).*

Une telle spécification aboutit à la forme étendue résumée sur la figure 8.23(b), dans laquelle on peut remarquer que l'interaction Mourir *a une priorité moins élevée que l'interaction* Se Déplacer*, ce qui a pour conséquence qu'un* `Loup` *est capable de* Se Déplacer *alors qu'il était sensé* Mourir*.*

Source \ Cible	∅
Vivant	!(Mourir, p = 0)
Mobile	!(Se Déplacer, p = 1) !(Attendre, p = 0)
Loup :: Mobile Vivant	

(a) Forme synthétique de la matrice d'interaction raffinée

Source \ Cible	∅
Vivant	(Mourir, p = 0)
Mobile	(Se Déplacer, p = 1) !(Attendre, p = 0)
Loup :: Mobile Vivant	(Mourir, p = 0) (Se Déplacer, p = 1) !(Attendre, p = 0)

(b) Forme étendue de la matrice d'interaction raffinée

Figure 8.23 – Illustration du problème lié à la réutilisation de priorités définies dans des familles d'agents mères différentes.

La solution du problème mentionné dans l'exemple précédent est évidente : il faut donner une priorité plus élevée à l'interaction MOURIR. Cette solution prend en compte dans la famille d'agents `Vivant` les priorités attribuées dans la ligne associée à la famille d'agents `Mobile`, alors que ces familles ne sont pas corrélées. En effet, une `Plante` est un `Vivant` mais pas un `Mobile`, un `Robot` est un `Mobile` mais pas un `Vivant`. Attribuer des priorités dans la ligne associée à `Mobile` et dans la ligne associée à `Vivant` satisfaisant tous les situations est impossible. Par conséquent, la seule solution à ce problème consiste à réattribuer la priorité de tous les éléments d'assignation présents dans la ligne associée à la famille d'agents `Loup`. Dans ce cas, à défaut de pouvoir être utilisées directement, les priorités fournies dans la ligne de chacune des familles d'agents mères de `Loup` peuvent servir d'indicateurs permettant de construire une relation d'ordre, qui sera par la suite convertie en un ensemble de priorités.

La réutilisation de comportements dans le modèle réactif de sélection d'interaction est donc moins facile à caractériser que la réutilisation dans une matrice d'interaction brute.

8.4 Synthèse du chapitre

Les outils de conception de simulation informatique doivent à la fois faciliter la description d'une explication à un phénomène et garantir la robustesse de son implémentation face aux révisions de modèle. Ils doivent donc concilier ingénierie logicielle et ingénierie des connaissances. Dans ce chapitre, nous montrons que IODA facilite l'obtention d'un tel compromis.

Modèle exhaustif et spécification synthétique

La construction de simulations large échelle nécessite un compromis entre deux problèmes semblant s'opposer :

- le modèle doit faire exhaustivement état de toutes les interactions ayant lieu entre les agents de la simulation afin d'interpréter plus facilement le modèle conçu ;
- la construction d'un modèle se révèle être fastidieuse, peu robuste aux modifications et parfois impossible s'il faut décrire manuellement tous les cas d'interactions entre agents.

Ce problème n'était jusqu'à présent pas traité pour deux raisons. D'une part, la plupart des approches existantes ne représentent pas toute action impliquant simultanément plusieurs agents en tant qu'interaction. Elles ne peuvent donc pas bénéficier d'un modèle offrant une vue d'ensemble exhaustive sur toutes les interactions ayant lieu entre les agents. D'autre part, même si certaines approches fournissent une représentation des connaissances s'en approchant, elles se limitent à la description d'un modèle formel. Elles ne prennent pas en compte les problèmes liés à la méthodologie permettant de spécifier de tels modèles.

IODA repose sur des concepts facilitant la résolution d'un tel dilemme. Pour étayer ce point, nous décrivons dans ce chapitre une extension de l'approche IODA reposant sur le concept de **spécialisation** similaire à la relation « sorte de » rencontrée dans les langages orientés objets.

La spécialisation exprime une relation sémantique entre deux familles d'agents ayant deux conséquences sur la matrice d'interaction brute :

- une famille d'agents peut être la cible de toutes les interactions pouvant être subies par une famille d'agents qu'elle spécialise. Par exemple, si `Mouton` spécialise `Herbivore` et qu'un `Loup` peut MANGER un `Herbivore`, alors un `Loup` peut aussi MANGER un `Mouton` ;
- une famille d'agents peut être la source de toutes les interactions pouvant être initiée par une famille d'agents qu'elle spécialise. Par exemple, si `Mouton` spécialise `Vivant` et qu'un `Vivant` peut MOURIR, alors `Mouton` peut aussi MOURIR.

La matrice d'interaction brute est donc décrite à l'aide d'un nombre beaucoup plus réduit d'interactions et permet donc la spécification synthétique recherchée.

Pour concilier spécification non-exhaustive et vue exhaustive sur les interactions ayant lieu dans la simulation, la matrice d'interaction brute est représentée sous deux formes.

- La **forme synthétique** est une version de la matrice ne faisant apparaître qu'un nombre restreint d'interactions. Elle est utilisée à des fins de spécification.

– La **forme étendue** est un version de la matrice faisant apparaître de manière exhaustive toutes les interactions pouvant avoir lieu entre les agents de la simulation. Elle est utilisée non seulement à des fins d'interprétation, mais aussi à des fins d'implémentation. En effet, elle peut être directement utilisée dans les algorithmes décrits dans le cœur de l'approche IODA et ainsi aboutir à une implémentation.

Des algorithmes permettent de déduire automatiquement une forme étendue à partir d'une forme synthétique et permettent donc d'atteindre le compromis recherché.

La spécialisation caractérisée ici correspond à la transposition de l'héritage à la conception de simulations large échelle. Elle permet de concevoir des familles d'agents comme des extensions d'autres familles d'agents.

Dépasser les limites de l'héritage

Certains principes d'ingénierie logicielle de l'héritage constituent une limite du point de vue de l'ingénierie des connaissances dans les simulations. L'approche usuelle en ingénierie logicielle est de concevoir les objets par ajouts et surcharge de méthodes, ce qui se transpose dans les simulations par l'ajout ou la surcharge d'interaction. Un problème se pose alors s'il y a ajout d'une exception à une famille d'agents existante, par exemple l'ajout de `Clients aveugles` modélisés comme des `Clients` particuliers ne pouvant pas LIRE. En effet, cet ajout ne peut être fait qu'en créant une famille d'agents intermédiaire factorisant leurs interactions communes. L'ajout d'une exception implique donc la modification de familles d'agents existantes.

Afin de remédier à ce problème, nous nous détachons de ce point de vue de l'héritage en permettant d'exprimer les exceptions. Dans ce but, la forme synthétique de la matrice d'interaction brute est construite à l'aide de trois **opérateurs de spécialisation** :

– l'opérateur d'**ajout** qui permet d'ajouter à un agent la capacité d'initier une interaction avec un autre agent pour cible. Cet opérateur était utilisé implicitement jusqu'à présent ;
– l'opérateur de **modification** qui permet de modifier la garde de distance d'une interaction que l'agent peut initier ;
– l'opérateur de **retrait** qui permet de retirer à un agent la capacité d'initier une interaction avec un agent particulier pour cible.

Ces opérateurs permettent à la fois de reproduire les schémas de conception de l'héritage et d'exprimer les exceptions aux familles d'agents existantes. En effet, l'ajout d'une interaction s'exprime avec un opérateur d'ajout, la surcharge d'une interaction est exprimée soit en redéfinissant une primitive abstraite, soit en utilisant un opérateur de modification et les exceptions sont exprimées avec des opérateurs de retrait.

Limites de la spécialisation

Afin de faciliter encore plus la conception de telles simulations, la conception du modèle de sélection d'interaction pourrait être facilitée en usant de concepts similaires.

La conception de modèles de comportements réactifs basés sur des priorités peut effectivement être simplifiée en définissant une forme synthétique et étendue de la matrice d'interaction raffinée et un opérateur d'attribution de priorités. Nous prouvons ce point par la description d'algorithmes automatisant l'attribution de priorités dans une famille d'agents spécialisant d'une autre famille d'agents.

Toutefois, le cas de l'héritage multiple nécessite une attention particulière. En effet, une priorité n n'a pas systématiquement le même sens dans deux familles d'agents indépendantes. Par conséquent, les priorités fournies automatiquement à une famille d'agents héritant de plusieurs familles d'agents différentes n'est pas satisfaisante dans tous les cas. Une vérification sémantique de ces priorités est nécessaire de la part du concepteur. Cette limite est inhérente à la spécification de comportements réactifs en utilisant une adaptation de l'héritage.

En conclusion, IODA fournit un compromis entre ingénierie logicielle et ingénierie des connaissances ne pouvant être obtenu dans d'autres approches actuelles. En effet, elle offre une vue d'ensemble sur les interactions d'une simulation sans pour autant avoir à en faire une description exhaustive. Il y est de plus possible de décrire autant des extensions que des exceptions au comportement d'une famille d'agents. La