Conception de Simulations Multi-Agents

Plan du chapitre :

Du fait de l'hétérogénéité des domaines d'application de la simulation, différentes approches aux spécificités très différentes sont utilisées pour implémenter le modèle d'une simulation multi-agents. Nous discutons dans ce chapitre des avantages et inconvénients de ces différentes approches, en les regroupant par philosophies de conception. Cette étude est scindée en trois sections.

Dans la section 2.1, l'analyse porte sur les moyens employés afin d'implémenter la simulation. Nous dégageons de cette étude l'importance d'utiliser une approche transversale de conception, accompagnant les concepteurs dès les premières étapes de la simulation jusqu'à l'obtention d'un simulateur concret. La possibilité de construire une telle approche dépend fortement de la structure du modèle utilisé.

Dans les deux sections qui suivent, les apports et les lacunes de différents modèles sont étudiés relativement au processus transversal de conception de simulations. La section 2.2 étudie cette question selon la perspective de l'architecture interne des agents et la façon dont leur comportement est construit. Enfin, la section 2.3 étudie cette question selon la perspective des interactions ayant lieu entre les agents et la façon dont le lien entre interactions et comportement des agents est établi.

Avant de nous attaquer l'état de l'art, nous commençons ce chapitre par une brève discussion concernant la distinction entre modèle conceptuel, modèle exécutable et simulateur.

Distinction pratique entre modèle conceptuel/modèle exécutable La distinction entre modèle conceptuel, modèle exécutable et simulateur marque la transition progressive entre descriptions de haut niveau, abstraites et peu précises (modèle non-formel) à des descriptions de bas niveau concrètes et pouvant être implémentées (modèle exécutable). La distinction modèle/implémentation que l'on trouve par exemple avec Aalaadin/Madkit [FG98] ou DEVS [ZKP00]/JAMES [SU01] est syntaxique. Elle est donc simple à identifier. La distinction entre modèle conceptuel et modèle exécutable n'est pas aussi simple.

En théorie l'unique différence entre modèle conceptuel et modèle exécutable tient à la présence (ou non) d'informations propres à l'implémentation. Il n'existe à ce jour aucune ontologie permettant de décrire précisément un phénomène et symétriquement aucune ontologie ne permet de définir clairement ce qui est propre à l'implémentation. Par conséquent, nous ne faisons pas de différence entre modèle conceptuel et modèle exécutable dans ce chapitre. Nous les désignons tous deux indistinctement par le terme « modèle ».

2.1 Pratique du processus de conception de simulations

Le processus de conception de simulations peut être pratiqué selon des approches très différentes nécessitant des degrés d'expertise en informatique variables. Nous en distinguons trois grandes familles :

- les approches ouvertes qui reposent sur des langages permettant de spécifier librement presque n'importe quel type de simulations. Cette famille d'approche comprend les langages de programmation, les plateformes multi-agents ouvertes, les plateformes multi-agents dédiées aux experts du domaine et les plateformes de simulation multi-agents ouvertes ;
- les approches dirigées par la structure du modèle qui aident la spécification des simulations en fournissant une architecture spécifique et précise au modèle et à l'implémentation ;
- les approches transversales qui décrivent non seulement un modèle formel et une architecture d'implémentation, mais fournissent aussi une méthodologie de conception. Cette dernière décrit explicitement comment construire le modèle et comment aboutir à son implémentation.

2.1.1 Langages de programmation

Le choix de la plateforme à utiliser pour implémenter une simulation est loin d'être évidente pour plusieurs raisons. La première raison est qu'il existe un très grand nombre de plateformes multi-agents : Nikolai [NM09] en compte plus de 50 et sa liste n'est pas exhaustive. Certains travaux tels que Masev [GGB08] établissent des critères de comparaison inter-plateformes pour aider ce choix. Il reste toutefois nécessaire de comparer les plateformes deux à deux pour trouver la plus appropriée à la simulation réalisée. Ces critères sont donc peu utilisés. La seconde raison est liée à l'opacité des plateformes et leur manque de documentation. En effet, en l'absence de connaissances précises sur le fonctionnement interne de la plateforme et des choix d'implémentation lui tant sous-jacents, des résultats erronés peuvent être obtenus avec un modèle pourtant correct.

Afin d'éviter ces problèmes, une première approche de l'implémentation d'un simulation consiste à implémenter intégralement le modèle dans un langage de programmation tel que C, C++, Java ou Fortran [Axe97, Sha98]. Les choix d'implémentation du simulateur sont pleinement contrôlés, évitant ainsi les erreurs liées à l'opacité ou le manque de documentation des plateformes existantes. Toutefois, ce gain est obtenu au détriment de deux problèmes majeurs du processus de simulation. D'une part, la réduction des efforts d'implémentation lors des révisions du modèle dépend uniquement des compétences du programmeur. D'autre part, la validité des choix d'implémentation repose uniquement sur l'expérience du programmeur concernant la simulation et le langage utilisés. De nos jours, ce type d'approche n'est donc que très peu utilisé [Sha98].

2.1.2 Plateformes multi-agents ouvertes

Comme nous l'avons mentionné dans le chapitre 1, le paradigme multi-agents est particulièrement adapté pour implémenter les simulations. La spécification d'une simulation pourrait donc s'appuyer sur des plateformes ouvertes dédiées à la conception de systèmes multi-agents telles que Cougaar [HTW04], JADE [BPR99], Madkit [FG98] ou MAGIQUE [BM97]. En effet, ces plateformes implémentent de manière concrète le concept d'agent et d'ordonnanceur de l'activité des agents. De plus elles s'appuient pour la plupart sur un modèle formel qui peut être utilisé pour représenter le modèle de la simulation. Par exemple, JADE repose sur les spécifications de la FIPA [FIP10b] et Madkit [FG98] sur le modèle AGR.

Exemple : la plateforme Madkit

Madkit est une plateforme écrite en Java permettant d'implémenter des systèmes multi-agents reposant sur le modèle Agent/Groupe/Role [FG98]. Cette approche vise principalement à concevoir des applications hétérogènes et réparties. Elle se focalise donc sur des problématiques d'interopérabilité entre agents hétérogènes. La spécification et l'implémentation est centrée sur l'organisation du système multi-agents et se base sur trois concepts fondamentaux : les agents, les groupes et les rôles.

Un *agent* est une entité autonome et communicante pouvant jouer des *rôles* dans des *groupes*. Un groupe est un regroupement d'agents représentant un sous-système du système multi-agents. Un agent

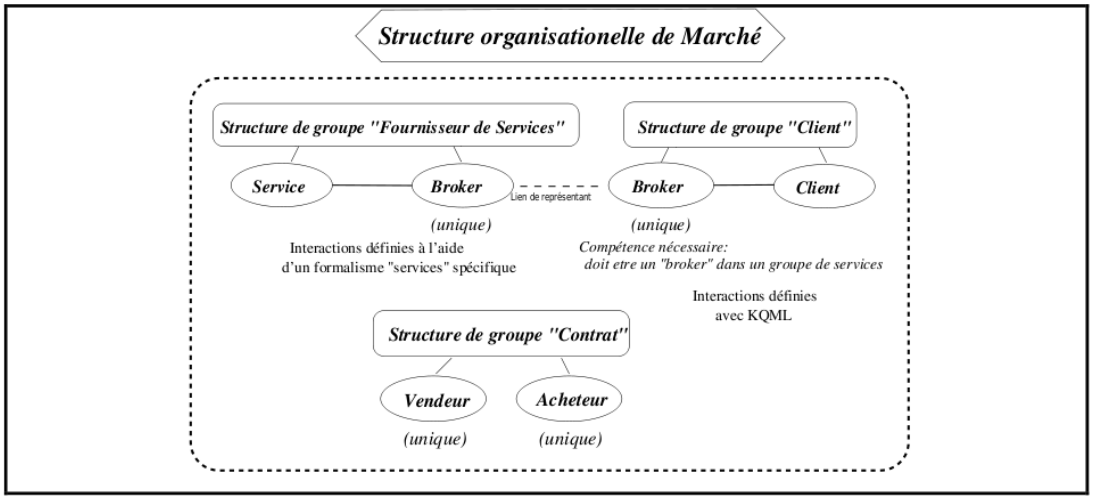

FIGURE 2.1 – Illustration des concepts de rôle, groupe, structure de groupe et structure organisationnelle du modèle AGR décrivant un marché. Dans cet exemple, un agent du rôle `Broker` sert de pivot entre agents du rôle `Client` et agents du rôle `Service` afin d'appareiller de manière dynamique un `Client` et un `Service` en tant que `Vendeur` et `Acheteur`.

peut appartenir à plusieurs groupes. Un rôle est une représentation abstraite d'une fonction, d'un service ou sert à identifier un agent au sein de son groupe. Chaque groupe spécifie l'ensemble des rôles qu'il peut contenir et chaque agent présent dans un groupe spécifie le ou les rôles qu'il peut y jouer. Pour compléter ces notions et permettre l'implémentation d'un système multi-agents, s'ajoutent au modèle les notions de *structure de groupe* et de *structure organisationnelle*. Une structure de groupe décrit les interactions [8] pouvant avoir lieu entre agents dans un groupe en fonction des rôle qu'ils peuvent y jouer. Elle est représentée sous la forme d'un graphe dirigé où les nœuds sont les identifiants des différents rôles du groupe et où les arcs connectent les rôles pouvant interagir. La structure organisationnelle décrit de manière macroscopique le système multi-agents. Elle est constituée de l'ensemble des structures de groupe composant le MAS et identifie les agents dits représentants qui servent d'interface entre des groupes différents.

L'architecture comportementale des agents ne faisant pas partie des problématiques fondamentales de cette approche, sa spécification est délaissée et doit être intégralement réalisée lors de l'implémentation. Il en va de même pour la notion d'environnement, limitée dans ce cas à un environnement social (le groupe). Malgré ces difficultés, il est toutefois possible d'implémenter des simulations sur ces plateformes. Par exemple, la plateforme Madkit a été utilisée pour implémenter la librairie Turtlekit [Mic00]. Cette dernière permet d'implémenter des simulations ayant lieu dans un environnement en deux dimensions.

Une extension du modèle AGR appelée AGRE [JFB05] réduit partiellement les problèmes liés à l'implémentation de simulations en intégrant à son modèle le concept d'environnement. Une généralisation récente du modèle AGR nommée MASQ [SFT09] fait de même en ajoutant de plus des concepts sociaux tels que les institutions ou les normes.

Discussion

Les plateformes ouvertes ont pour motivation première la spécification de systèmes multi-agents hétérogènes. Elles font donc sciemment le choix de ne pas imposer d'architecture interne spécifique aux agents, afin de garantir la plus grande hétérogénéité possible. Bien qu'il soit possible d'implémenter directement une simulation multi-agents sur ces plateformes, elles sont en général limitées au développement de librairies de simulation.

8. « interaction » peut désigner des notions différentes (voir section 2.3). Dans le cas présent, ils s'agit de protocoles d'interaction, *i.e.* de protocoles décrivant des échanges de messages réalisés par des agents pour atteindre un objectif.

Dans ces approches l'implémentation d'un élément fondamental de la simulation, le comportement des agents, nécessite la maîtrise de langages de programmation tels que JAVA ou C. Pour que le comportement des agents soit le plus simple à spécifier, d'autres plateformes permettent d'implémenter des simulations à l'aide de langages simples et accessibles à des non-informaticiens.

Ce problème a donné naissance à toute une gamme de plateformes permettant d'implémenter des simulations à l'aide de langages simples à manipuler.

2.1.3 Plateformes multi-agents dédiées aux experts du domaine

Plutôt que d'utiliser des langages complexes tels que JAVA ou C, certaines plateformes reposent sur des langages de programmation plus intuitifs à utiliser. La conception du comportement des agents reste libre et non guidée, mais peut être faite par des experts du domaine. Ces plateformes reposent sur des langages de programmation expressifs tels que Netlogo [WC99] ou sur des langages de programmation graphique que l'on retrouve dans Repast Simphony [NTCO07] ou dans SeSam [KHF06].

Exemple : la plateforme Netlogo

Netlogo [WC99] est une plateforme multi-agents basée sur le langage de programmation Logo. Elle permet de spécifier des simulations dans lesquelles des agents évoluent dans un espace en deux dimensions. Une simulation y consiste à contrôler le comportement d'un ensemble de tortues [9] similaires à des agents. Le comportement de chaque tortue y est décrit à l'aide de commandes simples et intuitives telles que `forward 5` pour faire avancer la tortue de 5 unités ou `right 90` pour faire tourner la tortue de 90 degrés sur sa droite. Il est aussi possible d'utiliser des commandes, des procédures et des fonctions plus évoluées permettant de spécifier des comportements complexes. Ces commandes incluent la manipulation d'ensembles d'agents, la perception dans l'environnement, l'ordonnancement de l'activité des agents ou la définition de races [10] de tortues (*i.e.* des « types » de tortues). Cette plateforme fournit de plus des outils graphiques très simples permettant de paramétrer, exécuter et analyser la simulation sans passer par une phase de compilation.

Discussion

Les plateformes s'appuyant sur des langages de spécification simples favorisent l'implication de personnes n'ayant pas des compétences poussées en informatique lors de l'implémentation. Elles réduisent donc les erreurs d'interprétation du modèle lors de l'implémentation. Les simulations ou les agents sont toutefois définis en un seul bloc de code peu réutilisable ne facilitant pas les révisions du modèle. Ces approches sont donc adéquates pour construire des prototypes de simulation ou construire des simulations contenant un nombre restreint d'agents au comportement peu varié. Elles ne sont par contre pas adaptées pour construire des simulations à plus grande échelle [Axe97], *i.e.* contenant une grande variété d'agents et de comportements.

2.1.4 Plateformes de simulation multi-agents ouvertes

Pour faciliter la conception du comportement des agents, une approche basée sur les librairies de simulation peut être envisagée. Cette approche se retrouve au sein de plateformes de simulation multi-agents ouvertes telles qu'Ascape [Par01] et Swarm [MBLA96] ou Opensteer [Rey02].

Exemple : Swarm

Swarm est une librairie de programmation écrite dans le langage orienté-objet Objective-C, permettant d'implémenter des simulations multi-agents. Elle est générique et ouverte et repose sur les concepts fondamentaux d'*agent*, de *swarm* et d'*ordonnanceur*.

Un swarm est objet (au sens de la programmation orientée-objet) composé d'une collection d'agents et d'un ordonnanceur régulant l'activité de ces agents. Il constitue donc un système multi-agents. Chaque

9. « turtles » dans le texte
10. « breeds » dans le texte

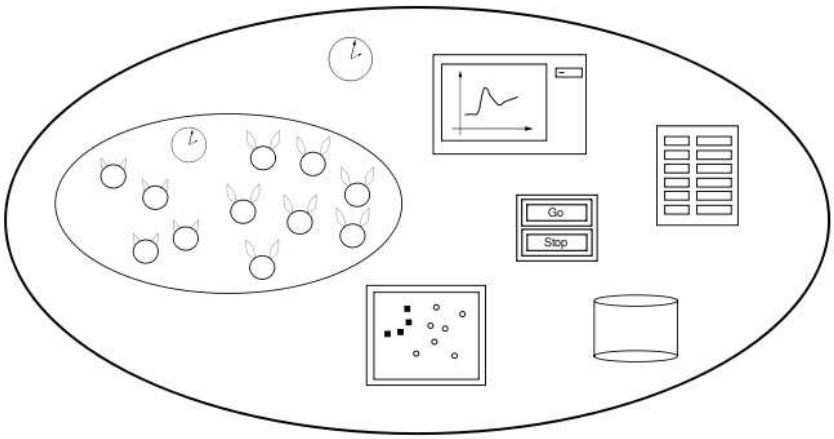

FIGURE 2.2 – Schéma représentant une simulation complète implémentée avec la librairie Swarm. Les ellipses représentent des swarms, l'horloge représente l'ordonnanceur associé au swarm. Les autres entités sont des agents.

agent est un objet (au sens de la programmation orientée-objet) qui peut aussi être un swarm. Cette structuration permet ainsi de construire une hiérarchie d'agents. Cette hiérarchie est particulièrement utile pour spécifier des phénomènes ayant lieu à des échelles différentes. Par exemple, modéliser le comportement de cellules dans un organisme, lui-même le fruit du comportement de protéines dans les cellules. La spécification du comportement des agents est libre et laissée à la discrétion du programmeur.

Cette librairie unifie la spécification du simulateur utilisé : les interfaces graphiques et de contrôle de la simulation sont aussi représentés par des agents dans un swarm (voir figure 2.2).

Outre son modèle hiérarchique, l'intérêt de swarm réside dans sa communauté. Non seulement Swarm est une plateforme ouverte et générique, mais sa communauté est très active, comme en témoigne les différentes éditions de la conférence Swarmfest [11] . Au fil de son utilisation par la communauté, des ensembles de librairies spécifiques aux domaines d'application ont été développées, telles que les swarms représentant un environnement en deux dimensions, les agents basés sur des réseaux de neurones, *etc.* Ces librairies permettent de réutiliser différentes architectures d'agents et d'environnement, réduisant ainsi les efforts d'implémentation d'une simulation à une autre.

Discussion

Les plateformes de simulation multi-agents ouvertes fournissent un modèle de base permettant de structurer la simulation à l'aide des notion d'agent ou d'ordonnanceur. Elles laissent de plus une liberté totale d'implémentation moins contraignante que dans les plateformes dédiées aux systèmes multi-agents. En effet, les plateformes ouvertes comme Swarm sont utilisées pour développer des librairies réutilisables, réduisant ainsi les coûts liés à l'implémentation d'une simulation. Elles favorisent de plus la validité de l'implémentation en réutilisant des librairies déjà validées.

De telles plateformes permettent donc d'implémenter un large spectre de simulations, tout en fournissant une aide à la conception au travers de librairies. Toutefois, cette aide se limite à la réutilisation de différentes architectures pour la simulation. La conception du modèle et de son implémentation ne sont pas guidées, si bien que la validité d'un simulateur dépend toujours compétences en programmation de la personne effectuant l'implémentation. Il en va de même pour la réutilisation de l'implémentation lors des révisions du modèle.

11. voir l'url `http://www.swarm.org/wiki/Swarm:_SwarmFest`

2.1.5 Approches dirigées par l'architecture du modèle

Les approches mentionnées jusqu'à présent sont ouvertes et permettent d'implémenter librement la simulation. Elles sont donc applicables à un grand nombre de simulations. Néanmoins, cette liberté est acquise au prix d'une absence presque totale d'aide la conception du comportement des agents. Les programmes conçus sont de plus peu robustes aux révisions du modèle.

Une autre approche consiste à fournir une architecture et un modèle précis aux agents et à leur comportement. Cette précision est acquise au prix de la liberté d'écriture du comportement et peut donc sembler restrictive. Elle est néanmoins nécessaire pour spécifier des simulations contenant un nombre important d'agents ou contenant des agents au comportement complexe. Ce problème n'est d'ailleurs pas limité aux simulations, comme le mentionne Ferguson à propos de la conception d'applications :

> « In most professions, competent work requires the disciplined use of established practices. It is not a matter of creativity versus discipline, but one of bringing discipline to the work so creativity can happen. » [FHK+97], d'après [Rob06]

La description d'une architecture précise du modèle permet de grandement guider la phase d'implémentation et d'éviter des choix pouvant aboutir à des simulateurs non valides. La transition entre modèle et implémentation est ainsi facilitée. En dépit de ces avantages, ces approches ne facilitent pas totalement la conception de simulations. En effet, les modèles ont une structure complexe et ne peuvent donc être conçus sans une aide appropriée. Ce problème est d'autant plus fondamental dans le cas de simulations contenant un grand nombre d'agents différents interagissant de manière variées.

Afin de faciliter la conception de simulations, le modèle doit pouvoir être conçu graduellement. Cette aide à la conception est obtenue dans les approches que nous qualifions de « transversales ».

2.1.6 Approches transversales

La conception du modèle nécessite plus que sa simple architecture. En effet, un modèle précis repose sur un grand nombre d'informations qu'il n'est possible de spécifier que graduellement. Il faut en particulier savoir par où commencer la spécification d'un modèle, quel cheminement suivre pour parvenir à un modèle complet et comment implémenter le modèle obtenu en un simulateur. Nous appelons *approche transversale de conception de simulations* les approches fournissant les outils répondant à ce problème.

Définition 1. ***Approche transversale de conception de simulations***
Nous appelons **approche transversale de conception de simulations** les approches supportant la conception d'une simulation du début du processus de conception de simulation (la description du modèle) à son implémentation sur une plateforme de simulation donnée.
Ces approches sont caractérisées par quatre éléments :
- un modèle formel ;
- une plateforme de simulation ;
- une méthodologie permettant de construire graduellement un modèle en passant de descriptions abstraites de haut niveau à des description fines et précises ;
- des moyens automatisant l'implémentation d'un modèle.

Bien que fondamental, ce problème est traité dans peu d'approches. Ses solutions se retrouvent principalement sous deux formes qui visent à simplifier la spécification du modèle tout en y introduisant un maximum d'informations nécessaires à l'implémentation.

Conception dirigée par des vues sur un unique modèle

Le premier type d'approches transversales repose un unique modèle qu'il est possible de remplir graduellement en usant de différentes vues. Chaque vue permet des spécifications de plus en plus précises pouvant être manipulées par des experts du domaine. L'implémentation est faite à l'aide d'un générateur de code qui se charge de la traduction des modèles graphiques en une implémentation exécutable ou en un ensemble de squelettes prêts à être remplis. Ce type de solution est appliqué par exemple dans le projet Manta [Dro93] pour des simulations en éthologie.

Exemple : Le projet Manta

Le projet Manta [DF92] a pour objectif de simuler le comportement d'insectes sociaux à l'aide du modèle formel d'Ethomodélisation [Dro93]. Cette approche postule qu'un agent dispose de primitives de comportements qui décrivent ses actes moteurs comme le déplacement, le suivi de phéromones, le dépôt d'un objet transporté, *etc.* Le comportement d'un agent est décrit sous la forme de séquences de primitives de comportement, appelées tâches. Ces tâches se déclenchent de manière exclusive en fonction de la force de stimuli internes ou externes selon un modèle d'activation de Lorenz [Lor84] : une tâche ne se déclenche qu'en présence du stimuli qui lui est associé et seulement si la force du stimuli pondérée par le poids de la tâche dépasse un seuil inhibiteur. La modification du poids et du seuil inhibiteur des tâches d'un agent lui permettent de s'adapter en fonction de ses expériences antérieures.

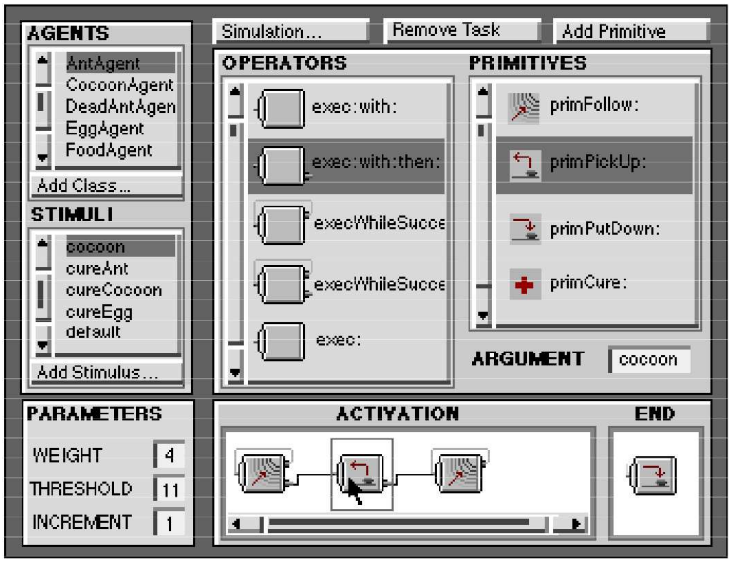

FIGURE 2.3 – Le Taskmanager du projet Manta qui permet de concevoir graphiquement des modèles d'Ethomodélisation [Dro93] et d'en générer le code. Cette capture d'écran illustre comment créer et éditer les tâches de chaque agent.

Dans Manta la conception du modèle et son implémentation se font par un outil appelé *TaskBrowser*. Cet outil permet de spécifier de manière graphique et intuitive l'ensemble des tâches de chaque agent en se reposant sur des bibliothèque de primitives de comportements pré-établies (voir figure 2.3). Il permet alors de générer le code correspondant prêt à être compilé et utilisé dans le simulateur.

Ingénierie dirigée par les modèles

Le second type d'approche transversale consiste à utiliser des méthodologies pouvant aller jusqu'à l'ingénierie dirigée par les modèles (MDE) telles que ADELFE [BCGP05], Gaia [ZJW03], INGENIAS [PGS03], O-MAsE [GODR$^+$08], Prometheus [WP04] ou Tropos [BPG$^+$04]. Dans ces approches, la conception repose sur un ensemble de modèles représentant de manière de plus en plus précise les fonctionnalités de l'application. Chaque modèle possède une représentation graphique dédiée qu'il est possible d'éditer à l'aide d'environnements de développement intégré (IDE) tels que la plateforme TAOM4E [SE 09] pour Tropos, la plateforme PDT [PTW05] pour Prometheus, la plateforme agentTool III [GODR09] pour O-MAsE ou l'Ingenias Development Kit (IDK) [Gru10] pour INGENIAS. Certaines d'entre elles permettent de plus d'automatiser la transition d'un modèle à un autre à l'aide de transformations de modèles, *i.e.* un ensemble de règles exprimant chaque élément d'un modèle source en éléments d'un modèle cible. Ces transformations sont automatisées dans les IDE par des outils dédiés comme ATL [ATL09] ou Kermeta [Tri09].

Exemple : La méthodologie INGENIAS

INGENIAS [PGS03] est une méthodologie de conception de systèmes multi-agents reposant sur un principe similaire au modèle AGR présenté en section 2.1.2. Cette approche fournit toutefois une architecture interne aux agents composée :

- d'un état mental des agents contenant des faits et des buts ;
- d'une architecture comportementale des agents fondée sur des buts réalisés à l'aide de tâches produisant et consommant des faits ;
- d'une description explicite du lien entre les tâches effectuées par les agents et la communication d'informations entre agents.

Cette méthodologie focalise la conception d'un système multi-agents selon cinq modèles/vues différents.

Le modèle portant sur l'organisation structure le système multi-agents et de définit les comportements de manière macroscopique. Ce modèle décompose le MAS en groupes et définit les rôles existant dans chaque groupe, les types d'agents présents dans chaque groupe, les dépendances sociales entre agents de l'organisation (par exemple la subordination) ainsi que l'organisation du comportement des agents. Par exemple, le comportement de chauve-souris [SPGS06] est divisé en un comportement de jour et un comportement de nuit.

Le modèle portant sur les interactions[12] décrit comment se déroulent les échanges d'informations ou les requêtes entre agents jouant certains rôles et quel est leur lien avec les tâches et l'état mental des agents.

Le modèle portant sur les agents établit le lien entre un type d'agent d'une part et les rôles qu'il joue, les buts qu'il poursuit, les états mentaux qu'il calcule ou gère d'autre part.

Le modèle portant sur l'environnement définit ce que les agents peuvent percevoir.

Le modèle portant sur les buts/tâches établit le lien existant entre les différents buts et les tâches effectuées par les agents. Il décrit de plus les faits consommés ou produits par les tâches.

Chaque modèle dispose d'une représentation graphique qui lui est propre basée sur un formalisme similaire à UML. L'IDE associé à INGENIAS appelé IDK [Gru10] permet d'éditer librement chacun de ces modèles et de passer automatiquement à une implémentation. Bien qu'initialement conçue pour la construction de systèmes multi-agents, INGENIAS peut aussi être utilisée pour décrire des simulations, en particulier dans le domaine des sciences sociales [SPGS06]. Toutefois, la conception de modèles avec INGENIAS reste fortement ouverte : aucune directive n'est donnée ni sur l'ordre dans lequel construire les modèles, ni sur la façon de procéder pour les remplir. De plus, elle laisse une part de l'architecture du comportement des agents non contrainte pour favoriser la cohabitation d'agents hétérogènes : la gestion et le calcul des états mentaux des agents n'est que mentionnée et doit être implémentée indépendamment.

Une extension [GMGSFF09] récente de la théorie liée à INGENIAS permet de remédier partiellement à ces problèmes. Elle identifie un ordre empirique dans lequel les modèles sont en pratique spécifiés (voir figure 2.4) et s'appuie dessus pour améliorer l'aide à la conception. Chaque modèle de ce cheminement fournit des informations qui sont réutilisées afin de pré-remplir automatiquement les modèles suivants à l'aide de transformations de modèles. Cette extension permet donc de mieux guider la conception des différents modèles.

Discussion

Qu'elles soient basées sur des vues multiples ou sur des modèles multiples, les approches transversales fournissent des outils facilitant la conception de simulations. En effet, elles permettent de spécifier graduellement et de manière modulaire le modèle d'une simulation à l'aide de représentations de plus en plus

12. au sens protocoles d'interaction

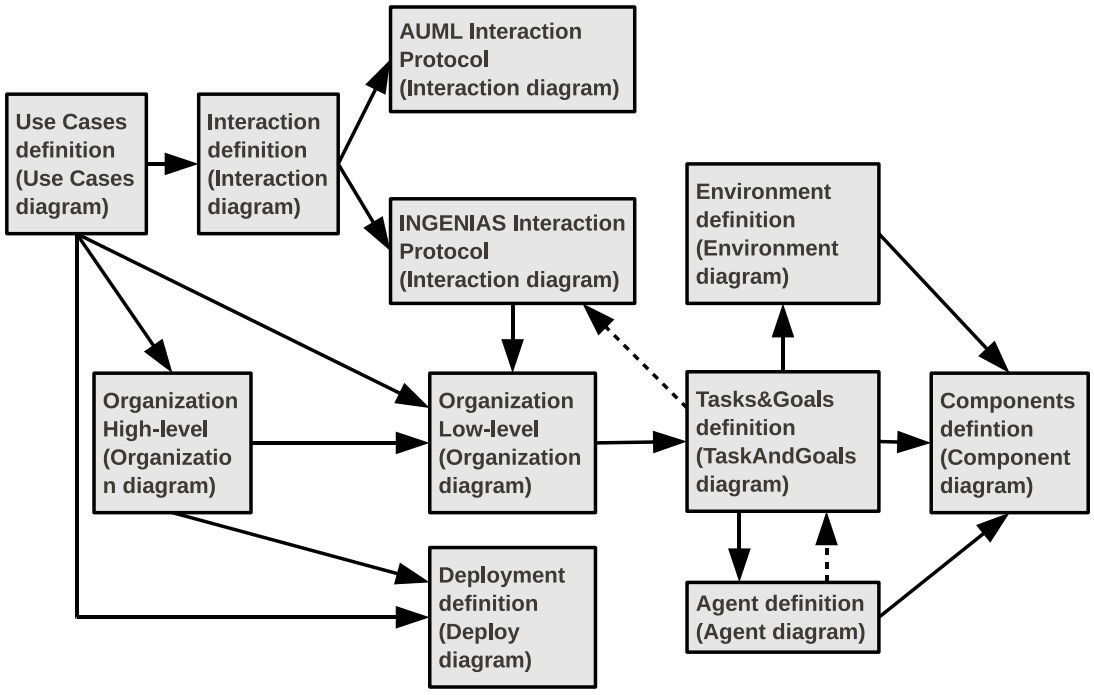

FIGURE 2.4 – Ordre de spécification des modèles guidant la conception dans l'extension d'INGENIAS présentée dans [GMGSFF09]. Les flèches pleines représentent l'ordre de conception des modèles et les flèches pointillées l'influence d'un modèle sur un autre.

précises ainsi que de passer à une implémentation par des procédés automatisés. Elles concilient donc moyens plus simples et intuitifs que la programmation pour spécifier une simulation et moyens permettant de parvenir à une implémentation. Elles fournissent donc le meilleur compromis aux différents problèmes inhérents à la simulation.

Toutefois, un problème subsiste dans ces approches : le processus de conception se focalise sur les buts que l'application doit atteindre et les fonctionnalités qu'elle doit exprimer. Elles utilisent pour cela des représentations telles que des diagrammes de cas d'utilisation (dans INGENIAS) ou de décomposition des buts devant être atteints par le logiciel (dans Tropos). Ce principe s'applique mal aux simulations puisque le but n'y est pas de fournir des fonctionnalités au sens logiciel, mais de reproduire les caractéristiques du phénomène. Pour les reproduire, la description du modèle doit d'abord se focaliser sur les éléments observables du phénomène pour seulement finir par l'introduction d'hypothèses de fonctionnement. Les éléments observables du phénomène ne s'expriment ni en termes de cas d'utilisation, ni en termes de buts : il s'agit d'actions entreprises par des entités et des interactions observées entre entités. Il est donc nécessaire d'utiliser des approches dédiées à la simulation se focalisant en priorité sur ces concepts.

Les approches transversales conçoivent une application multi-agents en suivant le même patron général résumé sur la figure 2.5. Elles commencent par décrire le système d'un point de vue macroscopique en identifiant les organisations (lorsqu'il y en a) et les relations observables entre agents (par exemple les protocoles d'interaction). Elles s'intéressent ensuite à ce que les agents sont capables de faire pour enfin finir par la description du comportement des agents. Ce cheminement permet de passer progressivement de spécifications macroscopiques observables dans le phénomène simulé à des spécifications microscopiques qui relèvent des hypothèses sur l'origine du phénomène. Il est particulièrement adapté à la conception de simulations, puisque la part des modèles les plus sujettes aux modifications sont spécifiées en aval du processus de conception.

Un tel cheminement n'est possible qu'avec des modèles ayant une structure adéquate qui va par ailleurs grandement conditionner les contributions aux problématiques inhérentes à la simulation. Dans les sections qui suivent, nous étudions différentes architectures et modèles de systèmes multi-agents et

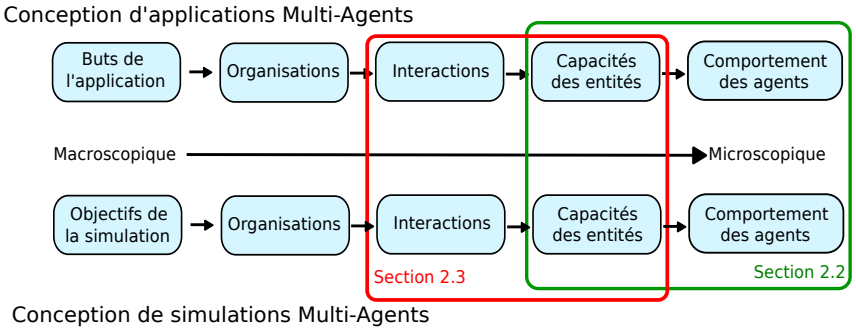

FIGURE 2.5 – Succession des principales étapes d'un processus transversal de spécification d'une simulation.

voyons en quoi elles favorisent ou non la description d'un tel cheminement. La prochaine section les étudie selon la perspective des « capacités des entités » et de leur lien avec le « comportement des agents ». Celle qui la suit se concentre davantage sur le lien entre les « Interactions » et les « Capacités des entités ». Dans cet état de l'art, nous ne prenons pas en compte la dimension « Organisations ».

2.2 Architectures multi-agents

On distingue usuellement les architectures internes par le degré de cognition exprimé par les agents. Ce dernier est caractérisé par une catégorie parmi les suivantes : *réactif*, *cognitif*, ou *hybride*. De telles architectures expriment comment un agent choisit les actions qu'il entreprend lorsque la parole lui est donnée par l'ordonnanceur de la simulation.

2.2.1 Architectures réactives

Dans les architectures réactives, le comportement des agents consiste à effectuer des actions en réaction à des stimuli perçu. Ces stimuli correspondent à une modification de leur environnement, à une sollicitation directe d'un autre agent ou à une modification de leur propre état interne.

Dans son expression la plus simple, le comportement d'un agent consiste à initier des actions en réaction à chaque stimulus qu'il perçoit. Le comportement d'un agent est alors décrit uniquement par un ensemble d'associations *stimulus/réaction*. En pratique, les modèles les plus utilisés raffinent ce principe : un agent peut y choisir les stimuli auxquels il répond en fonction de son contexte. L'agent réagit donc toujours à des stimuli, mais de manière plus intelligente permettant d'exprimer simplement des comportement parfois complexes. Ces architectures se retrouvent dans des approches telles que la subsomption [Bro86], la plateforme Maleva [BM07], la plateforme SeSam [KHF06] ou encore l'Éthomodélisation [Dro93].

Exemple : l'approche Maleva [BM07]

Maleva est une approche visant à faciliter la conception incrémentale d'agents réactifs en décrivant le comportement d'un agent comme un assemblage récursif de composants logiciels. Le comportement d'un agent y est un composant dont les bornes d'entrée correspondent à des perceptions (*e.g.* la quantité de phéromones là où l'agent se situe) et les bornes de sortie à des paramètres d'actions élémentaires (*e.g.* une distance à parcourir). Le composant peut être :

- *primitif* et calculer les valeurs des bornes de sortie à l'aide d'algorithmes manipulant les informations des bornes d'entrée ;
- ou *composite* et déterminer les valeurs produites en sortie à partir d'un assemblage interne de composants.

Le comportement est ainsi conçu de manière récursive et de plus en plus fine. La réutilisation du comportement lors des révisions du modèle est donc facilitée.

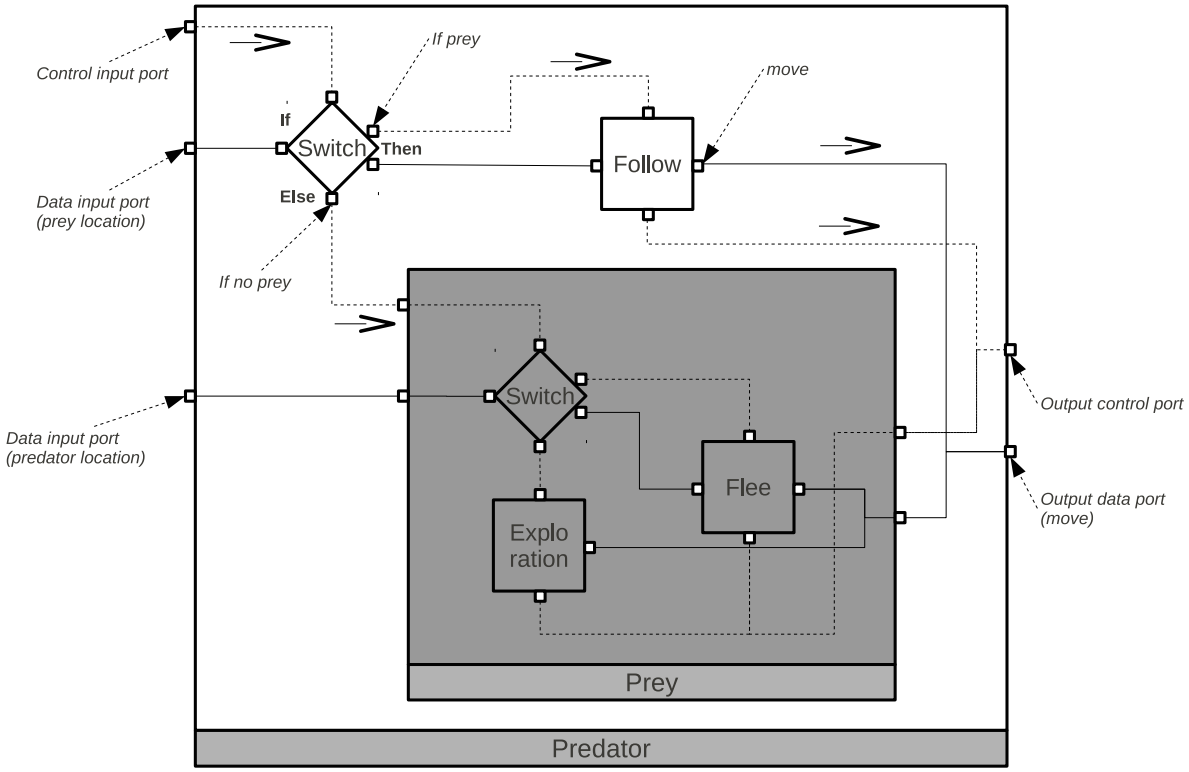

FIGURE 2.6 – Représentation du comportement d'un agent selon l'approche Maleva [BM07]. Cette représentation est ici illustrée dans le contexte de la simulation d'un écosystème où un prédateur est considéré comme une proie particulière pouvant poursuivre d'autres agents. Le comportement de proie est constitué de sous-composant décrivant dans quel cas la proie explore son environnement et dans quel cas ce comportement est subsumé par la fuite d'un prédateur. Le comportement d'un prédateur consiste à poursuivre une proie si c'est possible, sinon à se comporter comme une proie.

Chaque comportement dispose de plus de bornes de contrôle qui permettent d'inhiber le comportement d'autres composants d'une manière similaire à l'architecture de subsomption. L'inhibition se fait à l'aide de composants de contrôle qui reproduisent des structures de contrôle telles que les structures conditionnelles. La figure 2.6 illustre ces concepts pour un agent étant à la fois un prédateur et une proie.

Exemple : la plateforme SeSam

SeSam est une plateforme permettant de concevoir le comportement des agents sans connaître la syntaxe des langages de programmation « traditionnels ». Le comportement des agents y est spécifié à l'aide de graphes (voir figure 2.7). Dans ces graphes, les nœuds (appelés « activités ») sont des actions effectuées par l'agent et les arcs lient des activités effectuées en séquence. À chaque arc est associée une condition qui doit être évaluée à vrai pour que la transition vers la nouvelle activité puisse être effectuée. À chaque fois qu'un agent atteint un nouveau nœud dans son graphe, il effectue l'action qui y est associée puis cherche à effectuer une transition vers une nouvelle activité, si c'est possible.

Les actions effectuées dans une activité et les conditions de transition entre activités sont décrits sous une forme arborescente (voir figure 2.7). Chaque nœud de cet arbre correspond à une primitive d'un langage spécifique à SeSam, une opération arithmétique, une structure de contrôle ou un accesseur à l'état d'un agent ou de l'environnement.

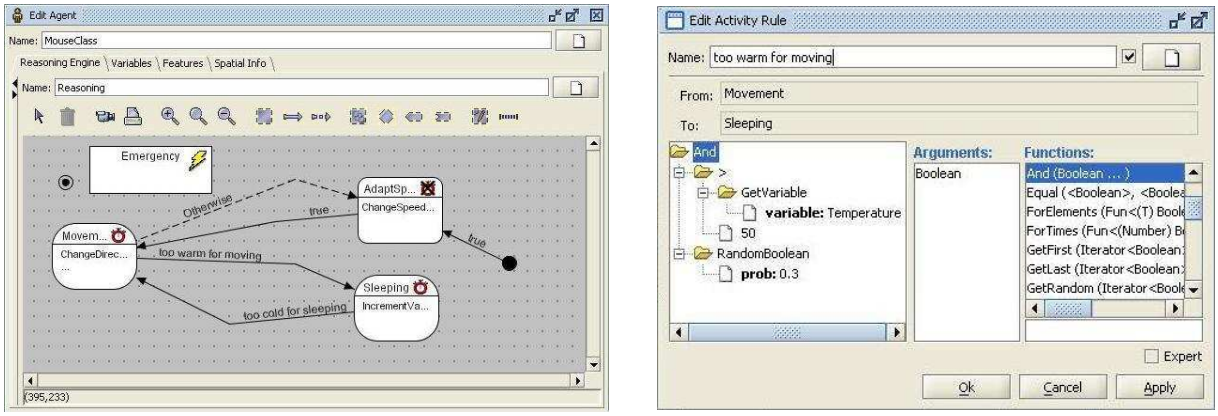

FIGURE 2.7 – Description avec SeSam du comportement d'un agent dont la température augmente lorsqu'il se déplace dans l'environnement et diminue lorsqu'il se repose. La capture d'écran gauche illustre l'interface graphique utilisée pour spécifier l'activité d'un agent sous la forme d'un graphe. Celle à droite illustre l'interface graphique utilisée pour spécifier une activité de l'agent.

La notion de temps dans le comportement est gérée en marquant les activités comme *instantanées* ou *non instantanées*. Une activité non instantanée nécessitera un pas de temps de simulation avant de pouvoir passer à une autre activité.

Discussion

Maleva et SeSam fournissent des représentations graphiques permettant de concevoir plus simplement le comportement des agents. De plus, la structure de Maleva permet de réutiliser facilement des composants logiciels. Cette approche favorise ainsi la conception de simulations contenant des agents dont une partie du comportement est un motif se retrouvant dans le comportement d'autres agents.

Toutefois, l'absence de séparation entre action et sélection d'action dans ces approches complexifie la spécification transversale de la simulation. En effet, la sélection d'action fait partie de la « boite noire » des entités du phénomène : contrairement aux actions que les entités entreprennent, ils ne sont pas directement observables dans la réalité. Ils font donc intégralement partie des hypothèses émises sur le fonctionnement du phénomène. En séparant action et sélection d'action, la modification des hypothèses est rendue plus simple et peut être interprétée plus facilement.

Cette séparation se retrouve dans des approches réactives telles que le projet Manta [DF92] présenté en section 2.1.6. Elle constitue de plus l'une des caractéristiques fondamentales des architectures cognitives et hybrides.

2.2.2 Architectures cognitives

Les architectures cognitives permettent de représenter des comportements dits « intelligents » étant mus par des buts que les entités cherchent à atteindre. Différentes architectures permettent de modéliser ce genre de comportements. Elles incluent les architectures calquées sur le fonctionnement du cerveau humain et les architectures basées sur la planification délibérative ou réactive.

Architectures issues de la théorie de la cognition humaine

Les architectures issues de la théorie de la cognition humaine se retrouvent dans des plateformes comme Soar [LNR87] ou Act-R [ABB+04]. Elles structurent la mémoire de l'agent comme la mémoire d'un humain contenant : une mémoire procédurale, une mémoire épisodique, une mémoire sémantique, une mémoire de travail, *etc.*

Ce type d'architecture sépare les actions de la sélection d'action à l'aide d'une mémoire procédurale, contenant des règles [13] de la forme *condition/action*. La condition représente les pré-requis logiques portant sur la mémoire de travail pour que l'action de la règle puisse être déclenchée. L'action décrit les effets de l'exécution de la règle sur mémoire de travail ou sur l'environnement de l'agent.

Le comportement des agents est construit par un apprentissage par renforcement : l'agent choisit la *règle* la plus adéquate à utiliser pour son contexte actuel (son but et contenu actuel de ses différentes mémoires). Il évalue ensuite la qualité du résultat et réévalue sur cette base l'intérêt de la règle pour ce contexte. Une évaluation positive favorise son utilisation future dans le même contexte alors qu'une évaluation négative l'entrave.

Les relations de l'agent avec les autres entités de la simulation n'est pas mentionnée explicitement dans le modèle. Au mieux, la mention des entités partenaires d'une action sont mêlées à la déclaration des attributs lus et modifiés dans la mémoire de travail. Leur gestion est donc laissée à la discrétion du concepteur.

Ce type d'architecture est utilisé comme support à la recherche sur la cognition humaine. Elle est peu adaptée pour décrire des phénomènes tels que la coopération entre équipes d'agents ou la construction explicite de plans pour atteindre des buts. Ils nécessitent en effet une représentation des connaissances plus abstraite.

Architectures à planification délibérative

D'autres architectures cognitives manipulent des connaissances abstraites permettant de se détacher de la théorie de la cognition propre à la psychologie humaine. Plus précisément, elles définissent explicitement dans le modèle la notion de plan. Les plans décrivent un enchaînement de règles permettant à l'agent d'aller de son état courant à un état dans lequel le but du plan est atteint. On distingue deux façons de construire les plans.

La première se trouve dans les architectures à planification délibérative telles que Prodigy [VCP+95] et les plateformes basées sur le planificateur SHOP [NCLMA99] ou GraphPlan [BF97]. Dans ces approches, les agents disposent d'une mémoire procédurale similaire à celle trouvée dans les architectures dédiées à la théorie de la cognition humaine. Pour atteindre son objectif, un agent résonne sur les règles de sa mémoire procédurale qu'il peut initier. Plus précisément, il planifie une succession de règles dont il doit effectuer successivement les actions afin de parvenir à un état dans lequel un de ses buts est réalisé. Les plans sont construits dynamiquement par un chaînage de règles selon un processus récursif. Nous présentons ici un type de chaînage particulier appelé chaînage arrière afin d'illustrer l'utilisation de la mémoire procédurale dans ce cas.

Le chaînage arrière construit un plan en déterminant dans un premier temps l'ensemble des règles dont les actions amènent à un état où le but recherché est atteint. Chacune de ces règles constitue alors

13. appelées « opérateurs » dans Soar, ou « règle de production » dans Act-R

le premier maillon d'un plan. Ces règles ne peuvent être exécutées que si le contexte de l'agent vérifie leurs conditions. Si les conditions ne sont pas toutes vérifiées, alors un plan est construit pour parvenir à un état intermédiaire où elles le sont. Le plan permettant d'atteindre le but recherché est alors la concaténation du plan permettant d'atteindre l'état intermédiaire et du maillon permettant de passer de l'état intermédiaire au but recherché. Cette construction récursive s'arrête au moment où les conditions des règles sont vérifiées par le contexte actuel de l'agent.

Le comportement d'un agent consiste à sélectionner un des buts qu'il cherche à atteindre, puis à sélectionner l'un des plans permettant de l'atteindre et enfin à exécuter l'action du premier maillon du plan.

Les architectures cognitives à planification délibérative nécessitent de fortes ressources en termes de mémoire et de calcul. En effet, le nombre de chaînes de règles qu'il est possible de construire croît exponentiellement avec le nombre de règles à la disposition de l'agent. Ces calculs peuvent être optimisés par des heuristiques estimant quels plans risquent de ne pas atteindre le but recherché. Malgré ces optimisations, une simulation ne peut contenir qu'un ensemble restreint de ce type d'agents pour être efficace en termes de temps de calculs. De plus, le comportement n'est décrit qu'en fonction de buts. Ces approches ne sont donc pas adéquates pour la conception de comportements réactifs.

Architectures à planification réactive

Pour éviter les problèmes liés à l'explosion combinatoire du nombre de plans qu'il est possible de construire une autre approche de la planification appelée planification réactive peut être utilisée. Elle consiste à construire les plans de manière réactive en se basant sur la décomposition de buts en sous-buts. Contrairement à la planification délibérative, où un plan est construit dynamiquement en construisant une chaîne de règles, les plans sont représentés de manière statique dans les règles. Chaque règle exprime dans ses conditions le but qu'elle permet de traiter et décrit dans ses actions les sous-buts qu'elle introduit pour le décomposer. Le plan est alors l'ensemble des règles qui ont permis de traiter son but et les sous-buts ayant été introduits. Cette approche est utilisée dans des plateformes comme APEX [Fre98], Jack [BHRH00], Jadex [BPL05], JAM [Hub99], Jason [BH06] ou PRS [GL87].

Exemple : la plateforme Jack

La plateforme Jack [BHRH00] repose sur une architecture à planification réactive fondée sur le modèle général Belief Desire Intention (BDI) [RG91]. La spécification du comportement y est centrée sur les *plans* qui constituent les éléments de base de la mémoire procédurale ainsi que sur les *évènements*. Un agent peut recevoir ou émettre des évènements représentant soit un stimuli interne (un but qu'il cherche à résoudre), soit un stimuli externe (un message envoyé par d'autres agents ou un évènement envoyé par l'environnement et perçu par l'agent).

Les plans sont représentés sous la forme de règles composées de 4 éléments :
- un *évènement* (« event ») auquel il répond ;
- un critère de *pertinence* (« relevance ») de l'évènement pour le plan. Ce critère permet de vérifier que des éventuels paramètres de l'évènement n'ont pas de valeur interdite pour ce plan ;
- un *contexte* (« context ») décrivant les conditions sous lesquelles le plan peut être effectué ;
- un *corps* (« body ») décrivant les actions initiées par l'agent lorsqu'il exécute ce plan.

Le corps du plan permet l'émission d'évènements internes pour ajouter des sous-buts à l'agent et ainsi effectuer la planification réactive. Il permet aussi l'émission d'évènements externes pour communiquer avec d'autres agents.

Le comportement d'un agent se déclenche lors de la réception d'un évènement et consiste à sélectionner un plan lui répondant, sous réserve que les critères de pertinence et de contexte soient vérifiés. Le corps du plan sélectionné est alors exécuté. Dans le cas où plusieurs plans peuvent répondre à un évènement, le plan utilisé est soit sélectionné aléatoirement, soit sélectionné en fonction de l'ordre dans lequel ils ont été ajoutés dans la mémoire procédurale, soit sélectionné par un méta-plan.

Jack fournit donc une architecture où il y a séparation entre mémoire procédurale et comportement. Elle permet de plus de faire de la planification réactive en décomposant des buts en sous-buts lors de l'exécution de plans. Elle nécessite donc moins de ressources computationnelles qu'une plateforme à

planification délibérative pour exécuter le comportement d'un agent. De plus, la notion d'évènement peut représenter un stimulus, un but ou un message entre deux agents. Elle unifie donc la représentation de comportements réactifs, cognitifs (planification réactive) voire coopératifs.

La modélisation des interactions s'y limite toutefois à l'utilisation de diagrammes d'interaction issus d'AUML [OPB00] et donc aux échanges de messages entre agents.

Discussion

Suite à cette étude des architectures cognitives, nous pouvons faire deux constats. Premièrement, les relations observables entre agents ne sont pas représentées ou limitées à des échanges de messages et des protocoles d'interaction. Leur intégration aux règles et au comportement des agents n'est pas détaillée et est laissée à la discrétion du concepteur, qui doit bien souvent distribuer le protocole d'interaction entre les différents agents y participant.

Deuxièmement, la spécification des buts et/ou situations auxquelles répondent une règle permet aux architectures à planification réactive de concevoir des agents étant à la fois réactifs et cognitifs avec un mécanisme unifié. Toutefois, elles nécessitent la décomposition de tous les buts qu'un agent peut avoir en sous-buts. Elles alourdissent donc la phase de conception de la simulation. De plus, ces décompositions ne sont pas exhaustives. Les agents ne bénéficient donc pas de l'ensemble des alternatives s'offrant aux agents dans les architectures délibératives.

Afin de concilier ces deux aspect et fournir un compromis entre efficacité en termes de temps de calculs et généricité, des architectures dites hybrides peuvent être utilisées.

2.2.3 Architectures hybrides

Les architectures hybrides consistent à faire coexister plusieurs architectures comportementales (appelées « couches ») et donc plusieurs représentations des connaissances au sein d'un même agent. Le comportement d'un agent y est le fruit de l'activation contextuelle d'une de ces architectures. L'activation peut se faire selon deux approches : celles dites « verticales » et celles dites « horizontales ».

Exemple d'architecture horizontale : TouringMachines [Fer92]

Les architectures hybrides horizontales consistent à activer en parallèle toutes les couches et à utiliser une unité de contrôle pour choisir la couche qui régira le comportement de l'agent. Dans l'architecture TouringMachines [Fer92], cette unité de contrôle est similaire à l'architecture de subsomption : une action proposée par la couche réactive (« Reactive Layer ») peut être ignorée si la couche de planification (« Planning Layer ») trouve une action plus intéressante. La couche de planification peut elle-même être ignorée au profit d'une couche de modélisation (« Modeling Layer ») qui permet d'anticiper les modifications futures de l'environnement et donc de choisir des plans ayant le plus de chances d'aboutir à une résolution du but de l'agent. Les critères déterminant comment la subsomption a lieu sont considérés comme dépendants du domaine d'application et sont donc décrits de manière ad-hoc.

Exemple d'architecture verticale : InteRRaP

Dans les architectures hybrides verticales, le comportement d'un agent est décidé en trois phases (les deux dernières phrases sont illustrées sur la figure 2.8 pour l'architecture hybride InteRRaP). La première phase consiste à étudier ce que l'agent perçoit, à déterminer les situations nécessitant une réaction immédiate ainsi qu'à identifier les buts que l'agent cherche à résoudre. La seconde phase détermine le degré de cognition que l'agent doit mettre en œuvre pour déterminer l'action qu'il doit effectuer. Pour cela, elle inspecte séquentiellement chaque couche jusqu'à atteindre une couche disposant des compétences suffisantes pour identifier les actions devant être exécutées. La dernière phase détermine les actions à effectuer en descendant de la couche activée la plus cognitive à la couche la plus réactive.

L'architecture InteRRaP [FMP95] repose sur trois couches :

- une couche de comportement réactive (« Behavior Based Layer » ou « BBL ») qui choisit de manière réactive une règle à exécuter (appelée « Pattern of Behavior ») afin de répondre à une situation nécessitant une réaction immédiate ;

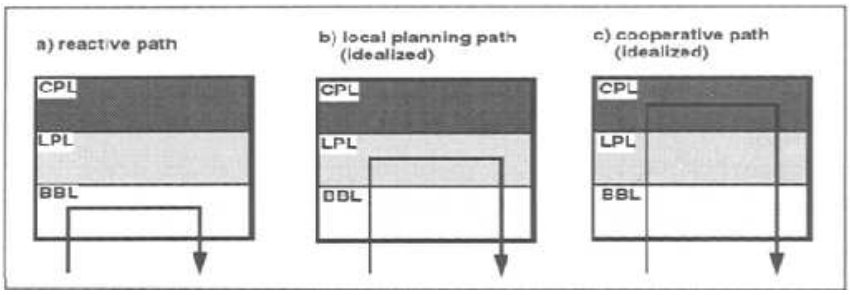

FIGURE 2.8 – Cheminement au travers des couches de l'architecture InteRRaP décrivant un comportement purement réactif (voir a), cognitif (voir b) ou coopératif (voir c).

- une couche de planification locale (« Local Planning Layer » ou « LPL ») qui est activée lorsque la couche de comportement réactive n'est pas suffisamment compétente pour répondre à la situation de l'agent. Elle consiste à construire un plan partiel permettant d'atteindre un des buts de l'agent. Les nœuds de ce plan partiel sont soit des plans, soit une situation permettant de déclencher une règle de la couche inférieure ;
- une couche de planification coopérative (« Cooperative Planning Layer » ou « CPL ») qui est activée lorsque la couche de planification locale est incapable de trouver des plans permettant d'atteindre les buts de l'agent. Elle consiste à décomposer le but en sous-buts et à résoudre chaque but. Ces buts sont résolus soit par la couche de planification locale de l'agent, soit par la couche de planification locale d'un agent acceptant de coopérer.

Discussion

Les architectures hybrides fournissent un compromis permettant de concilier aspects réactifs, cognitifs, voire coopératifs au sein d'une architecture pour agents. Ces architectures permettent de cumuler les avantages des différentes approches mentionnées dans cette section. En effet, la première couche réactive gère efficacement les situations ne nécessitant pas de planification. Ensuite, une couche de planification réactive gère efficacement les buts le plus souvent poursuivis en décrivant une bibliothèque de plans réactifs. Une couche de planification délibérative gère ensuite les cas non-supportés par la librairie de plans en construisant des plans de manière dynamique. Enfin, une dernière couche de planification coopérative gère les but ne pouvant pas être atteint sans le concours d'autres agents. Pour cela, elle permet de négocier avec d'autre agents le sous-traitement de certains de ses buts. Ces couches permettent de répondre efficacement aux différentes situations de l'agent : une couche cognitive peu efficace en termes de calculs ne sera appelée que si ses couches inférieures, qui sont plus efficaces, sont incapables de trouver une action à effectuer.

Il faut toutefois noter que la conception de tels agents se focalise sur ce que les agents sont capables de faire et sur leur comportement. Les interactions entre agents ne sont pas modélisées explicitement et sont considérées comme un effet de bord de leur comportement. La seule mention de ces relations se situe au sein de la couche de planification coopérative, où une négociation reposant sur des protocoles d'interactions est utilisée pour déléguer des buts à d'autres agents. De plus, l'utilisation de telles approches pose un problème de modélisation. En effet, elle suppose d'être capable d'établir une distinction claire entre tous les niveaux et de répondre à des questions non triviales. Par exemple : « Comment déterminer si un plan doit être spécifié dans la couche de planification réactive ou s'il doit être construit dynamiquement par la couche de planification délibérative ? ».

2.2.4 Synthèse relative aux architectures comportementales

Les architectures comportementales pour agents sont nombreuses et leurs propriétés varient du tout au tout.

Les architectures réactives permettent de décrire des comportements limités ne faisant que réagir à

des situations, mais permettent de déterminer rapidement les actions à entreprendre. La séparation entre actions et sélection d'action y est peu usuelle, impliquant ainsi des problèmes lors des révisions de modèle ainsi que lors de la conception graduelle d'une simulation. Ces architectures sont toutefois intéressantes dans le cas de simulation multi-agents contenant beaucoup d'entités de par leur efficacité en termes de calculs.

Les architectures cognitives permettent de décrire des agents plus élaborés pouvant raisonner au prix de décisions plus lentes. La séparation entre actions et sélection d'action y est usuelle et permet donc de mieux supporter les problématiques inhérentes aux révisions de modèle. Ces architectures sont donc intéressantes dans le cas de simulations contenant un faible nombre d'entités qui agissent en planifiant leurs actions.

Les architectures hybrides fournissent un compromis entre les architectures réactives et cognitives. Elles permet des décisions rapides lorsque cela est possible à l'aide de décisions réactives ou des décisions plus lentes lorsqu'il est nécessaire de raisonner pour déterminer les actions à entreprendre. La séparation entre action et sélection d'action y est usuelle autant pour les aspects réactifs que cognitifs. Ces architectures sont donc adéquates aux simulations contenant beaucoup d'agents et aux simulations contenant des agents pouvant planifier.

Bien que les architectures hybrides fournissent le meilleur compromis entre architectures réactives et cognitives, elles n'ont toutefois pas à être utilisées de manière systématique pour concevoir une simulation. En effet, selon les domaines d'application, les agents peuvent ne pas disposer de facultés réactives ou cognitives. Par exemple, en éthologie le comportement des animaux peut être décrit de manière satisfaisante avec des règles réactives, alors qu'en sociologie certains comportement ne sont mus que par des buts. Dans de tels cas, utiliser des architectures hybrides peut s'avérer moins efficace en termes de calculs puisque certaines couches ne sont jamais utilisées. Par conséquent, l'idéal serait de décrire autant que faire se peut les connaissances et le contenu de la mémoire procédurale indépendamment du comportement des agents et de décider par la suite du comportement à utiliser. Les architectures existantes permettent de le faire partiellement, grâce à la séparation entre actions et sélection d'action. Néanmoins, la structure des règles change en fonction de l'architecture utilisée pour exprimer le comportement. Il faudrait donc pouvoir trouver une représentation valide pour toutes les architectures.

La conception transversale d'une simulation nécessite de prendre en compte les interactions entre les entités présentes dans la simulation dès les premières étapes du processus de conception de simulations. Les architectures présentées dans cette section ne permettent pas ou très peu de telles spécifications. Elles sont en effet au mieux cantonnées à la description de protocoles d'interaction, *i.e.* d'échanges de messages entre agents. Pourtant, les relations entre entités ne sont pas restreintes à ce cas. Dans la section qui suit, nous étudions les différentes relations pouvant exister entre les entités d'une simulation.

2.3 Description de la dynamique macroscopique du phénomène

Les simulations explicatives visent à expliquer l'apparition d'une dynamique macroscopique observable dans l'environnement (*i.e.* un phénomène émergent) à l'aide des actions effectuées individuellement par les entités le composant. Toutefois, les comportements individuels seuls n'expliquent pas l'apparition du phénomène. Ce sont les interactions qu'ont les agents les uns sur les autres qui en sont à l'origine. Ces interactions ont différentes natures, que nous proposons d'étudier dans cette section.

La terminologie désignant ces influences ne fait pas consensus, principalement à cause de la forte connotation de certains des termes utilisés dans le domaine des systèmes multi-agents. Ainsi, avant de nous attaquer à l'étude de l'état de l'art, nous tenons à rappeler le sens premier du terme central à cette section : les « interactions ».

2.3.1 La notion d'interaction

Selon la neuvième édition du dictionnaire de l'académie française, le terme interaction a deux sens :

> « **PHYS.** Action réciproque de deux ou plusieurs corps. *La gravitation est un phénomène d'interaction entre deux corps. Interaction électromagnétique. Interaction forte, action attractive entre particules, qui assure la cohésion des noyaux atomiques. Interaction faible, qui se*

manifeste par des forces d'attraction ou de répulsion entre particules, responsables en particulier de la radioactivité bêta.
Par ext. Influence qu'exercent les uns sur les autres des phénomènes, des faits, des objets, des personnes. *L'interaction des faits économiques et politiques.* » [fra10]

En son sens premier, le terme interaction désigne les quatre interactions élémentaires entre deux corps à l'origine de tout phénomène de la physique. Elles incluent les interactions nucléaire fortes et faibles, l'interaction électromagnétique et la gravitation. En simulation informatique, le terme interaction est donc principalement compris en son sens par extension désignant tout type d'influence qu'un phénomène peut avoir sur un ou plusieurs autres phénomènes. Ce second sens est tout à fait adéquat pour désigner ce que nous étudions dans cette section. Cette définition tranche toutefois avec la définition usuelle que l'on rencontre dans les systèmes multi-agents que nous présentons dans la sous-section qui suit.

2.3.2 Protocoles d'interaction

Le sens le plus couramment rencontré de l'interaction dans le domaine des systèmes multi-agents est bien plus précis et connoté que la définition générale présentée dans la section précédente. Il est exprimé dans le contexte d'applications réparties devant communiquer entre elles pour atteindre un but qu'elles se sont fixé. Il n'est donc pas spécifique à la simulation, ce qui explique pourquoi nous parlons ici de processus de conception de systèmes multi-agents plutôt que de processus de conception de simulations.

Dans ce contexte, une **interaction** est un terme utilisé par abus langage pour désigner un **protocole d'interaction**, *i.e.* un ensemble structuré d'échanges de messages entre plusieurs entités afin d'atteindre un but particulier. Par exemple, le protocole d'interaction Contract Net de la FIPA [FIP10a] décrit les échanges de messages permettant à un agent d'effectuer un appel à propositions, puis d'accepter ou refuser les propositions provenant d'autres agents.

La description d'un protocole d'interaction peut se faire sous diverses formes. Les plus répandues sont les diagrammes issus d'UML similaires aux diagrammes de séquence, par exemple les diagrammes d'interaction de AUML [OPB00]. Dans ces diagrammes, une interaction a lieu entre plusieurs agents jouant chacun un rôle. Le diagramme d'interaction décrit les différents scénarios d'échanges de messages pouvant avoir lieu entre les rôles du protocole. La figure 2.9a illustre un tel protocole dans le cas d'un client effectuant une commande. À ces diagrammes peuvent s'ajouter des diagrammes d'activité décrivant le comportement d'un agent à la réception d'un des messages du protocole. La figure 2.9b illustre un tel diagramme dans le cas où un agent jouant le rôle *Order Processor* reçoit une requete *placeOrder* lors d'une commande.

AUML fournit ainsi des outils permettant de décrire des protocoles d'interaction. Toutefois, l'intégration de ces interactions au comportement des agents pose problème. En effet, les protocoles sont en pratique distribués dans le comportement des différents agents y participant sous la forme d'une réaction à la réception d'un message. Un protocole d'interaction n'a donc pas de représentation explicite dans les connaissances des agents. En conséquence, à chaque révision du modèle, une modification du protocole d'interaction implique la ré-implémentation partielle de tous les agents pouvant jouer un rôle dans cette interaction.

De plus, nous avons montré précédemment que le modèle seul n'est pas suffisant pour spécifier une simulation. Il faut pouvoir concevoir graduellement le modèle à l'aide d'une méthodologie. Il se pose alors un problème antérieur à la description du protocole d'interaction : dans quel cas un protocole d'interaction se révèle-t-il nécessaire pour modéliser le système multi-agents ?

Des approches permettent de remédier à certains de ces problèmes. Le langage IOM/T [DTH05] permet par exemple d'éviter la fragmentation du protocole lors de l'implémentation. De même, l'ingénierie des besoins orientée par les buts [14] que l'on retrouve dans Tropos [BPG+04] ou PASSI [Cos05] intègre la conception des protocoles d'interaction dans un processus de conception incrémental.

Exemple : Réification des protocoles à l'implémentation avec IOM/T [DTH05]

Le langage IOM/T [DTH05] donne les moyens d'éviter la fragmentation des protocoles d'interaction en les spécifiant chacun dans une unité logicielle unique. Comme pour la plupart des protocoles d'in-

14. traduction de : « Goal-Oriented Requirements Engineering »

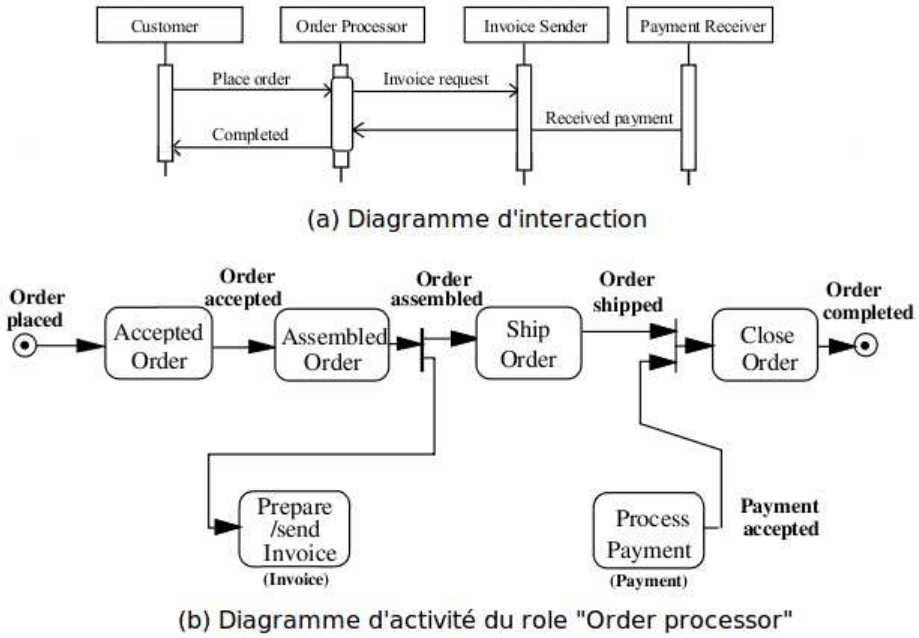

FIGURE 2.9 – Exemple d'un protocole d'interaction décrit avec AUML. Le diagramme d'interaction (a) décrit les échanges de messages permettant à un client d'effectuer une commande. Le diagramme d'activité (b) décrit le comportement d'un agent nommé « Order Processor » en réaction à la réception d'une requête de commande provenant d'un client.

teraction, la structure générale des échanges de messages y est décrite avec un diagramme d'interaction provenant d'AUML. Toutefois, IOM/T ne se restreint pas à cette simple description : chaque diagramme d'interaction est automatiquement transformé en code dans un langage proche de JAVA (voir figure 2.10). Le protocole d'interaction est implémenté sous la forme d'un unique fichier décrivant précisément :

- les rôles impliqués dans l'interaction ;
- le protocole d'échanges de messages entre les agents. Ce protocole décrit non seulement les messages échangés entre les rôles, mais aussi les instructions exécutées par les agents entre les envois de messages.

Le protocole est implémenté sous la forme d'une séquence représentant les interventions successives des différents rôles dans l'interaction. Chaque intervention d'un rôle est représentée dans un bloc, dans lequel les différentes actions entreprises par l'entité jouant ce rôle sont décrites. Ces actions incluent l'envoi d'un message, l'attente d'une réponse, des structures conditionnelles portant sur les messages reçus, *etc.* Ainsi, la sémantique du protocole n'est plus éparpillée au sein des différents agents participant à l'interaction, mais centralisée dans une seule représentation. Le protocole peut donc être réutilisé par des agents très différents, tout en minimisant les éventuelles modification possibles des agents faisant suite à une modification du protocole d'interaction.

Cette représentation du protocole d'interaction pose un problème fondamental : chaque agent dispose de spécificités ne pouvant être exprimées directement dans le protocole d'interaction. Par exemple, dans un protocole de ping itératif, le nombre d'itérations de la requête dépend de critères propres à l'agent initiant ce protocole. Pour remédier à ce dilemme, dans IOM/T les actions entreprises par les agents dans le protocole peuvent être exprimés sous la forme de fonctionnalités abstraites (« functionnalities ») relatives à chaque rôle. L'implémentation de la fonctionnalité se fait au sein des agents pouvant jouer ce rôle. Un rôle est alors similaire à une interface au sens des langages orientés-objet, où les méthodes désignent les fonctionnalités devant être implémentées par un agent jouant ce rôle. Ainsi, un protocole d'interaction est réifié en une unique entité logicielle tout garantissant la diversification du comportement des agents participant à l'interaction.

```
interaction PingProtocol {
 role Sender {
  AID getTarget();
  boolean isContinue();
  void knowAsDead();
 }
 role Receiver {
  boolean doesReply();
  ACLMessage res;
 }
 protocol {
  while (Sender.isContinue()) {
   play Sender {
    ACLMessage ping = new ACLMessage();
    ping.setReceiver(getTarget());
    ping.setContent("(ping)");
    ping.setPerformative("QUERY_REF");
    sendAsync(ping); //# m1
   }
   play Receiver {
    ACLMessage ping = recvBlock(); //# m1
    res = ping.createResponse();
   }
   if (Receiver.doesReply()) {
    play Receiver {
     res.setContent("(alive)");
     res.setPerformative("INFORM");
     sendAsync(res); //# m2
    }
    play Sender() {
     ACLMessage msg = recvBlock(); //# m2
    }
   } else {
    play Receiver {
     res.setContent(ping.getContent());
     res.setPerformative("NOT_UNDERSTOOD");
     sendAsync(res); //m3
     play Sender {
      ACLMessage msg = recvBlock(); //# m3
      knowAsDead();
     }
    }
   }
   play Sender {
    ACLMessage msg = new ACLMessage();
    msg.setReceiver(getTarget());
    msg.setContent("(end-of-loop)");
    msg.setPerformative("INFORM");
}}}}
```

FIGURE 2.10 – Description d'un protocole de ping itératif dans le langage IOM/T.

Ainsi, des librairies de protocoles d'interactions précis et réutilisables sont construits. L'absence de dispersion du protocole au sein des agents facilite de plus les itérations du processus de conception. Toutefois, cette représentation ne décrit pas comment un agent choisit d'initier un tel protocole. L'intégration du protocole d'interaction dans le comportement des agents n'est donc pas complète.

Exemple : Spécification incrémentale des protocoles d'interaction avec Tropos

Tropos [BPG+04] est une méthodologie permettant de concevoir de manière transversale un système multi-agents. Nous n'en donnons qu'une description schématique dans cette section : seul le processus donnant lieu à l'identification des interactions entre entités nous intéresse.

Cette méthodologie repose sur la décomposition des buts auxquels l'application doit répondre et les dépendances existant entre ces buts. Ces buts peuvent être de deux types :

- les « buts principaux » (« hard goal » dans le texte) représentent une fonctionnalité que le système doit fournir. Par exemple « recenser des informations relatives aux musées d'une ville » ;
- les « buts secondaires » (« soft goal » dans le texte) représentent la qualité du service fourni par le système multi-agent. Par exemple « maximiser la satisfaction des clients ».

Chaque but est représenté sous la forme d'un nœud dans un graphe dirigé. Chaque arc représente une relation de décomposition entre « buts principaux » ou une relation d'influence positive ou négative de la résolution d'un « but principal » sur un « but secondaire ».

La méthodologie Tropos conçoit une simulation en quatre phases distinctes. La première et la seconde phase, intitulées « Early Requirements Phase » et « Late Requirements Phase » ont pour objectif d'identifier tous les buts du système développé et leur décomposition par la construction puis le raffinement d'un graphe. Lors de la troisième phase, intitulée « Architectural design Phase », les « buts principaux » sont regroupés par les fonctionnalités qu'ils expriment puis sont attribués à des agents dont le rôle est de satisfaire lesdits buts. La dernière phase, intitulée « Detailed Design Phase », consiste à décrire d'un point de vue algorithmique tous les éléments identifiés par les étapes précédentes. La troisième phase permet d'identifier quels seront les protocoles d'interaction à mettre en place dans l'application. En effet, chaque

arc connectant deux buts distribués à des agents différents implique la mise en place d'une communication entre eux et donc d'un protocole d'interaction.

Ainsi, l'utilisation d'un graphe permet d'identifier de manière intuitive les protocoles d'interaction devant être mis en place. Cette représentation favorise la communication entre experts en informatique et non-informaticiens, voire la spécification directe d'une partie du système par des non-informaticiens.

Discussion

Les échanges de messages structurés sous la forme de protocoles sont une première forme d'interaction (*i.e.* d'influence entre le comportement des différentes entités d'une simulation. Ce type d'interactions est particulièrement approprié dans le cas des simulations en sciences sociales, où les différentes entités peuvent communiquer entre elles par actes de langage. Il est aussi particulièrement adapté aux simulations nécessitant une forme de coopération entre agents afin d'atteindre un but commun. Leur intégration dans le processus de conception et dans l'implémentation du système multi-agents n'est pas un problème trivial et est encore un sujet actif de recherche. Des propositions telles que IOM/T, Madkit ou Tropos y contribuent en donnant les moyens de réifier le protocole d'interaction ou d'identifier les interactions ayant lieu entre agents sous la forme d'un graphe au sein d'un procédé transversal. Ces approches souffrent toutefois de problèmes liés à l'implémentation du comportement des agents qui limitent leur intérêt pour la spécification de simulations.

En effet, une première façon d'implémenter le protocole d'interaction consiste à le réifier sous la forme d'une entité logicielle séparée des agents. Dans ce cas, la décision d'initier un protocole d'interaction est laissée à la discrétion du concepteur. Elle est donc réalisée de manière ad hoc. C'est par exemple le cas d'IOM/T ou de Madkit. Dans le cas où le protocole est fragmenté et distribué aux agents, il est possible de le traduire en un ensemble de règles déclenchées par la réception d'un message. Dans ce cas, il y a effectivement séparation entre action et sélection d'action au prix d'une dispersion du protocole d'interaction. C'est par exemple le cas dans Ingenias [SPGS06].

Ces solutions reposent sur des principes qui s'opposent et rendent donc difficile la constitution d'un compromis pourtant nécessaire à la conception de simulations.

2.3.3 Interactions et Actions

Les échanges de messages ne sont pas l'unique forme d'interaction entre entités. Odell *et al.* [OPB00] en donnent une intuition en représentant des actions telles que le clonage d'un agent par un autre agent sous la forme de diagrammes d'interaction. Par conséquent, ce qui est usuellement considéré comme une action n'est-il pas en fait une forme d'interaction ?

Cette question nous amène à considérer le sens commun du terme « action », ainsi que son sens pratique dans les systèmes multi-agents. D'après la neuvième édition du dictionnaire de l'académie française, le terme action a 5 sens dont deux propres au théâtre et au droit que nous ne mentionnons pas :

> « n.f. XIIe siècle. Emprunté du latin *actio*, dérivé de *actum*, supin de *agere*, « agir ».
> Opération d'un agent quelconque ; résultat de cette opération. *Action de* entre dans la définition des substantifs dont le sens correspond à celui d'un verbe ; le même substantif exprimant en général l'opération, le processus et le résultat de l'action considérée, on précise : « *Action de ; résultat de cette action* » [. . .].
> **1.** Exercice effectif de la faculté d'agir, par opposition à *Rêverie, Inertie, Intention, Spéculation, Contemplation, Hésitation.*[. . .]
> **2.** Production d'un effet ou manière d'agir sur quelque chose ou quelqu'un.[. . .]*La gravitation est une action à distance.*[. . .]
> **3.** Manifestation extérieure de la faculté d'agir ; ce qu'on fait. *Accomplir une bonne, une mauvaise action.*[. . .] »[fra10]

Le sens 2 est particulièrement intéressant, car il correspond exactement au problème mentionné au début de cette section. Une action peut être interprétée comme le moyen qu'un agent a d'influer sur quelque chose d'autre et donc comme une interaction entre plusieurs agents. La présence de plusieurs entités de la simulation dans une action justifie l'intérêt des diagrammes d'interaction et de séquence pour les décrire. Pourtant, dans les architectures existantes, les règles constituent une représentation des actions mais ne

font pas mention explicitement des participants autres que l'agent à leur origine. Les autres sujets de l'action sont relégués au rang de paramètres et ne sont pas mentionnés lors du processus de conception.

Certaines approches spécifiques à des domaines d'application font toutefois exception à cette règle. C'est en particulier le cas de la chimie organique, par exemple dans l'approche proposée par Desmeulles dans sa thèse [Des06], ou par Cannata *et al* [CCM08]. Ce type de représentation est aussi utilisée dans d'autres approches telles que le modèle Mascaret développé par Querrec [Que02] sous le nom d'interactions réactives.

Exemple : Réification des interactions par Desmeulles

Desmeulles propose un modèle générique permettant de réifier la notion d'interaction en son sens large, c'est à dire tout type d'action impliquant simultanément plusieurs entités de la simulation. Il fournit une application de ce modèle générique au cas des simulations en biochimie, où les seules interactions sont soit les déplacements, soit les réactions chimiques. Nous cantonnons notre présentation à ce cas particulier, afin d'avoir des propos les plus explicites possibles quant à la modélisation d'un phénomène avec cette approche.

Le modèle proposé repose sur les concepts de réaction (*i.e.* réaction chimique), d'espèce, de compartiment, de réacteur et d'organisation. Un réacteur correspond à un milieu dans lequel des réactions ont lieu. Un réacteur est caractérisé par un compartiment décrivant sa forme et son volume ainsi que par des espèces décrivant chacune le nombre de moles d'une espèce chimique qu'il contient. Enfin, une organisation est composée d'un ensemble de réacteurs dont les espèces peuvent éventuellement interagir afin de décrire les flux de molécules d'un réacteur à un autre. La figure 2.11 illustre ces concepts pour une application de transcription de l'ADN d'une cellule.

De tels modèles sont conçus à l'aide d'outils graphiques tels que le graphe présenté dans la figure 2.11. Cette représentation graphique est ensuite transformée en un fichier XML représentant explicitement les réactions, espèces, compartiments, réacteurs et organisations. Ce fichier alors exploité pour générer un simulateur dans lequel les interactions sont implémentées sous la forme d'agents. Leur comportement consiste à calculer régulièrement le nouveau nombre de moles de chaque espèce chimique impliquée cette réaction.

Cette approche se conforme à l'ensemble des pré-requis permettant de faciliter la conception de simulations. En effet, il propose un processus de conception comprenant la construction d'un modèle et son implémentation. Un morphisme entre modèle et implémentation est garanti. De plus, le modèle fait apparaître de manière explicite à la fois entités et influence des différentes entités entre elles sous la forme d'interactions. Il favorise donc la conception transversale d'une simulation.

Toutefois, dans ce modèle une interaction ne peut représenter qu'une loi de l'environnement. Cela implique que l'exécution d'une interaction ne fait pas partie d'un processus décisionnel : elles ont toujours lieu dès que leur contexte le permet. En ce sens, cette approche est similaire aux modèles de calcul à base de règles comme les P-systèmes [Pŏŏ] ou la chemical abstract machine [BB90]. Cette représentation n'est pas conçue pour être utilisée dans des architectures telles que la subsomption ou tout autre architecture où un agent a la possibilité de choisir l'action qu'il effectue. Cette approche n'est donc valable que pour des simulations contenant des agents purement réactifs.

Discussion

Toute action entreprise par un agent peut être interprétée comme une interaction entre plusieurs agent. La description de la dynamique du phénomène étudié passe donc par l'identification des actions pouvant lier les entités de la simulation. Actuellement, la notion d'entité subissant une action initiée par un autre agent n'est explicite que dans deux cas :

- les protocoles d'interaction, dont l'intégration dans la connaissance des agents et dans le processus de délibération des agents n'est toutefois pas complète et doit être réalisée manuellement par le concepteur ;
- les interactions purement réactives qui sont déclenchées de manière systématique sans passer par un procédé de délibération.

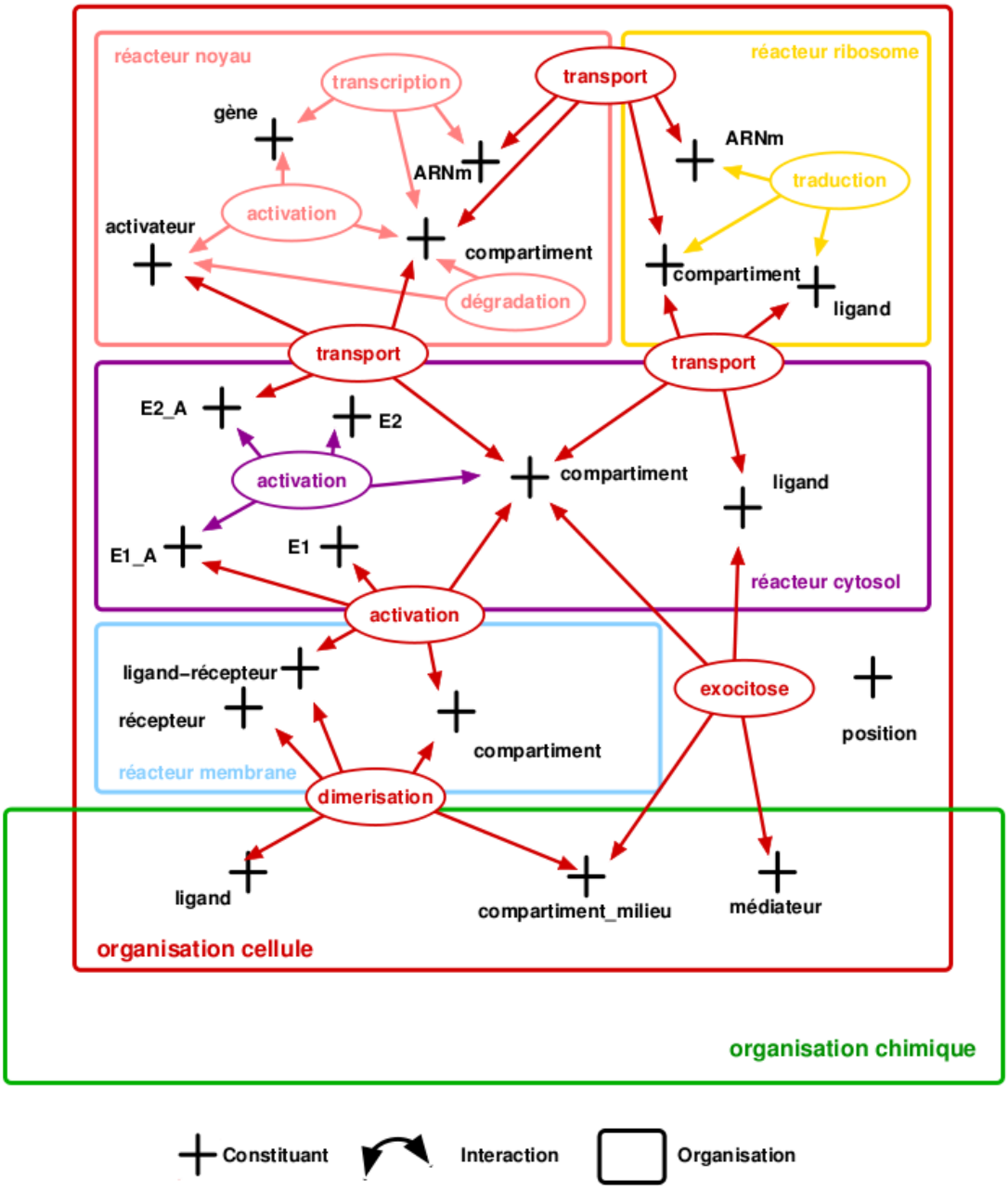

FIGURE 2.11 – Représentation graphique du modèle d'une simulation en biochimie avec l'approche de Desmeulles [Des06], dans le cas de la transcription de l'ADN dans une cellule.

La plupart des simulations explicatives font régulièrement apparaître des actions impliquant plusieurs entités sans pour autant être dans les deux cas précédemment mentionnés. Il s'avère donc nécessaire de leur trouver une représentation adéquate. Cette idée rejoint une théorie développée en psychologie appelée théorie des affordances, qui cherche à décrire les actions qu'il est possible de faire sur les éléments contenus dans un environnement.

2.3.4 Théorie des affordances

Les affordances font partie d'une théorie à l'origine de la théorie écologique de la perception (en anglais « ecological psychology ») initialement développée par le psychologue James J. Gibson en 1977 [Gib77]. Ces notions ont été ensuite reprises et étendues par Donald Norman en 1988 [Nor88] sous le concept d'affordances perçues, afin de concevoir des interfaces homme-machine.

Selon la théorie initiale émise par Gibson, tout élément contenu dans un environnement dispose de propriétés physiques (*e.g.* une surface, une masse) qui caractérisent les actions qu'une entité peut faire avec cet élément. Ces actions, appelées affordances de cet élément pour l'entité, sont relatives à l'entité cherchant à agir. Par exemple, si un élément de l'environnement est étendu, rigide et à hauteur de genoux relativement à une entité, alors il permet à l'entité d'effectuer l'action « s'asseoir ». C'est le cas d'une chaise, d'un lit ou d'une souche d'arbre pour un être humain. Ces éléments ne le permettent par contre pas pour un éléphant puisque les propriétés de rigidité et de hauteur ne sont pas vérifiées pour lui. La caractérisation de ces actions est objective et indépendante du fait qu'une entité est consciente ou pas qu'il lui est possible de les effectuer. En complément à cette théorie, Norman décrit les affordances perçues comme l'ensemble des actions qu'une entité a conscience de pouvoir faire.

Ces courants de pensée considèrent que chaque élément de l'environnement peut être le sujet d'actions lui étant intrinsèques. Ils peuvent être transposés au cas des simulations informatiques, afin de représenter les éléments de la mémoire procédurale des agents. Ce constat a en particulier inspiré les travaux de Cornwell *et al* [COST03] ainsi que de Papasimeon [Pap09]. Nous décrivons ces deux approches selon trois points de vue :

- la philosophie générale de l'approche ;
- la représentation concrète des affordances dans le modèle et en particulier leur exploitation pour définir le comportement des agents ;
- l'intégration des affordances au processus de conception de simulations.

Dans ces deux approches, il y a distinction entre d'une part les objets qui représentent toute entité présente dans l'environnement, et d'autre part les agents qui sont un sous-ensemble des objets pouvant percevoir et agir sur les autres objets de la simulation.

Les affordances en simulation selon Cornwell *et al* [COST03]

Dans l'approche de Cornwell *et al*, la définition des affordances perçues repose sur quatre principes :

- Chaque objet définit un ensemble de « types perçus » [15] lui étant rattachés.
- Chaque type perçu contient un ensemble d'actions.
- Chaque agent perçoit les objets au travers d'un ou plusieurs types perçus.
- Chaque type perçu dispose de règles de perception qui s'il est activé ou non. S'il est désactivé, un agent ne peut pas le percevoir.

Selon ces principes, les affordances d'un agent pour un objet correspondent à l'ensemble des actions contenues dans les types perçus de l'objet perçus par l'agent. Les affordances perçues sont alors les affordances contenues dans un type perçu activé.

La représentation d'une affordance dans ce modèle est propre au modèle comportemental utilisé dans l'application Performance Moderator Function Server (PMFserv). Dans ce modèle, un agent sélectionne l'affordance dont il effectue l'action en fonction de préférences calculées à l'aide d'un modèle émotionnel fondé sur l'utilisation de buts. Pour être utilisée dans cette architecture comportementale, chaque affordance décrit les effet de son action sur l'objet, mais aussi ses éventuels effets sur les buts de l'agent, sur ses émotions ainsi que sur ses préférences.

Dans cette approche, la conception d'un modèle est incrémentale et suit le procédé suivant :

15. « perceptual type » dans l'article original

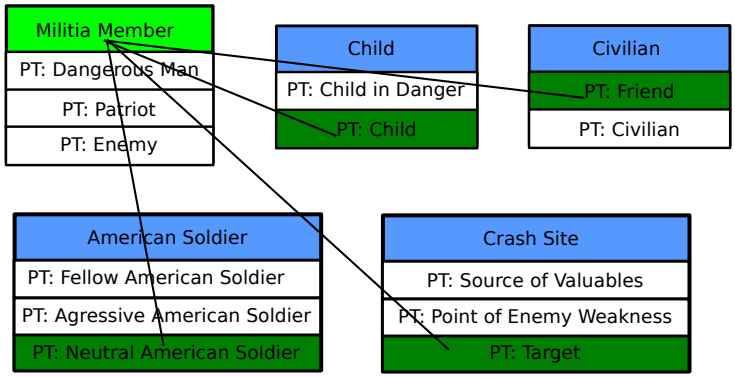

FIGURE 2.12 – Aperçu de l'outil « Object/Perception Editor » développé par Cornwell *et al* [COST03] pour éditer les affordances dans une simulation. Cette image illustre le graphe faisant apparaître les interactions entre entités de la simulation. Le trait entre « Militia Member » et « PT :Neutral American Soldier » se lit « Un agent Militia Member interagit avec un agent American Soldier via le type perçu PT :Neutral American Soldier ».

1. identifier les agents et les objets de la simulation ;
2. identifier les types perçus associés à chaque objet. Ils peuvent éventuellement provenir d'une bibliothèque de types perçus ;
3. relier agents et types perçus dont ils ont connaissance à l'aide de graphes (voir figure 2.12) ;
4. identifiées les règles de perception de chaque type perçu et les actions contenues dans chaque type perçu ;
5. décrire les actions.

La conception des affordances est donc progressive et répond en partie au besoin d'une approche transversale de conception d'une simulation. De plus, des bibliothèques réutilisables d'objets et de types perçus sont construits et contribuent donc grandement à la simplification des révisions du modèle ainsi qu'à la conception de nouvelles simulations.

Les affordances en simulation selon Papasimeon [Pap09]

Dans l'approche de Papasimeon, la définition des affordances est faite dans le cadre d'une architecture Belief-Desire-Intention (BDI) [RG91]. Deux problèmes liés à l'exploitation de la théorie des affordances en simulation y sont considérés :

- Comment utiliser la théorie des affordances au sein d'architectures BDI existantes sans pour autant en modifier profondément la structure ?
- Comment sont calculées les affordances d'un agent à un moment donné de la simulation ?

Pour répondre au premier problème, la conception des connaissances des agents est centrée sur trois éléments distincts :

- les « actions possibles » qui décrivent les actions atomiques qu'un agent peut entreprendre ;
- les « intentions possibles » qui décrivent une séquence sémantique plus ou moins complexe d'actions possibles qu'un agent peut exécuter ;
- les « affordances » qui associent à un objet de l'environnement une intention possible que l'agent peut exécuter si certaines conditions sont rencontrées.

Une affordance entre un agent et un objet est donc représentée sous la forme d'une règle condition/action. Les conditions y décrivent quelles propriétés l'agent et l'objet doivent vérifier pour que l'affordance soit perçue. Les actions consistent à ajouter une intention possible aux intentions de l'agent. Cette représentation n'émet aucune hypothèse quant à l'architecture comportementale des agents qui peut aussi bien être réactive que cognitive.

En son sens le plus strict donné par Gibson, les affordances doivent être définies dans l'environnement et non plus dans la mémoire procédurale des agents. Pourtant, un agent

doit pouvoir manipuler les affordances pour produire son comportement. Pour répondre à ce problème, Papasimeon propose deux approches différentes pour modéliser les affordances :

- s'appuyer sur le sens strict des affordances selon Gibson et stocker toutes les affordances dans l'environnement ;
- s'appuyer sur une définition plus souple des affordances et considérer que les affordances perçus d'un agent sont contenues dans sa mémoire procédurale.

Le comportement d'un agent à un instant t est défini dans cette architecture comme une séquence de 5 étapes :

1. Le recensement des entités perçues ;
2. Le recensement des affordances de l'agent avec les entités perçues. Ce recensement est plus ou moins efficace en termes de calculs selon la façon dont sont modélisées les affordances ;
3. La sélection d'une affordance en fonction de l'état mental de l'agent ;
4. L'identification de l'intention possible associée à l'affordance sélectionnée ;
5. L'exécution des actions décrites dans l'intention possible récupérée.

La différence de cet algorithme avec ceux de l'architecture BDI usuelle se trouve principalement dans la deuxième et la troisième étape.

Le modèle formel fourni par cette approche ne décrit pas concrètement la délibération de l'agent, et donne donc pas lieu à un processus de conception transversal. D'ailleurs, aucun outil d'aide à la conception n'est mentionné dans cette thèse.

Discussion

La transposition de la théorie des affordances à la simulation fait apparaître explicitement les interactions existant entre agents et entités. Cette approche est donc indiquée afin de construire un processus transversal de conception de simulations. En pratique, les approches étudiées ici rencontrent deux types de limites.

L'approche proposée par Cornwell *et al* et le modèle sous-jacent basé sur les types perçus ont été conçus pour favoriser le génie logiciel dans leur architecture appelée PMFServ. Dans cette architecture, la connaissance est représentée de manière très spécifique et repose sur certains concepts ne permettant pas son utilisation dans des agents réactifs ou dans les architectures cognitives « classiques ».

L'approche proposée par Papasimeon fournit un modèle formel précis pour décrire les affordances et leur lien au comportement des agents. Toutefois, ce modèle est voulu générique et applicable à des agents pouvant utiliser des processus de délibération très différents. En conséquence, aucune description précise du contenu des règles représentant les affordances n'est fournie. Cette absence de description précise augmente le risque de mélanger actions et sélection d'action. Pour illustrer ce point, nous reprenons un exemple développé par Papasimeon de son manuscrit de thèse [Pap09].

Cet exemple modélise un jeu de capture du drapeau impliquant des Tanks représentés par des agents. Les conditions sous lesquelles un tank α tente d'INTERCEPTER un autre tank ϵ sont :

$$IsType(\alpha, Tank) \wedge HasStatus(\alpha, Alive) \wedge IsType(\epsilon, Tank) \wedge HasStatus(\epsilon, Alive) \wedge OnOpposingSides(\alpha, \epsilon) \wedge HasCapability(\alpha, InterceptCapability) \wedge IsVisible(\alpha, \epsilon) \wedge IsClosestEnemyTank(\alpha, \epsilon) \wedge HasIntention(\alpha, DefendBase)$$

Cette description mêle :

- conditions physiques requises portant sur l'agent et l'objet. Par exemple « hasStatus(a, Alive) » précise qu'un Tank doit être en vie pour intercepter un autre tank ;
- conditions sous lesquelles une affordance est perçue. Par exemple « HasCapability(a,e) » précise que cette affordance est perçue uniquement si α est capable d'Intercepter ;
- conditions propres à la délibération. Par exemple « IsClosestEnemyTank(a,e) » qui précise que l'interception se fait sur le tank ennemi le plus proche.

L'ajout de conditions propres à la délibération réduit donc la possibilité de réutiliser ces règles. Afin de favoriser le génie logiciel, il est nécessaire de guider davantage la conception de ces conditions.

La théorie des affordances est donc prometteuse et permet de représenter les interactions entre entités provenant de tout type d'action. Toutefois, les solutions actuelles n'offrent pas de compromis satisfaisant entre généricité de l'approche utilisée, la possibilité de pratiquer le génie logiciel et conception graduelle du modèle.

2.3.5 Synthèse relative à la dynamique macroscopique du phénomène

Les éléments à l'origine d'un phénomène émergent sont le comportement des entités constituant le phénomène, mais aussi les interaction entre les entités. Le processus de conception de simulations doit donc non seulement se focaliser sur la description du comportement des agents, mais aussi sur ces interactions.

Le sens le plus commun du terme interaction dans les systèmes multi-agents désigne des protocoles d'échanges de messages entre agents. Ces protocoles sont initiés par un agent afin de répondre à un de ses propres besoins/buts. Leur conception et leur intégration dans le modèle est complexe, puisqu'elle nécessite :

- de **centraliser la description du protocole complet en une seule entité logicielle** afin de faciliter ses futures modifications ;
- de **prendre en compte les spécificités de chaque agent** participant au protocole ;
- de **concrétiser le lien entre l'aspect descriptif du protocole et son intégration dans le comportement des agents**.

Dans les solutions existant actuellement, un compromis n'est trouvé au mieux qu'entre deux de ces trois éléments.

Les protocoles d'interaction permettent de décrire des échanges de messages et supposent que les agents aient des buts. Ils constituent le moyen privilégié de modéliser une grande part de phénomènes en sciences sociales. Toutefois, ce ne sont pas les seules formes d'interactions rencontrées en simulation. En effet, dans des domaines d'applications tels que la biochimie, l'enjeu principal d'une simulation est de décrire les différentes réactions chimiques à l'origine du phénomène observé. Dans ce contexte, une interaction n'est pas un échange de message répondant à un but, mais une réaction chimique ayant lieu systématiquement entre plusieurs espèces chimiques s'y prêtant. Ce type d'interaction s'apparente beaucoup à la notion de lois de l'environnement auxquelles sont sujettes les entités d'une simulation. La notion de but leur étant extérieure, de telles interactions sont modélisées différemment des protocoles d'interaction.

Bien que ces modèles de l'interaction diffèrent, ces deux approches mettent en exergue l'intérêt d'identifier les interactions pouvant avoir lieu lors de la simulation sous la forme d'un graphe. Ils restent toutefois insuffisants pour concevoir des agents dont la délibération repose sur une architecture réactive exempte de buts. Une caractérisation des interactions dans le cas général est donc nécessaire pour poursuivre cet effort.

En appliquant aux simulations les principes de la théorie des affordances développée par Gibson et Norman, la notion d'interaction peut être généralisée à toute action qu'un agent peut faire sur une autre entité. La mention des entités sujettes aux actions est rendue explicite permettant ainsi :

- d'utiliser des représentation sous la forme de graphes pour modéliser toute action ayant lieu entre entités ;
- d'intégrer la notion d'interaction au comportement des agents.

En pratique, l'intégration au comportement est actuellement effectuée de deux manières :

- intégrée finement au modèle mais restreinte à une architecture décisionnelle particulière. Par exemple des agents émotionnels ayant des buts dans la proposition de Cornwell *et al* ;
- intégrée à gros grain au modèle en décrivant uniquement les principes à mettre en œuvre. Dans ce cas, le modèle est incomplet et aucun guide méthodologique de conception n'est défini. C'est par exemple le cas de l'approche de Papasimeon [Pap09].

Une représentation des connaissances s'inspirant de la théorie des affordances semble être la meilleure alternative pour à la fois :

- commencer le processus de conception par une vue d'ensemble sur la dynamique du phénomène ;
- intégrer les interactions au comportement des agents.

Toutefois, les approches actuelles ne sont pas suffisantes en l'état pour modéliser des simulations, en particulier lorsqu'il s'agit de modéliser des agents dont la délibération est réactive.

2.4 Synthèse du chapitre

Dans cette thèse, nous cherchons à **faciliter la conception de simulations large échelle**. Précédemment, nous avons déterminé que les agents et systèmes multi-agents reposent sur des principes généraux qui, en théorie, facilitent la conception de telles simulations. En pratique, des approches très différentes permettent de modéliser et implémenter un système multi-agents. Ce chapitre a exploré et analysé ces approches selon trois perspectives différentes, afin d'en dégager des principes facilitant la conception de simulations large échelle. Nous synthétisons ici la conclusion de ces trois études.

Comment parvenir à une implémentation ?

Pour parvenir à l'implémentation d'une simulation, il convient d'utiliser **une approche transversale**, caractérisée par :

- un **modèle précis, décrivant le phénomène à différents niveaux d'abstractions** : les organisations (lorsqu'il y en a), les interactions, ce que les agents sont capables de faire et le comportement des agents ;
- une plateforme permettant d'implémenter le modèle ;
- une **méthodologie de conception, permettant de construire progressivement le modèle** ;
- un procédé permettant d'**implémenter automatiquement** un modèle.

Une telle approche a de nombreux avantages du point de vue de la simulation :

- La conception d'une simulation commence par des descriptions abstraites de haut niveau ne nécessitant que peu de connaissances en informatique. Les experts du domaine sont donc plus facilement impliqués dans la conception du modèle ;
- L'automatisation de l'implémentation évite les erreurs provenant d'une implémentation manuelle ;
- Les différents niveaux d'abstraction permettent d'intégrer progressivement des informations dans le modèle. Il est ainsi plus aisé de concevoir des modèles contenant une grande quantité d'informations, ce qui facilite la construction simulations large échelle.

La spécification du modèle à différents niveaux d'abstraction n'est possible qu'avec une représentation des connaissances et une architecture adéquates. Pour les caractériser, nous avons étudié les modèles et architectures existants selon deux perspectives : l'architecture interne des agents ainsi que le lien entre caractérisation des interactions et spécification du comportement des agents.

Quelle architecture interne utiliser ?

Une approche transversale de conception modélise le niveau microscopique de la simulation (les agents) en deux niveaux d'abstraction : ce qu'un agent est capable de faire (ses capacités) et la façon dont il choisit les actions qu'il entreprend (sa sélection d'action). Pour modéliser ces deux niveaux, il faut :

- **séparer le déclaratif** (les éléments décrits dans le modèle) **du procédural** (les algorithmes permettant d'implémenter le modèle) ;
- représenter les **actions** des agents de manière **générique** et **indépendante de leur comportement** ;
- représenter les actions des agents sous la forme de **règles condition/effet**.

Sous ses conditions, il devient possible :

- d'**unifier la représentation des connaissances** des agents et ainsi modéliser aussi bien des agents réactifs que cognitifs avec une représentation unique ;
- de **repousser l'introduction d'hypothèses de fonctionnement à la fin du processus de conception**. En effet, contrairement aux interactions et aux actions entreprises par les agents, qui sont observables dans le phénomène, le comportement des agents est uniquement le fruit de suppositions. Sa spécification tardive permet donc de focaliser sur des descriptions plus objectives du phénomène ;
- de concevoir des **bibliothèques d'actions réutilisables**. Les révisions du modèle et la construction de nouveaux modèles dans le même domaine d'application sont ainsi facilitées.

Comment concrétiser le lien entre interactions et comportement des agents ?

Afin de passer de la description d'un niveau d'abstraction élevé (les interactions entre agents) à un niveau plus concret (le comportement des agents), il faut :

- que **toute action impliquant simultanément plusieurs agents** soit représentée par une **interaction** ;
- que les interactions entre agents soient modélisées à l'aide d'un **graphe** ou d'une forme équivalente où les agents sont des nœuds et les interaction sont des arcs ;
- intégrer les interactions au comportement des agents en les modélisant sous la forme de **règles conditions/effets**.

Sous ces conditions, il devient possible :

- de modéliser un phénomène à un haut niveau d'abstraction, sous une forme facile à manipuler ;
- d'établir un lien direct entre interactions et comportement des agents, facilitant ainsi le passage de descriptions de haut niveau à des descriptions proches de l'implémentation ;
- de faire le lien entre modèle formel et concepts spécifiques au domaine simulé, puisque la notion d'interaction peut autant modéliser une réaction chimique que la discussion entre deux agents, ou encore l'ingestion d'un aliment.

Sur quelles approches existantes faut-il se reposer ?

D'un point de vue méthodologique, les solutions s'approchant le plus des propriétés recherchées sont issues de l'ingénierie de systèmes multi-agents dirigée par les modèles. En effet, elles expriment toutes les propriétés caractérisant une approche transversale. Toutefois, ces approches sont prévues pour la conception d'applications réparties. Elles se focalisent sur l'inter-opérabilité entre agents logiciels hétérogènes et restreignent donc la notion d'interaction à des actes de langage. Le lien entre interactions et comportement des agents n'y est donc pas satisfaisant.

Du point de vue de la structure du modèle et de son implémentation, les solutions s'approchant le plus des propriétés recherchées sont issues de la théorie des affordances. En effet, elles caractérisent de manière satisfaisante le lien entre interactions et comportement des agents. Toutefois, ces approches se focalisent sur des architectures comportementales cognitives. De plus, la méthodologie permettant de construire graduellement un modèle n'y est pas définie. Elles ne constituent donc pas une approche transversale.

Notre objectif dans cette thèse est de construire une approche bénéficiant des avantages méthodologiques, logiciels et de représentation des connaissances énoncés ici. Afin de concrétiser une telle approche, nous décrivons dans la partie suivante comment construire un modèle centré sur les interactions et comment intégrer ce modèle à une approche de conception transversale bénéficiant des avantages identifiés dans ce chapitre.