Simulations multi-agents multi-niveaux : quatre patterns de conception

Philippe Mathieu, Gildas Morvan, Sébastien Picault

Simulations multi-agents multi-niveaux : quatre patterns de conception

Philippe Mathieu[a] philippe.mathieu@univ-lille.fr
Gildas Morvan[b] gildas.morvan@univ-artois.fr
Sébastien Picault[a] sebastien.picault@univ-lille.fr

[a]Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL (équipe SMAC)
Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

[b]Univ. Artois, EA 3926, Laboratoire de Génie Informatique et d'Automatique de l'Artois (LGI2A), Béthune, France

Résumé

Cet article introduit quatre patterns de conception, définis à partir de l'identification de « situation-types » minimales rencontrées dans la littérature et destinés à systématiser et simplifier la conception de simulations multi-agents multi-niveaux. Ces simulations ont pour but de gérer des entités appartenant à des niveaux d'abstraction ou d'organisation différents mais couplés. Pour chaque pattern, nous présentons des cas d'utilisation ainsi que les structures de données et algorithmes associés. À des fins de généricité, ces patterns font appel à une description unifiée des capacités d'action et d'évolution des agents. Nous proposons ainsi un cadre conceptuel et opérationnel précis pour les concepteurs de simulations multi-niveaux.

Mots-clés : *simulation multi-agents, simulation multi-niveaux, patterns de conception*

Abstract

This paper introduces four design patterns, drawn from minimal typical situations found in the literature, and meant to systematize and simplify the design of multilevel multiagent simulations. Such simulations aim at handling entities which belong to different, but coupled, abstraction or organization levels. For each pattern, we present use cases and associated data structures and algorithms. For genericity purposes, these patterns rely upon a unified description of the capabilities for action and change of the agents. Thus, we propose a precise conceptual and operational frame for the designers of multilevel simulations.

Keywords: *multiagent simulation, multilevel simulation, design patterns*

1 Introduction

Les SMA se sont d'emblée construits comme des systèmes à deux niveaux : le niveau « microscopique », celui des agents dotés d'un certain comportement, et le niveau « macroscopique », celui du système pris comme un tout. Mais bien souvent, ce dernier reste *implicite*, au sens où il est « extérieur » (hétérogène) aux agents : il est d'une *nature différente* (ensemble de spécifications ou de problématiques à résoudre avant la conception du SMA, ensemble de descripteurs agrégés pendant ou après le fonctionnement du SMA), et s'il exerce une *rétroaction* sur le comportement des agents, ce n'est souvent que de façon *implicite*.

Expliciter l'articulation entre les niveaux microscopique et macroscopique reste donc une question ouverte. Non seulement on ne dispose pas à l'heure actuelle de méthode générale pour ce faire, mais les questions scientifiques sous-jacentes à cette problématique dans le domaine SMA sont encore assez largement impensées. Or, la période actuelle voit s'effectuer une mutation de l'usage des SMA et des sujets de recherche correspondants. De disciplines comme l'éthologie ou la sociologie, qui s'intéressaient à un nombre d'agents relativement restreint (quelques centaines ou milliers), les applications des SMA se déplacent vers des systèmes à large échelle (écologie, biologie moléculaire ou cellulaire, réseaux sociaux, marchés financiers, transports, etc.) où les approches classiques rencontrent des limitations. Les questions liées à l'organisation des systèmes, et plus précisément à leur organisation en sous-systèmes, prennent le pas sur celles concernant les architectures individuelles de décision.

Pour répondre à ces nouveaux besoins, on voit émerger des *simulations multi-agents multi-niveaux*, c'est-à-dire des SMA dans lesquels on cherche à donner une représentation explicite du niveau macroscopique (et de tout niveau intermédiaire pertinent), si possible sous forme d'agents. Cela peut prendre plusieurs formes et répondre à des objectifs variés, parmi les-

quels : la prise en compte dans le modèle de niveaux d'abstraction « utiles » ou « pertinents » pour les experts du domaine ; l'adaptation dynamique du niveau de détail de la simulation, par exemple pour économiser du temps de calcul lorsque c'est possible ; le couplage de modèles hétérogènes permettant de simuler des processus à différentes échelles ; ou encore la détection de comportements collectifs intéressants ou de structures spatiales émergentes, avec ou sans réification (agentification).

Nous partons du constat, dressé par [5], qu'il n'existe pas aujourd'hui de caractérisation précise ou univoque de ce qui relève ou non du multi-niveaux, que ce soit en matière d'objectifs, de structure du système, d'architecture d'agent ou d'algorithmes. Tout au plus peut-on identifier des problématiques récurrentes, pour lesquelles de nombreuses implémentations sont proposées.

Notre objectif ici est de montrer que les simulations multi-agents multi-niveaux font appel à la combinaison de situations élémentaires, qui peuvent être caractérisées à partir d'un petit nombre de critères généraux, et pour lesquelles il existe des solutions d'implémentation récurrentes. Autrement dit, nous cherchons à identifier et décrire des *design patterns* multi-niveaux au même sens que [4] dans le domaine du génie logiciel.

Le plan de l'article est le suivant : dans la section suivante, nous exposons un certain nombre de prérequis opérationnels sur lesquels nous nous appuyons et décrivons notre méthode d'analyse et d'identification des patterns. Nous donnons ensuite la description détaillée de chacun d'entre eux (nom, caractéristiques, algorithmes, cas d'utilisation). Enfin, nous discutons un certain nombre de points notamment méthodologiques, théoriques ou pratiques, avant de conclure.

2 Démarche proposée

La démarche que nous souhaitons entreprendre vise à caractériser les situations récurrentes observées en pratique dans les simulations multi-niveaux, à en donner une typologie, à déterminer quelles réponses sont les plus adaptées à chaque type de besoins. Pour cela, il est nécessaire de fixer au préalable les hypothèses minimales concernant les outils manipulés, en l'occurrence les agents et les moteurs de simulation. Toute simulation multi-agents qui souscrit à ces

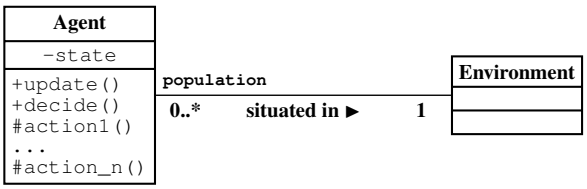


FIGURE 1 – Structure élementaire d'un agent et sa relation à l'environnement.

critères doit pouvoir mettre en œuvre les patterns que nous proposons sans autre prérequis.

2.1 Prérequis opérationnels

Un agent est une entité située dans un environnement, disposant d'un *état* non accessible directement (on notera dans la suite $s_t(a)$ l'état de l'agent a à l'instant t) ainsi que de *primitives* correspondant à ses capacités élémentaires de perception et d'action. Nous posons que l'état de l'agent peut, d'une part, suivre une dynamique d'évolution propre à l'agent, et d'autre part être affecté par les actions de l'agent et par celles des autres agents. L'agent doit donc gérer deux processus clefs en principe indépendants : l'évolution de son état au cours du temps, et la sélection d'actions (autrement dit le choix de primitives d'actions à effectuer en réponse à son état et à ses perceptions). Par souci de simplification, nous supposons dans la suite que le déclenchement respectif de chacun de ces processus passe par *au plus* un point d'entrée public (méthode, procédure, fonction...), que nous nommons par convention `update` et `decide` respectivement. C'est donc ce point d'entrée `decide` qui effectue la séquence classique perception-décision-action (fig. 1).

Les primitives d'action des agents n'ont pas vocation à être exécutées directement par le moteur de simulation, mais leur exécution est nécessaire lors des interactions entre agents. On peut donc considérer que leur accès n'est autorisé que pour la réalisation effective des comportements des agents.

L'autonomie d'un agent réside essentiellement dans le fait que les seuls points d'accès publics soient `update` et `decide`, afin de garantir qu'il n'y a aucune autre façon de manipuler l'état de l'agent, ni de fausser ses perceptions, ni de provoquer ses actions. En outre, ces deux points d'accès doivent être régis par un *ordonnanceur* qui est le seul habilité à déterminer à quel moment l'état de l'agent doit être recalculé et à quel moment une décision doit être prise.

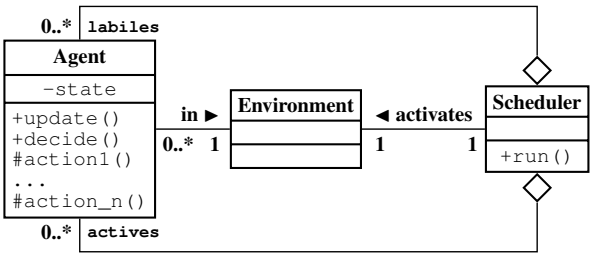


FIGURE 2 – Conservation par l'ordonnanceur des listes d'agents actifs et labiles.

Cet ordonnanceur peut être global au système, comme c'est souvent le cas en simulation, ou local à l'agent, par exemple l'horloge interne d'un robot.

Bien évidemment un agent peut ne pas offrir de point d'entrée `update` s'il ne change jamais spontanément d'état, ou pas de point d'entrée `decide` s'il ne peut rien faire par lui-même. Dans ce dernier cas, l'usage veut qu'on ne parle pas d'agent mais de ressource, ou d'artefact, ou d'objet actif, etc. De telles distinctions ne nous semblent pas particulièrement pertinentes dans le contexte qui nous intéresse. Elles constituent en effet une *restriction* de la notion d'agent qui nous semble d'autant plus arbitraire que le passage de la simulation multi-agents classique à la simulation multi-niveaux peut, comme nous allons le montrer, impliquer une certaine perte d'autonomie des agents : il faudrait donc, au fur et à mesure des processus d'agrégation ou de désagrégation, transformer des agents en artefacts ou ressources et vice-versa.

Nous préférons donc conserver le bénéfice d'une homogénéité maximale des entités présentes dans le modèle, sachant que ce traitement uniforme n'a pas de coût en temps de calcul s'il est pris en compte de façon appropriée dans le moteur de simulation [6]. Cela suppose uniquement de savoir identifier les agents disposant d'un point d'entrée `udpate` (que nous appelons dans la suite *agents labiles*) et ceux disposant d'un point d'entrée `decide` (que nous appelons *agents actifs*) : ici, l'ordonnanceur maintient à jour deux listes (fig. 2).

L'un des choix majeurs lors de la réalisation d'un simulateur concerne la gestion de la prise de parole par les agents. Cette partie, appelée « ordonnanceur » peut, si elle est réalisée sans une attention particulière, amener une grande variabilité dans les résultats obtenus. Ici, l'ordonnanceur doit gérer deux aspects : la mise à jour de l'état des agents (`update`) et la prise de décision des agents (`decide`).

L'ordonnanceur, *délibérément simple*, choisi pour expliquer nos patterns, est séquentiel et équitable [8] ; il repose sur l'identification explicite des agents *labiles* et des agents *actifs* et effectue à chaque instant t de la simulation le déclenchement de `update` pour tous les agents labiles, suivi du déclenchement de `decide` pour tous les agents actifs (algorithme 1).

Algorithme 1 : `run()` : Ordonnancement séquentiel et équitable des agents

```
for chaque instant t de la simulation do
    for a ∈ shuffle(labiles) do
        a.update()
    end
    for a ∈ shuffle(actives) do
        a.decide()
    end
end
```

2.2 Méthode d'analyse

Face à la croissance exponentielle, depuis le début des annéees 2000, de simulations s'appuyant sur des systèmes multi-niveaux, un travail de classification et de caractérisation s'impose, afin de réduire la complexité des cas particuliers à quelques critères clefs, permettant de décrire de façon univoque les processus effectivement mis en œuvre.

Nous avons cherché à identifier et hiérarchiser les questions pertinentes d'un point de vue opérationnel pour la conception d'un SMA multi-niveaux. Cette approche implique nécessairement une forme de *réductionnisme méthodologique* : pour s'affranchir des domaines d'application des simulations étudiées, et gagner en abstraction, en généralité et en réutilisabilité, nous avons d'abord voulu décrire des « situation-types » minimales, en nous focalisant sur chacune des transformations micro/macro ou macro/micro rencontrées. Ainsi, notre premier objectif était de construire des *schémas minimaux* basés sur deux **niveaux relatifs micro et macro**, le macro représentant une agrégation, une composition, un regroupement potentiel d'agents micro. Ces situations épurées peuvent être alors caractérisées par la *nature des relations existant entre ces deux niveaux*.

Les patterns que nous avons ainsi identifiés sont donc caractérisés par une succession de questions discriminantes portant sur les relations entre agents micro et agents macro (cf. fig. 3). Chaque pattern est déterminé en premier lieu par le **sens du changement de niveau** : soit

l'agrégation où des agents micro créent ou rejoignent un agent macro, soit la **désagrégation** qui produit des agents micro à partir d'agents macro. Cette distinction permet de caractériser les relations entre le **niveau source** et le **niveau cible**. Le deuxième critère, que ce soit pour une agrégation ou une désagrégation, consiste à décider soit si les agents du niveau source sont **détruits** (ce qui conduit de façon terminale au pattern ZOOM), soit **conservés** (quitte à subir d'éventuelles altérations). Dans ce cas, les agents du niveau source peuvent être **asservis** (fin de la branche, pattern MARIONNETTISTE) ou **autonomes** par rapport au niveau cible. Enfin, dans cette dernière situation, le couplage entre les deux niveaux peut être **unidirectionnel** en imposant au niveau cible de *refléter* l'état du niveau initial (pattern VUE) ou **bidirectionnel** en donnant à chaque niveau une totale autonomie comportementale (COHABITATION).

Bien entendu, ces patterns atomiques ont vocation à être *combinés* au sein des applications de simulation multi-niveaux pour chaque couple d'agents micro/macro. De plus, dans le cas où les changements de niveaux peuvent se faire dans les deux sens, on peut utiliser un pattern pour réaliser l'agrégation et un autre pour la désagrégation.

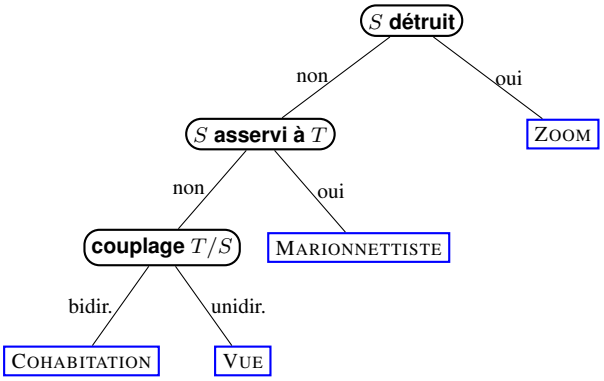


FIGURE 3 – Arbre d'identification des patterns pour un passage du niveau source S au niveau cible T

3 Description des patterns

Dans cette section nous nous proposons de décrire les quatre patterns que nous avons mentionnés ci-dessus en suivant pour chacun d'eux des exemples de la littérature qui correspondent selon nous à une mise en œuvre du pattern considéré ; la définition générale du pattern ; l'explicitation des algorithmes d'agrégation et de désagrégation induits ; puis ses avantages et limitations.

Pour ce faire, les agents du niveau micro sont notés dans la suite μ_i, l'agent du niveau macro correspondant M ; $s_t(a)$ désigne l'état de l'agent a à l'instant t ; on note $\mathcal{S}$ et $\mathbb{S}$ les espaces d'états respectifs des agents micro et macro, autrement dit : $\forall \mu_i, s_t(\mu_i) \in \mathcal{S}$ et $\forall M_j, s_t(M_j) \in \mathbb{S}$. Enfin, en vue d'assurer un couplage entre les niveaux considérés, on doit disposer d'une fonction permettant de définir l'état d'un agent macro à partir d'un ensemble d'agents micro (dans le cas de l'agrégation) ou réciproquement, de reconstruire n états d'agents micro à partir de l'état d'un agent macro (dans le cas de la désagrégation), soit :

$$\begin{array}{rccc} \textit{compose} : & \wp(\mathcal{S}) & \rightarrow & \mathbb{S} \\ & \{s_t(\mu_1), ..., s_t(\mu_n)\} & \mapsto & s_t(M) \end{array}$$

$$\begin{array}{rccc} \textit{decompose} : & \mathbb{S} & \rightarrow & \wp(\mathcal{S}) \\ & s_t(M) & \mapsto & \{s_t(\mu_1), ..., s_t(\mu_n)\} \end{array}$$

Ces deux fonctions sont naturellement dictées par le domaine d'application, et même plus spécifiquement par la nature des agents micro et macro considérés. Une simulation en comporte autant que de couples micro/macro impliqués.

Par ailleurs, la quantité d'information correspondant aux variables d'état d'un agent macroscopique est souvent moindre que celle contenue dans les variables d'état des n agents micro qui le composent : en conséquence, la plupart des fonctions concrètes $decompose$ ne peuvent affecter certaines caractéristiques des agents micro de façon déterministe à partir de l'état de l'agent macro. Elles doivent alors intégrer une *politique* d'affectation des variables indéterminées : par exemple, au moyen d'une valeur par défaut (constante ou périodique), d'une loi de probabilité ou d'informations statistiques.

3.1 ZOOM

Exemples. La modélisation du trafic routier fait souvent appel à trois niveaux d'observation nommés « micro » (véhicules), « méso » (groupes de véhicules) et « macro » (flux), qu'il faut parfois coupler pour simuler des réseaux hétérogènes de grande taille. Ce couplage doit garantir notamment la cohérence des simulations [2]. Le couplage utilisé lors de l'agrégation de véhicules individuels est généralement destructif. En effet, les agents micro (véhicules) sont détruits lorsqu'ils entrent dans une zone gérée par un agent macro, lequel modifie son état interne (caractérisé par les vitesse et densité moyennes du flux) en fonction de la densité et de la vitesse des agents micro.

La simulation hybride dynamique de flux de trafic [1] permet de désagréger dynamiquement une zone macro en un ensemble d'agents micro (véhicules). Le couplage utilisé pour la désagrégation est ici encore destructif. Lorsque l'on souhaite passer dynamiquement d'une représentation macro à une représentation micro dans une zone donnée, l'agent macro associé à cette zone est détruit et désagrégé en un ensemble d'agents micro représentant les véhicules.

Définition. Ces deux exemples sont caractéristiques du pattern ZOOM, mis en œuvre dans les cas de l'agrégation et de la désagrégation destructives : le passage des agents vers le niveau cible s'accompagne de leur destruction dans le niveau source (fig. 4).

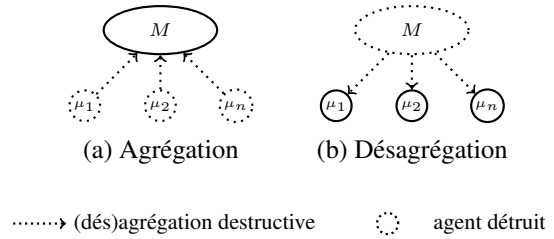


FIGURE 4 – Pattern ZOOM

Algorithmes. Les algorithmes mis en œuvre par le pattern ZOOM, que ce soit pour l'agrégation (algo. 2) ou la désagrégation (algo. 3), sont relativement simples, puisque les agents du niveau source sont détruits au profit des agents du niveau cible. La seule question épineuse est celle de la cohérence entre les niveaux, qui doit être assurée par les fonctions $compose$ et $decompose$.

Algorithme 2 : Agrégation dans le pattern ZOOM

créer l'agent macro M
$s_t(M) \leftarrow compose(\{s_t(\mu_1), ..., s_t(\mu_n)\})$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  `population`.remove(μ_i)
  `labiles`.remove(μ_i)
  `actives`.remove(μ_i)
end

Avantages et limitations. Ce type de couplage entre niveaux a pour mérite la simplicité et une extrême économie computationnelle puisque les seuls agents conservés dans le système sont ceux considérés comme pertinents à un niveau d'observation donné.

Algorithme 3 : Désagrégation dans le pattern ZOOM

créer n agents micro $\mu_1, ..., \mu_n$
$\{s_t(\mu_1), ..., s_t(\mu_n)\} \leftarrow decompose(s_t(M))$
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end
`population`.remove(M)
`labiles`.remove(M)
`actives`.remove(M)

En revanche, il entraîne une perte d'information généralement importante : la plupart des caractéristiques individuelles des agents micro disparaissent avec eux lors de l'agrégation. Certes la fonction $compose$ pourrait conserver l'ensemble des variables d'état des agents micro en tant que variable d'état de l'agent macro, mais c'est rarement le cas, ne serait-ce que pour ne pas payer le gain computationnel d'un important surcoût en mémoire. À l'inverse, il peut être nécessaire d'introduire de l'information dans le cas de la désagrégation. Il s'agit d'un choix explicite du modélisateur, exploitant souvent des données statistiques.

3.2 MARIONNETTISTE (PUPPETEER)

Exemples. Dans le projet RIVAGE, précurseur de la modélisation multi-niveaux [17] en hydrologie, les agents micro sont des *boules d'eau* en interaction. Ils peuvent s'agréger en agents macroscopiques *ravine* ou *mare*. Ces agents ne sont pas détruits mais leur comportement est gelé et géré par les agents ravine ou mare. De même dans GALAXIAN [7] (simulation de bataille spatiale), des agents *chasseurs* peuvent s'agréger pour constituer un agent *escouade*, qui « gèle » les comportements individuels des chasseurs et gère lui-même leur trajectoire et les ordres de tir. Quand l'agent escouade décide de se dissoudre, les chasseurs retrouvent leur autonomie première.

Quant à la désagrégation, SWARM [10], une des premières plateformes explicitement multi-niveaux, s'appuie sur une méthodologie de conception descendante (*top-down*) des modèles. Ainsi, le comportement d'un agent macro est défini comme le résultat des comportements des agents micro qui le composent. En revanche, il n'est pas possible de définir de rétroactions de l'agent macro vers les agents micro.

Définition. Dans le pattern MARIONNETTISTE, les agents du niveau source ne sont pas détruits mais « gelés », *i.e.* conservent leur état, susceptible d'évoluer selon sa dynamique propre, mais délèguent leur comportement aux agents du niveau cible (fig. 5). Ainsi les agents du niveau source (« marionnettes ») *perdent la capacité à exécuter leur méthode* `decide`. En revanche, leur état continuant d'évoluer, ils peuvent donc toujours exécuter `update`.

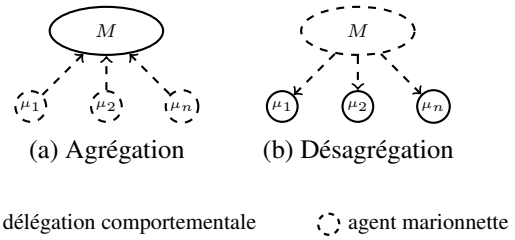


FIGURE 5 – Pattern MARIONNETTISTE

Algorithmes. Les algorithmes mis en œuvre pour le pattern MARIONNETTISTE, que ce soit pour l'agrégation (algo. 4) ou la désagrégation (algo. 5), supposent d'abord que l'ordonnanceur de la simulation ne considère plus les agents du niveau source comme *actifs* (pour ne plus activer `decide`). D'autre part, la délégation comportementale proprement dite, pour être assurée, suppose que les agents du niveau cible soient spécialisés (fig. 6) : outre leur comportement propre, ils sont en charge de l'activation de certaines actions des marionnettes (dépendantes du domaine).

Algorithme 4 : Agrégation dans le pattern MARIONNETTISTE

créer l'agent macro M
$s_t(M) \leftarrow compose(\{s_t(\mu_1), ..., s_t(\mu_n)\})$
M.puppets $\leftarrow \{\mu_1, ..., \mu_n\}$
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  `actives`.remove(μ_i)
end
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)

Avantages et limitations. Le pattern MARIONNETTISTE permet d'intégrer explicitement des entités du niveau cible dans la modélisation. Il permet également de simuler plus efficacement des systèmes composés de très nombreux agents. Contrairement au pattern ZOOM, il est non destructif et évite donc la perte d'information. De plus, lorsqu'on souhaite détruire les

Algorithme 5 : Désagrégation dans le pattern MARIONNETTISTE

créer n agents micro $\mu_1, ..., \mu_n$
$\{s_t(\mu_1), ..., s_t(\mu_n)\} \leftarrow decompose(s_t(M))$
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  μ_i.puppets $\leftarrow \{M\}$
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end
`actives`.remove(M)

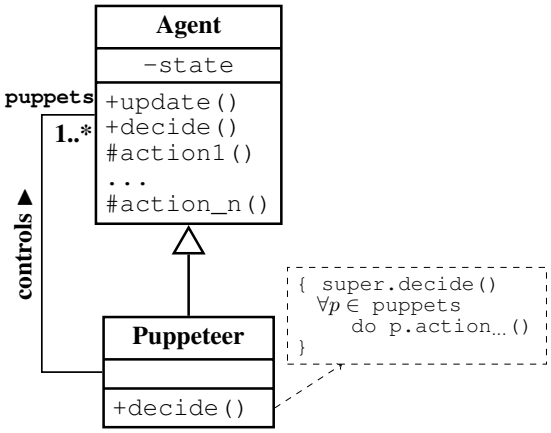


FIGURE 6 – Modélisation d'un agent MARIONNETTISTE (*Puppeteer*) capable de piloter l'action de ses agents « marionnettes ».

agents marionnettistes, on peut rendre facilement leur autonomie aux agents marionnettes.

Dans le cas de l'agrégation cependant, ce pattern ne peut être mis en œuvre facilement que si les comportements des agents agrégés sont similaires et que le comportement de l'ensemble est trivial. Ainsi, dans les exemples présentés ci-dessus, les agents agrégés sont inactifs (ils ne se déplacent pas et leur état n'évolue pas) ou caractérisés par une dynamique collective facile à piloter au niveau macro (mouvement linéaire).

Inversement dans ce cas de la désagrégation, une analyse poussée des travaux utilisant SWARM [10] semble indiquer que cette approche n'a été utilisée qu'au niveau technique pour dissocier la conception des outils d'observation, l'ordonnancement des différents types d'agents et enfin le comportement des agents eux-mêmes, sans que la possibilité de concevoir des modèles hiérarchiques complexes ait été exploitée. La désagrégation par MARIONNETTISTE semble donc pour l'instant limitée à des systèmes relativement simples caractérisés par une causalité ascendante et pouvant être

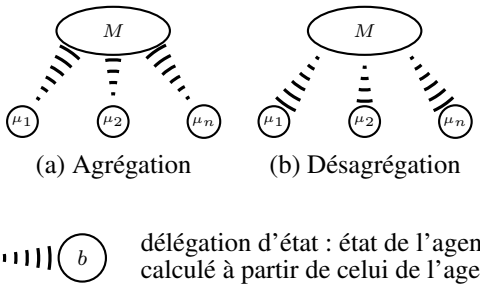
décomposés analytiquement en sous-systèmes fonctionnant de façon autonome.

3.3 VUE (VIEW)

Exemples. La majorité des vues proposées dans les simulations multi-agents pour détecter l'émergence d'un phénomène collectif se limitent bien souvent à la visualisation de caractéristiques propres aux agents ou agrégées par familles, par population ou au niveau de l'environnement. Ainsi, les phénomènes émergents ne sont souvent ni réifiés, ni agentifiés, i.e. pas explicitement représentés dans la simulation, et n'ont donc aucune autonomie. Pour remédier à ce problème, les agents impliqués dans un phénomène collectif peuvent être représentés collectivement par un agent « vue » doté d'un comportement autonome, reflétant les agents micro. Cette approche a été utilisée pour agentifier, ou plus simplement et plus souvent réifier des phénomènes émergents dans des simulations d'écoulement de fluides [19], de flocking [11] ou encore des simulations sociales [16].

De façon similaire, les simulations hybrides interactives de flux de trafic [18] donnent à l'utilisateur la possibilité de « zoomer » dans une zone macroscopique et de générer ainsi par désagrégation une représentation microscopique composée de véhicules en interaction. Ces agents « vues » micro sont *autonomes* vis-à-vis de l'agent macro visualisé : leurs comportements sont totalement définis par des modèles comportementaux (accélération, changement de voie) leur étant propres.

Définition. Les agents VUE sont destinés à réifier dans le niveau cible une représentation des agents du niveau source. Ils possèdent donc un état qui doit être calculé pour refléter à chaque instant l'état des agents dont ils sont issus. Ils sont donc *labiles*, mais leur méthode `update` doit être définie comme la *composition* ou la *décomposition* de l'état des agents qu'ils représentent. Dans le cas général, ils ont également la capacité d'exécuter `decide`, du moment que leur état n'est pas affecté par leurs actions (donc sans impact sur l'état des agents dont ils sont une vue). L'échange d'information (le couplage entre niveaux) est donc bien dans ce cas unidirectionnel, de ou des agents visualisés (niveau source) vers le ou les agents VUES (niveau cible). Les capacités des agents visualisés ne sont pas modifiées. On peut donc parler d'une forme de *délégation d'état* (fig. 7)



(a) Agrégation (b) Désagrégation

FIGURE 7 – Pattern VUE

Algorithmes. Nous ne traiterons pas ici des nombreux algorithmes proposés pour détecter des phénomènes émergents (pour une présentation exhaustive, voir [12]). Les algorithmes mis en œuvre pour le pattern VUE, que ce soit pour l'agrégation (algo. 6) ou la désagrégation (algo. 7), supposent principalement, pour assurer la « délégation d'état », de spécialiser les agents du niveau cible (fig. 8) pour redéfinir `update` comme la composition ou la décomposition des états des agents du niveau source.

Algorithme 6 : Agrégation dans le pattern VUE

créer l'agent macro M
M.observed $\leftarrow \{\mu_1, ..., \mu_n\}$
redéfinir
M.update()=$\{s_t(M) \leftarrow compose(\{s_t(\mu_1), ..., s_t(\mu_n)\})\}$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)

Algorithme 7 : Désagrégation dans le pattern VUE

créer n agents micro $\mu_1, ..., \mu_n$
$\{s_t(\mu_1), ..., s_t(\mu_n)\} \leftarrow decompose(s_t(M))$
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  μ_i.observed $\leftarrow \{M\}$
  redéfinir μ_i.update()=$\{s_t(\mu_i) \leftarrow decompose(s_t(M))[i]\}$
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end

Avantages et limitations. Le pattern VUE permet de fournir à l'observateur des représentations de phénomènes collectifs pertinentes et qui, contrairement aux vues classiques, prennent à part la simulation. Par ailleurs, aux premiers travaux sur l'agentification de ces phénomènes, fortement liés au domaine applicatif, succèdent peu à peu des boîtes à outils génériques, comme SIMANALYZER [3], permettant un champ d'utilisation plus étendu.

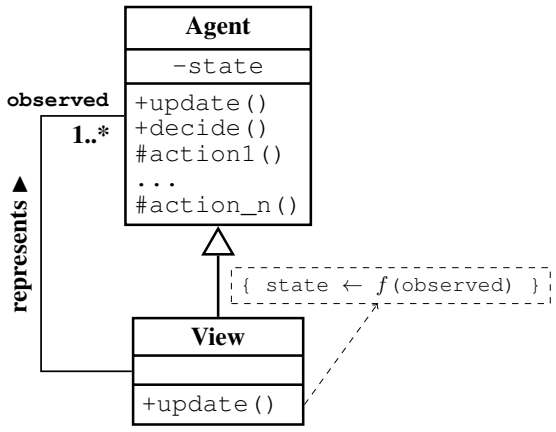


FIGURE 8 – Modélisation d'un agent VUE (*View*) dont l'état est calculé à partir des agents qu'il représente. La fonction f à utiliser est soit *compose* (agrégation), soit *decompose* (désagrégation).

Notons par ailleurs que la majorité des approches implémentant le pattern VUE pour l'agrégation ne font que réifier les phénomènes détectés sans les agentifier pleinement : en d'autres termes, ces objets-vues disposent d'un `update` mais pas de `decide` et ne sont donc pas considérés comme actifs par l'ordonnanceur. La réalisation du pattern est évidemment d'autant plus simple qu'il n'y a pas de comportements spécifiques à définir pour l'agent *Vue*.

Dans le cas contraire, le fait que le couplage entre niveaux soit unidirectionnel peut poser des problèmes de cohérence de la visualisation. Pallier ce problème nécessite de prendre en compte la bidirectionnalité de l'échange d'information et donc de mettre en œuvre le pattern COHABITATION, présenté dans la section suivante, plus compliqué à implémenter et plus coûteux computationellement.

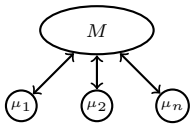
Dans le cas de la désagrégation, il est nécessaire d'introduire dans les agents *Vues*, comme dans le pattern ZOOM, de l'information additionnelle. C'est cependant ici moins problématique car cela n'affecte pas la qualité des résultats des simulations, mais seulement le réalisme de leur visualisation.

3.4 COHABITATION

Exemples. Dans les simulations du développement de larves de diptères [13], les agrégats d'agents micro détectés sont agentifiés dynamiquement afin de pouvoir prédire plus précisément la température générée par les interactions entre insectes et donc simuler précisément leur développement. Les agents macro ainsi créés agissent de façon autonome par rapport aux agents micro, ils peuvent se diviser, fusionner ou disparaître. Ils rétroagissent sur les agents micro en spécifiant la température ressentie par ceux-ci en fonction de la taille et de la densité de l'agrégat.

De même, dans les simulations hybrides de flux de trafic précédemment décrites (cf. section 3.1), la désagrégation est réalisée en générant des agents micro en fonction des caractéristiques de l'agent macro en amont et de la zone micro en aval. Il y a dans ce cas, contrairement au cas de l'agrégation, un échange d'information bidirectionnel entre niveaux micro et macro. En effet, l'agent macro générant des agents micro, doit prendre en compte l'état du trafic en aval. Par exemple, une congestion entraîne une réduction de la vitesse initiale des véhicules générés. De même, si le taux d'occupation de la route est maximal, les véhicules ne pourront être générés, modifiant l'état (vitesse, flux, densité) de l'agent macro.

Définition. Le pattern COHABITATION représente le cas le plus complexe de simulation multi-niveaux. En effet, les agents des niveaux source et cible peuvent exécuter `update` et `decide`. Les interactions entre les niveaux micro et macro sont bidirectionnelles, en ce sens que ces derniers s'influencent mutuellement (fig. 9). Les simulations de systèmes très complexes, comme les systèmes biologiques y ont ainsi particulièrement recours. Il est notamment utilisé dans les nombreux travaux sur la modélisation multi-niveaux du développement de tumeurs [20].



$\longleftrightarrow$ échange d'information bidirectionnel

FIGURE 9 – Pattern COHABITATION

Algorithmes. Comme pour la VUE, des algorithmes de détection de phénomènes émergents sont nécessaires, mais ils sont généralement spécifiques au domaine d'application. Les opérations à réaliser pour l'agrégation (algo. 8) ou la désagrégation (algo. 9) sont quant à elles plutôt simples puisqu'elles consistent simplement à ajouter les agents du niveau cible à la simulation. Tous les agents sont considérés comme labiles et actifs. La principale difficulté consiste

donc à définir de façon appropriée les comportements spécifiques au domaine dont les agents de chaque niveau doivent être dotés.

Algorithme 8 : Agrégation dans le pattern COHABITATION

créer l'agent macro M
$s_t(M) \leftarrow compose(\{s_t(\mu_1), ..., s_t(\mu_n)\})$
`population`.add(M)
`labiles`.add(M)
`actives`.add(M)

Algorithme 9 : Désagrégation dans le pattern COHABITATION

créer n agents micro $\mu_1, ..., \mu_n$
$\{s_t(\mu_1), ..., s_t(\mu_n)\} \leftarrow decompose(s_t(M))$
for $\mu_i \in \{\mu_1, ..., \mu_n\}$ **do**
  `population`.add(μ_i)
  `labiles`.add(μ_i)
  `actives`.add(μ_i)
end

Avantages et limitations. Le pattern COHABITATION permet d'intégrer dans une simulation des agents appartenant à des niveaux d'observation ou d'organisation différents en les traitant d'une façon homogène : que les agents soient « atomiques », ou issus d'un processus d'agrégation ou de désagrégation, ils sont tous gérés par l'ordonnanceur de la même façon. Leur état évolue selon une dynamique qui leur est propre, et ils peuvent être dotés de comportements arbitrairement complexes. La modélisation peut donc être aussi fidèle que nécessaire au système à représenter.

En revanche, outre les questions classiques de détection d'émergence, il est nécessaire de doter tous les agents de comportements appropriés et cohérents. Les interactions qui peuvent avoir lieu entre les agents d'un couple micro/macro d'une part, mais aussi entre les agents de chacun de ces niveaux et d'autres agents, sont plus difficiles à concevoir, ne serait-ce qu'en raison de la combinatoire qu'elle induit. Ce pattern doit donc être utilisé lorsque les objectifs de la simulation justifient pleinement le maintien de deux niveaux totalement autonomes.

Par ailleurs, le coût computationnel associé à la conservation d'agents micro pleinement actifs peut être élevé. On évitera donc de faire appel à ce pattern dans le cas d'une agrégation produite par un nombre important d'agents micro, ou inversement pour une désagrégation susceptible de donner naissance à de nombreux agents micro.

Pattern	Niveau source	Niveau cible
ZOOM	*détruit*	`update` `decide`
MARIO.	`update`	`update` `decide` étendu
VUE	`update` `decide`	`update` simplifié `decide`
COHAB.	`update` `decide`	`update` `decide`

TABLE 1 – Capacité d'action ou d'évolution des agents des niveaux source et cible dans chaque pattern

4 Conclusion et perspectives

Dans les sections précédentes, nous avons montré que les relations entre agents dans un système multi-niveaux peuvent être caractérisées par la combinaison de patterns fondés sur les couplages entre un niveau « de départ » et un niveau « d'arrivée », dans le sens micro vers macro (dit « agrégation ») aussi bien que dans le sens macro vers micro (dit « désagrégation »).

Ces patterns correspondent à des situations minimales abstraites indépendantes d'un domaine particulier, ils peuvent donc s'appliquer de façon générique. Ils sont caractérisés par les capacités des agents du niveau source et du niveau cible à exécuter ou non `update` et `decide` après transformation (tab. 1).

Nous sommes par ailleurs bien conscients que la difficulté majeure qui se pose lors de la conception de modèles multi-niveaux est la *spécification fonctionnelle*, au sens de l'ensemble des comportements propres au domaine visé, en particulier la nature exacte des couplages qui s'opèrent entre chaque niveau. Décider s'il est plus pertinent, lors de la formation d'un embouteillage, de faire disparaître les véhicules entrants ou les faire coexister avec l'agent macroscopique correspondant, cela est en partie de la responsabilité de l'expert du domaine, selon la nature des comportements à modéliser. Toutefois l'informaticien peut exercer un rôle de conseil à ce stade, par exemple lorsque des contraintes de performance sont attendues, plaidant en faveur du pattern MARIONNETTISTE ou du pattern VUE, ou au contraire si la difficulté à expliciter les couplages suggère plutôt de maintenir une COHABITATION entre agents du niveau micro et du niveau macro.

Le travail de classification et de caractérisation de patterns que nous avons mené ici appelle par ailleurs des *développements méthodologiques*.

Eu égard au caractère empirique de notre démarche, il est sans doute prématuré de proposer une méthodologie de mise en œuvre clefs en main pour les patterns que nous avons décrits ci-dessus. Néanmoins, nous pouvons indiquer deux pistes principales. D'abord, le concepteur peut avoir une idée assez claire des *relations* qui doivent exister entre niveaux lors d'une agrégation ou d'une désagrégation : destruction des agents du niveau de départ, asservissement au niveau d'arrivée, etc. Dans ce cas il suffit de suivre l'arbre de décision que nous avons défini pour aboutir automatiquement au pattern approprié. En revanche, il nous semble que dans la majorité des cas, l'identification du pattern le plus pertinent pour une situation donnée ne pourra venir que des *avantages et limitations* de ce pattern, selon les contraintes qui s'exercent sur les agents à modéliser : efficacité computationnelle, facilité à étendre le modèle, compatibilité avec d'autres patterns déjà définis entre l'un des niveaux et un autre groupe d'agents, etc.

Il est possible que certaines situations ne rentrent pas totalement dans notre grille d'analyse, ne serait-ce que parce que des hypothèses simplificatrices ont été introduites dans les modèles ou des raccourcis mis en œuvre lors de l'implémentation. Parmi les quatre patterns que nous avons décrits, le plus riche d'un point de vue comportemental est évidemment le pattern COHABITATION, qui a d'ores et déjà donné lieu à des modèles formels généraux [14, 15] et à l'identification de patterns comportementaux [9]. Nos travaux à venir viseront donc notamment à étendre le cadre méthodologique correspondant et à systématiser l'identification de ces situations complexes.

Références

[1] N. Bouha, G. Morvan, H. Abouaïssa, and Y. Kubera. A first step towards dynamic hybrid traffic modeling. In *29th European Conf. on Modelling and Simulation (ECMS'15)*, pages 64–70, 2015.

[2] E. Bourrel. *Modélisation dynamique de l'écoulement du trafic routier : du macroscopique au microscopique*. Thèse de doctorat, INSA Lyon, 2003.

[3] P. Caillou and J. Gil-Quijano. Description automatique de dynamiques de groupes dans des simulations à base d'agents. *RIA*, 27(6) :739–764, 2014.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[5] J. Gil-Quijano, G. Hutzler, and T. Louail. Accroche-toi au niveau, j'enlève l'échelle. Éléments d'analyse des aspects multiniveaux dans la simulation à base d'agents. *RIA*, 24(5) :625–648, 2010.

[6] Y. Kubera, P. Mathieu, and S. Picault. Everything can be agent ! In *9th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'10)*, pages 1547–1548. IFAAMAS, 2010.

[7] P. Mathieu and S. Picault. The Galaxian project : A 3D interaction-based animation engine. In *Advances on Practical Applications of Agents and Multi-Agent Systems*, pages 312–315. Springer, 2013.

[8] P. Mathieu and Y. Secq. Environment updating and agent scheduling policies in agent-based simulators. In *4th Int. Conf. on Agents and Artificial Intelligence (ICAART)*, pages 170–175, 2012.

[9] A. Maudet, G. Touya, C. Duchêne, and S. Picault. Patterns multi-niveaux pour les SMA. In *23e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2015)*, pages 19–28. Cépaduès, 2015.

[10] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The SWARM simulation system : A toolkit for building multi-agent simulations. Technical Report 96-06-042, Santa Fe Institute, 1996.

[11] T. Moncion. *Modélisation de la complexité et de la dynamique des simulations multi-agents : application pour l'analyse des phénomènes émergents*. Thèse de doctorat, Université d'Évry-Val d'Essonne, 2008.

[12] G. Morvan. Multi-level agent-based modeling - a literature survey. *CoRR*, abs/1205.0561, 2013.

[13] G. Morvan, D. Jolly, A. Veremme, D. Dupont, and D. Charabidze. Vers une méthode de modélisation multi-niveaux. In *7e Conf. de Modélisation et Simulation MOSIM*, volume 1, pages 167–174, 2008.

[14] G. Morvan, A. Veremme, and D. Dupont. IRM4MLS : the influence reaction model for multi-level simulation. In *Multi-Agent-Based Simulation XI*, volume 6532 of *LNCS*, pages 16–27. Springer, 2011.

[15] S. Picault and P. Mathieu. An interaction-oriented model for multi-scale simulation. In *22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pages 332–337. AAAI, 2011.

[16] G. Prévost and C. Bertelle. Detection and reification of emerging dynamical ecosystems from interaction networks. In *Complex Systems and Self-organization Modelling*, volume 39 of *Understanding Complex Systems*, pages 139–161. Springer, 2009.

[17] D. Servat. *Modélisation de dynamiques de flux par agents. Application aux processus de ruissellement, infiltration et érosion*. Thèse de doctorat, Université Paris VI, 2000.

[18] J. Sewall, D. Wilkie, and M.C. Lin. Interactive hybrid simulation of large-scale traffic. *ACM Trans. on Graphics (SIGGRAPH Asia)*, 30(6), 2011.

[19] P. Tranouez. *Contribution à la modélisation et à la prise en compte informatique de niveaux de descriptions multiples. Application aux écosystèmes aquatiques*. Thèse de doctorat, Université du Havre, 2005.

[20] L. Zhang, Z. Wang, J.A. Sagotsky, and T.S. Deisboeck. Multiscale agent-based cancer modeling. *J. Math. Biol.*, 48(4–5) :545–559, 2009.