Design Patterns pour les environnements dans les simulations multi-agents

Philippe Mathieu, Sébastien Picault, Yann Secq

Design Patterns pour les environnements dans les simulations multi-agents

Philippe Mathieu, Sébastien Picault, Yann Secq

Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL (équipe SMAC)
Centre de Recherche en Informatique Signal et Automatique de Lille
F-59000 Lille, France
prenom.nom@univ-lille1.fr

RÉSUMÉ. *L'environnement, considéré généralement comme un des concepts clefs des SMA, tout particulièrement en simulation, fait pourtant rarement l'objet d'une spécification précise ou même explicite, car son implémentation est considérée comme évidente ou donnée. Nous défendons au contraire l'idée que la façon dont on modélise l'espace ou les relations entre agents dans une simulation, conduit à la mise en œuvre d'un nombre réduit de solutions efficaces. Notre démarche vise à formaliser les fonctions fondamentales de l'environnement : permettre aux agents de localiser leurs voisins, et leur fournir de l'information. Ainsi, les compromis entre choix de modélisation (topologie de l'environnement, nature des informations) d'une part, et priorités opérationnelles (efficacité d'exécution, pertinence de la représentation des connaissances) d'autre part, permettent d'identifier quatre grand* patterns *d'environnements. Dans cette approche unificatrice, « l'environnement » habituellement monolithique et parfois complexe d'une simulation multi-agent peut être modélisé et implémenté comme la combinaison de plusieurs* patterns.

ABSTRACT. *Environment, usually regarded as one of the key concepts of MAS especially in simulation, is however rarely specified in a precise or even explicit way, since its implementation is assumed obvious or given. On the contrary, we argue that the way of modeling space and connections between agents in a simulation, allows only a few efficient implementation solutions. We aim at formalizing the fundamental purposes of the environment, i.e. helping the agents to find their neighbors, and providing them with information. Thus, the search for a balance between modeling issues on the one hand (environment topology, nature of the information) and the operational priorities on the other hand (execution efficiency, relevance of knowledge representation), outlines four environment patterns. Through this unifying approach, the usual, monolithical and sometimes complex, "environment" of a multiagent simulation can be modeled and implemented as the combination of severals patterns.*

MOTS-CLÉS : *simulation multi-agent, environnement, parcimonie, ingénierie, patron de conception.*

KEYWORDS: *multiagent-based simulation, environments, parsimony, engineering, design patterns.*

---

1. Introduction

Il est important pour les SMA, en tant que discipline scientifique, de se reconnaître autour d'un noyau minimal de concepts théoriques et de modèles opérationnels, de structures et d'algorithmes qui fassent consensus. Ce « noyau » se doit d'être suffisamment réduit pour former un « dénominateur commun » applicable autant que possible à la diversité des situations existantes, tout en offrant un cadre opérationnel qui permette de passer à la réalisation effective des systèmes. Ce modèle minimal de SMA a vocation à être enrichi pour chaque application particulière et nous ne prétendons nullement qu'il puisse être élaboré d'un simple trait de plume ; néanmoins l'expérience accumulée depuis 25 ans doit nous permettre de l'esquisser.

Nous souhaitons ici faire porter notre réflexion sur la définition minimale de la notion d'environnement, et sur ses implications en termes d'implémentation. On s'accorde en général sur le fait que les agents agissent « dans un environnement » (Ferber, 1995 ; Russell, Norvig, 1995 ; Macal, North, 2010), tant il est sous-entendu que chacun *sait bien* ce qu'est un environnement (Okuyama *et al*., 2005) : au mieux, si différentes sortes d'environnements sont décrites, rien n'est dit sur leur implémentation.

Cet appel implicite au sens commun fait de l'environnement le parent pauvre des SMA, à quelques exceptions près. En particulier, le problème est crucial en simulation (cf. les essais de formalisation de Helleboogh *et al*. (2007) et Weyns et Holvoet (2007)), de sorte qu'il n'a été abordé de façon approfondie quasiment que dans les workshops E4MAS (*Environments for Multiagent Systems*). Aussi, le terme lui-même recouvre des réalités très hétérogènes (Weyns *et al*., 2005), à la fois dans leur structure et dans leurs usages, qui vont d'une description abstraite du « milieu », au sens écologique, où les agents « vivent » (Odell *et al*., 2003), au contexte matériel et logiciel d'exécution de la plateforme, voire à une réalité « extérieure » à la plateforme avec comme problématique centrale l'interopérabilité (Ricci *et al*., 2007 ; Behrens *et al*., 2011).

Ces difficultés à définir d'une façon claire l'environnement, viennent à notre sens d'une focalisation excessive sur les agents eux-mêmes, comme nous l'avons montré à propos des comportements (Kubera *et al*., 2011). Ainsi, bien que les critères de caractérisation des environnements, donnés par Russell et Norvig (1995, ch. 2), restent fréquemment cités (accessible ou non, déterministe ou non, statique vs. dynamique, continu vs. discret), une lecture attentive montre qu'il ne s'agit en fait pas tant de caractéristiques propres à l'environnement, que des capacités de perception, de cognition et d'action des agents qui s'y trouvent. Ainsi, par exemple, l'accessibilité est en tout premier lieu une affaire de fiabilité des perceptions, voire de perception globale opposée à l'obligation d'une perception locale. De même, on glisse de l'environnement « discret » à l'environnement *fini* alors qu'il n'y a pas de lien nécessaire entre les deux (cf. *InfiniteForest* de Payet *et al*. (2009)).

On mesure bien le caractère obsolète de ces classifications, où prédomine un point de vue centré agent, en les comparant aux enjeux énoncés par Weyns *et al*. (2005) où s'esquissent des questions relatives aux liens entre agents et environnements, au be-

soin d'une taxonomie des environnements, et à la primauté des relations topologiques. En effet, pour agir, l'agent a besoin de savoir sur quels autres agents il peut exercer ses actions (communication comprise). L'agent étant en règle générale décrit comme une entité à perception limitée, il ne peut (doit) interagir qu'avec ses *voisins*. **La question essentielle est donc : comment calculer les voisins d'un agent ? L'environnement *doit* être en premier lieu une réponse à cette question.**

Notre objectif ici est triple : 1° chercher une définition minimale de ce que peut bien « signifier » l'environnement lorsque l'on cherche à modéliser un phénomène ou résoudre un problème au moyen d'un système multi-agent ; 2° en donner une caractérisation formelle ; 3° montrer dans quelle mesure la diversité des environnements rencontrés dans la plupart des SMA s'unifie en quelques *patterns* dont le choix ne dépend que d'un nombre réduit de critères de modélisation et d'implémentation, un peu à la façon des *Design Patterns* en génie logiciel (Gamma *et al*., 1994).

Dans ce qui suit, nous poserons comme hypothèse que « l'environnement » que nous cherchons à caractériser est **construit** comme partie du SMA, ce qui est typiquement le cas en *simulation* ou en *résolution distribuée de problèmes* (Simonin, Gechter, 2005). Le cas d'environnements issus de l'existence d'un espace « naturel » (physique, biologique ou socio-culturel), par exemple en robotique mobile, nécessiterait un traitement spécifique que nous n'aborderons pas ici.

Par ailleurs, nous poserons comme hypothèse que **toute entité peut être considérée comme un agent**, dont l'activité est caractérisée selon trois critères indépendants : *l'activité* (faculté d'effectuer des actions), la *passivité* (faculté de subir des actions) et la *labilité* (faculté de changer spontanément d'état). Nous avons en effet montré (Kubera *et al*., 2010) que cette démarche unificatrice n'entraîne pas de perte d'efficacité algorithmique. Bien entendu, nous ne poussons pas le réductionnisme au point de vouloir *tout* agentifier : selon le niveau d'analyse nécessaire à la modélisation, on est amené à choisir *un niveau de granularité en deçà duquel les agents sont remplacés par une information agrégée*. Par exemple, la température en un point est une mesure de l'agitation moléculaire (donc calculable à partir des vitesses de déplacement d'agents molécules) ; mais aux échelles ordinaires il est plus pertinent de considérer que c'est une information agrégée sur un certain volume spatial.

Dans un effort similaire pour circonscrire une base commune aux approches SMA, nous avons défendu l'idée que chaque agent, afin de garantir son autonomie, doit n'offrir qu'un seul point d'accès public qui est l'appel à prendre une décision. Cette invocation est réalisée par un ou plusieurs ordonnanceurs (selon que le système est distribué ou non), qui donnent la parole aux agents, de façon synchrone ou séquentielle. Ceux-ci doivent alors décider d'effectuer certaines actions (ou aucune) en fonction du contexte (Mathieu, Secq, 2012). Nous adoptons également ce point de vue dans le reste de l'article.

Cet article est organisé comme suit. Après avoir décrit formellement une notion minimale d'environnement, à l'aune de ses fonctionnalités principales que sont placer les agents et fournir de l'information, nous décrivons quatre patterns correspondant

chacun à un ensemble de contraintes et à leur implémentation. Nous expliquons ensuite comment les combiner pour traiter les situations usuelles rencontrées dans une simulation multi-agent, en illustrant notre propos par un exemple concret ainsi que par l'implémentation de ces patterns dans une plateforme reconnue. Nous terminons par une comparaison entre notre approche et les points de vue classiques, en défendant l'intérêt d'une unification de concepts ordinairement disjoints.

2. Formalisation de la notion d'environnement

Notre objectif étant de rendre compte d'une façon homogène de la plupart des situations rencontrées habituellement, nous cherchons une base minimale pour définir un environnement dans le cadre des SMA. Plus largement, cela impose d'identifier les éléments formels nécessaires à la caractérisation même d'un SMA : d'abord dans sa structure, et ensuite seulement dans son fonctionnement.

Nous défendons l'idée que les deux principales fonctions dévolues à l'environnement dans un SMA sont :

1. **placer les agents** les uns par rapport aux autres, selon des relations de voisinage et d'accessibilité particulières ;

2. **porter des informations** correspondant à des granularités n'ayant pas été agentifiées.

Mais pour commencer nous avons besoin du **temps**. Quelle que soit l'opinion métaphysique qu'on défende, en informatique il est exclu d'utiliser un temps continu (au sens de $\mathbb{R}$) : nous considérons donc dans la suite que l'exécution d'un SMA passe par une suite **d'instants** notés $t \in [0, T_{\max}]$, pouvant être séparés par des durées quelconques (et pas nécessairement constantes). Cette représentation est évidemment indépendante du caractère synchrone ou asynchrone, centralisé (par un ordonnanceur commun) ou distribué, du système considéré, et permet de traiter aussi bien des systèmes à temps discret qu'à événements discrets.

À chaque instant, le SMA contient un ensemble d'agents, autrement dit des entités qui jouent un rôle dans le modèle, et que nous noterons $\mathcal{A}_t$. À ce stade nous ne faisons (et ne souhaitons faire) aucune hypothèse particulière quant aux capacités d'action, de perception, de cognition de ces entités.

2.1. Première fonction : placer les agents

Pour définir l'environnement, il faut d'abord déterminer quel espace partagent ces agents. La façon la plus simple de le formaliser est de supposer qu'il existe un ensemble E dont les éléments sont des « lieux » susceptibles d'accueillir des agents. A priori, cet ensemble E peut être quelconque. Toutefois, il faut être capable également de déterminer dans quelle mesure ces lieux sont reliés les uns aux autres. Pour cela il faut au minimum définir une application $\mathbf{d} : \mathbf{E} \times \mathbf{E} \to \mathbb{R}^+ \cup \{+\infty\}$ qui satisfait les conditions suivantes :

1. $\forall x, y \in E, d(x, y) = 0$ ssi $x = y$
(séparation)

2. $\forall x, y, z \in E, d(x, z) \leq d(x, y) + d(y, z)$
(inégalité triangulaire)

Une telle fonction est une **quasidistance** (Steen, Seebach, 1995) : elle ne possède pas nécessairement la propriété de symétrie (on peut avoir $d(x, y) \neq d(y, x)$). Néanmoins, (E, d) constitue un *espace quasimétrique* dans lequel la quasidistance définit une **topologie**.

En toute rigueur, la quasidistance peut elle-même dépendre du temps (e.g. un agent construit un mur et modifie ainsi les relations d'accessibilité d'un point à un autre, donc la topologie, au cours de l'exécution). Toutefois nous gardons la notation d (et non d_t) par souci de simplification.

Cette description minimaliste généralise Ferber (1995, p. 14), où l'environnement est présenté comme « un espace disposant généralement d'une métrique », et couvre en fait des situations fort diversifiées :

– l'espace usuel en 3D correspond à $E = \mathbb{R}^3$, avec pour d la distance euclidienne ;

– une grille à deux dimensions, dans laquelle on utilise le voisinage de Moore (chaque cellule est à distance 1 de ses 8 voisines), correspond à $E = \mathbb{Z}^2$, avec pour d la distance de Tchebychef (écart maximum entre les coordonnées des points) ;

– de même un espace continu 2D « torique » de largeur w et de hauteur h correspond mathématiquement à l'espace quotient $\mathbb{R}^2/(w\mathbb{Z} \times h\mathbb{Z})$ (avec par exemple la distance euclidienne) ;

– un réseau d'accointances se modélise par un graphe, i.e. un ensemble E de sommets muni de la distance géodésique (plus petit chemin entre chaque couple de nœuds);

– un environnement de type « soupe » où les agents sont « non localisés » (c'est-à-dire tous au même endroit) se représente au moyen d'un singleton $E = \{p\}$, muni de la distance triviale $\forall x, y\ d(x, y) = 0$.

N.B. : une raison qui d'ordinaire fait que l'on distingue les environnements spatiaux des environnements sociaux, c'est qu'il peut sembler que les premiers sont d'abord liés à une distance, tandis que les seconds donnent la primauté à la topologie. En pratique cela n'a aucune espèce d'importance : comme l'illustrent les exemples ci-dessus, la topologie découle souvent de la quasidistance, mais réciproquement la quasidistance peut se construire à partir des relations topologiques.

Il ne suffit toutefois pas de dire dans quelle sorte de monde les agents peuvent évoluer pour avoir défini un *environnement*. À chaque instant, chaque agent se trouve à un endroit précis de l'espace ; en outre lors de la modélisation on s'intéresse en général seulement à un sous-ensemble $\mathcal{E} \subset (E, d)$. Si l'on admet que l'environnement a bien vocation à localiser les agents, à induire des relations de *voisinage* qui vont permettre à chaque agent d'effectuer des actions en fonction de ses perceptions locales et de son état, il est indispensable d'adjoindre à l'espace où se trouvent les agents, une

fonction qui permet de leur attribuer une **position**. Une telle fonction est de la forme : $\boldsymbol{pos}_t : \mathcal{A}_t \to \mathcal{E}$. L'environnement peut alors être défini ainsi :

> Un environnement est un sous-ensemble $\mathcal{E}$ d'un espace quasimétrique (E, d) muni d'une famille de fonctions $(pos_t)_{t\in[0,T_{\max}]}$ qui donnent les positions successives des agents dans $\mathcal{E}$.

On notera que cette description (ainsi que ce qui suit) s'extrapole aisément aux situations dans lesquelles les agents ont une *extension spatiale*, i.e. occupent une partie de l'espace ($\wp(\mathcal{E})$) et non simplement un point, en définissant une fonction $\overline{pos_t}$: $\mathcal{A}_t \to \wp(\mathcal{E})$.

La raison pour laquelle d'ordinaire nous n'identifions pas la **position** comme étant ce qu'elle est, une *fonction*, c'est que la plupart du temps nous ne sommes pas en mesure de donner cette fonction en *intension* mais seulement en *extension* — et ce, de façon distribuée puisque c'est en général chaque agent a qui connaît la valeur de $pos_t(a)$ sous la forme d'un attribut. Pire : le passage de pos_t à pos_{t+1} est lui-même en général l'effet d'un algorithme propre à chaque agent (l'effet de son comportement), de sorte que la connaissance de pos_t ne permet que très rarement de prévoir ce que sera pos_{t+1}. Il n'en reste pas moins, comme nous le verrons ensuite, que cette façon de décrire l'environnement permet de mettre en évidence les critères qui peuvent présider aux choix d'implémentation – et ce, d'une manière totalement indépendante du domaine visé.

On peut par ailleurs définir la fonction réciproque de la position, le *contenu* associé à un point de l'environnement, comme suit :

$$\begin{array}{rcl} \boldsymbol{cont}_t : \mathcal{E} & \to & \wp(\mathcal{A}_t) \\ p & \mapsto & \{a \in \mathcal{A}_t \mid pos_t(a) = p\} \end{array}$$

À partir de cette formalisation, nous affirmons que ce qui distingue divers types d'environnements et les façons possibles de les implémenter, c'est :

– d'une part *la nature de l'espace quasimétrique* (E, d) (espace continu ou discret, topologie de cet espace, etc.) ;

– d'autre part, le *choix d'expliciter ou non ces fonctions*, en particulier $cont_t$.

En effet, ce qui compte pour l'action des agents, c'est de savoir avec quels autres agents (leurs *voisins*) ils peuvent interagir. Le comportement des agents est donc directement dépendant de la fonction en charge de la *perception des voisins (neighbors)*, soit : $\nu_t : \mathcal{A}_t \to \wp(\mathcal{A}_t)$.

La fonction la plus « simple » (au sens topologique) qu'on puisse définir pour déterminer les voisins d'un agent a est simplement le calcul des agents situés dans une boule centrée sur la position de a et de rayon r (souvent appelé « halo de perception » de a) :

$$\nu_t(a, r) = \{a' \in \mathcal{A}_t \setminus \{a\} \mid d(pos_t(a), pos_t(a')) \leq r\}$$

mais il existe bien sûr une infinité de fonctions de voisinage possibles.

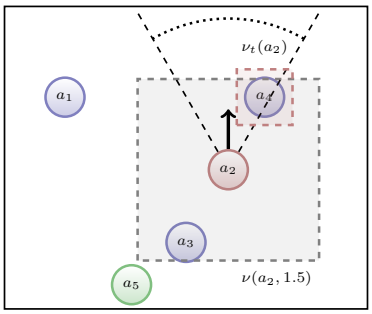


FIGURE 1. *Un exemple de perception anisotrope, pour un agent percevant dans un angle de 60°« devant lui ». La fonction de perception ν_t est donnée par l'intersection entre la boule $\nu(a_2, 1.5)$ (calculée ici avec une distance de Tchebychef) et le secteur angulaire de 60°défini par a_2 et son orientation*

En particulier, la boule topologique correspond évidemment à une perception isotrope. Pour représenter une perception « orientée » (par exemple un agent disposant d'un cap et ne percevant que devant lui), il peut être nécessaire de calculer l'intersection de cette boule et d'un secteur angulaire (cf. figure 1), ou plus généralement d'appliquer aux points situés dans la boule des contraintes représentant les modalités de perception que l'on souhaite modéliser.

En pratique, le calcul de ce voisinage peut s'avérer coûteux et c'est donc un compromis temps/espace entre algorithmique et structure de données qui donne naissance à l'implémentation de cette fonction sous telle ou telle forme, comme nous le décrivons dans la section 3.

2.2. Deuxième fonction : porter de l'information

Au placement des agents et à la détermination de leurs voisins, s'ajoutent également des choix de modélisation de « l'information » dont l'environnement peut être porteur. Comme nous l'avons signalé, selon l'échelle de la modélisation il peut être nécessaire d'approximer les niveaux sous-jacents par une mesure macroscopique.

Prenons l'exemple des phéromones : en toute rigueur, il s'agit de molécules qui sont soumises à des phénomènes physico-chimiques comme le mouvement brownien (qui cause leur diffusion) et l'évaporation ou la dégradation. En tant que telles, elles devraient donc être représentées par des agents. Dans ce premier cas, l'environnement ne « porte » aucune information : ce sont les molécules concernées qui s'en chargent, *en tant qu'agents*, en interagissant avec d'autres sortes d'agents. Mais en pratique, on préfère souvent les modéliser par une double approximation : d'une part on les agrège numériquement (un peu à la façon d'une concentration en chimie) ; d'autre part on les place dans un espace discrétisé (les cases d'une grille ou un graphe (Colorni *et al.*, 1991)). Dans ce second cas, qui est bien souvent utilisé à seules fins de réduire le coût

computationnel d'un trop grand nombre d'entités (et au prix d'une approximation des mécanismes de diffusion par exemple), la littérature SMA n'offre pas de consensus pour nommer cette représentation purement informationnelle.

Or, puisqu'il s'agit au fond de procéder à une approximation comme en physique ou en chimie, il nous semble judicieux de faire appel à la notion de **champ**, déjà utilisée par exemple pour la navigation réactive ou les phéromones (« champs de potentiels » de Latombe (1991), « co-fields » de Mamei et Zambonelli (2006), « champs moyens » chez Van Dyke Parunak (2011)), et que nous souhaitons généraliser en considérant que toute « information » (force de gravitation, température, obstacle...) présente dans l'environnement est une fonction de champ. Le champ est alors une fonction définie sur $\mathcal{E}$, à valeurs dans un ensemble d'informations $\mathcal{I}$. Ainsi par exemple, la concentration de phéromones dans chaque case d'une grille est donc une fonction ***phero:*** $\mathbb{Z}^{\mathbf{2}} \rightarrow \mathbb{R}^{+}$. Elle est définie *en extension* et change à chaque pas de temps au moyen d'un algorithme de diffusion-évaporation (tel que décrit par Michel (2013)).

Un champ peut être parfois, bien que plus rarement, défini en intension : par exemple dans *InfiniteForest* de Payet *et al*. (2009) où, au moyen d'une technique procédurale, il est possible de se placer en n'importe quel point de $\mathbb{Z}^2$ et d'y trouver un paysage constant dans le temps mais différent de tous les autres, avec un coût computationnel et un coût mémoire extrêmement réduits.

Là encore, les choix de représentation explicite ou non de ces fonctions de champs entraînent des contraintes sur l'implémentation de l'environnement opérationnel.

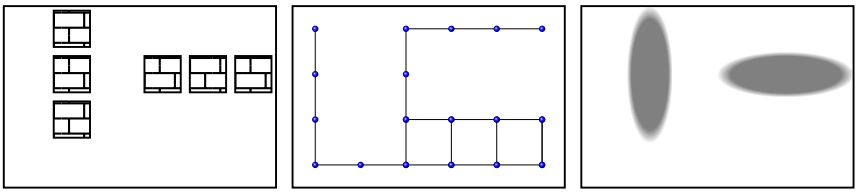
En tout état de cause il n'existe que très rarement une solution univoque pour un problème donné. Prenons le cas de la modélisation *d'obstacles* dans un environnement. Trois possibilités s'offrent au concepteur :

1. modéliser chaque obstacle comme un agent (cf. figure 2a) ;
2. modéliser l'accessibilité d'un point à un autre de l'environnement par la fonction de distance elle-même (cf. figure 2b) ;
3. modéliser l'accessibilité à un point de l'environnement par une fonction de champ (cf. figure 2c).

Dans le premier cas (un obstacle est un agent), on peut évidemment associer des comportements aux « obstacles » (par exemple une porte à ouverture sur badge, un mur qui peut être cassé par un autre agent, etc.). En revanche il faut disposer d'algorithmes de simulation efficaces pour éviter que de tels agents peu actifs ne fassent exploser le temps nécessaire pour un tour de parole (Kubera *et al*., 2010).

Le deuxième cas (l'obstacle modifie la topologie, i.e. la distance) présente l'avantage de ne pas avoir à matérialiser les obstacles en tant que tels, mais suppose souvent de donner la fonction de distance en extension.

Enfin, le dernier cas (l'obstacle comme information), là encore les obstacles ne sont pas matérialisés et la fonction de champ qui les représente n'est pas nécessai-



(a) Agents (b) Topologie (c) Champ

FIGURE 2. *Diverses possibilités de modélisation des obstacles. Ceux-ci peuvent être représentés soit par des agents (2a), soit par la fonction de distance i.e. la topologie de l'espace, qui détermine les relations d'accesibilité entre les points (2b), soit par une fonction de champ (2c)*

rement booléenne (par exemple elle peut donner une probabilité de passer, un coût associé au déplacement, etc.). En revanche, il peut être également nécessaire de donner cette fonction de champ en extension.

Selon la modélisation choisie pour une telle question, les contraintes sous-jacentes (présence d'agents, fonctions données en intension ou en extension, environnement continu ou discret) vont peser dans le choix du *pattern* d'environnement le plus approprié.

Les contraintes liées à la recherche de voisins sont souvent prépondérantes par rapport à celles induites par la représentation de l'information. Néanmoins, le cas inverse peut se produire lorsque les agents n'interagissent jamais directement entre eux mais seulement par stigmergie, *via* l'environnement. Dans ce cas la question de la recherche des voisins perd tout intérêt au profit de la seule question de la gestion des champs par l'environnement. De même, dans l'approche IRM (Ferber, Müller, 1996) les influences émises par chaque agents peuvent être représentées par autant de fonctions de champ, leur résultante (la réaction) étant elle aussi une fonction de champ.

3. Quatre *patterns* d'environnements fondamentaux

Il ne fait aucun doute qu'on rencontre *de facto* une très grande variété de structures et d'algorithmes permettant d'implémenter l'environnement (voire les environnements) d'un SMA. Notre propos ici n'est ni de prétendre à une approche normative, ni de viser à une énumération exhaustive de tous les cas, mais bien plutôt de montrer que, selon la nature de l'espace métrique (E, d) qui sous-tend un environnement, et selon le choix d'implémenter ou non la fonction réciproque de la position, on peut identifier un nombre limité de grandes familles d'implémentations, correspondant chacune à des usages récurrents, et dans lesquelles s'inscrivent la plupart des situations usuelles.

Dans ce qui suit, nous présentons les quatre *patterns* fondamentaux que nous avons identifiés, en expliquant d'abord brièvement leur principe, leurs avantages et inconvénients. Nous montrons ensuite comment sont réalisées deux opérations majeures : la *recherche de voisins* et le *déplacement* d'un agent, autrement dit les algorithmes (notés respectivement **`neighbors(a,r)`** et **`move_to(a, p)`**) qui construisent les valeurs des fonctions ν_t et pos_{t+1} pour chaque agent. Enfin nous indiquons les situations dans lesquelles s'applique chaque *pattern*.

Au lieu de chercher à construire un environnement unique relativement complexe qui empêche la séparation des préoccupations [1], il est préférable de *combiner plusieurs de ces patterns d'environnements fondamentaux*. Ainsi par exemple, si des robots simulés doivent cartographier collectivement un bâtiment (Bautin *et al.*, 2013), ils sont à la fois situés dans un environnement continu dans lequel se trouvent des obstacles (autres entités), et dans un environnement social (réseau d'accointances) pour échanger leurs informations. Dans de très rares cas à l'inverse, l'« environnement » peut devenir implicite, en n'étant plus déduit que des relations d'accointances entre agents (§ 3.4).

3.1. Le pattern **`AgentSet`**

3.1.1. Principe

L'implémentation la plus immédiate et la plus générale (applicable à un environnement quelconque) consiste à doter chaque agent d'un attribut **`pos`** représentant $pos_t(a)$ (position de l'agent a à l'instant t), l'environnement se réduisant à une collection d'agents qui représente $\mathcal{A}_t$ et à la fonction d (figure 3).

3.1.2. Avantages et inconvénients

L'avantage de cette méthode est sa simplicité d'écriture ; son inconvénient principal, une complexité de la perception des voisins à chaque tour de parole en $\mathcal{O}(n^2)$, n étant le nombre d'agents dans le système. Pour calculer sa fonction **`neighbors`**, chaque agent doit en effet calculer la distance aux $(n-1)$ autres agents (cf. algorithme 1). En revanche le déplacement d'un agent n'affecte que lui-même puisqu'il consiste simplement à modifier l'attribut **`pos`** de l'agent concerné (algorithme 2).

Dans cette implémentation, seule la fonction pos_t est explicitement réifiée. En pratique, l'environnement opérationnel se réduit à un ensemble d'agents.

3.1.3. Usages

En principe, le *pattern* `AgentSet` peut être employé pour réaliser n'importe quelle simulation multi-agents, dans la mesure où il constitue une implémentation *a minima* de la notion d'environnement. Cependant, dans la pratique, il nous semble plus particulièrement approprié pour des environnements relativement quelconques dans les-

1. au sens de la *separation of concerns* en génie logiciel

Algorithme 1 : Calcul des voisins d'un agent avec le *pattern* `AgentSet`

$\forall a_i \in \mathcal{A}_t$ **`neighbors`**(a_i, r):
N $\leftarrow \emptyset$
for $a_j \in \mathcal{A}_t, a_j \neq a_i$ **do**
  if $d(pos_t(a_i), pos_t(a_j)) \leq r$ **then**
    N $\leftarrow$ **N** $\cup \{a_j\}$
  end
end
return **N**

Algorithme 2 : Déplacement d'un agent avec le *pattern* `AgentSet`

`move_to`(a_i, p) : $pos_{t+1}(a_i) \leftarrow p$.

quels les agents sont *en nombre réduit*, sans quoi le coût du calcul des voisins le rend prohibitif et fait préférer l'un des deux *patterns* ci-après. Il convient également si les champs éventuels sont *donnés en intension*. En revanche, il ne se prête pas bien à la représentation « d'obstacles », que ceux-ci soient représentés par des agents (car il faut procéder par « lancer de rayon »), par la quasidistance ou par des champs (car alors ces fonctions doivent être données en extension).

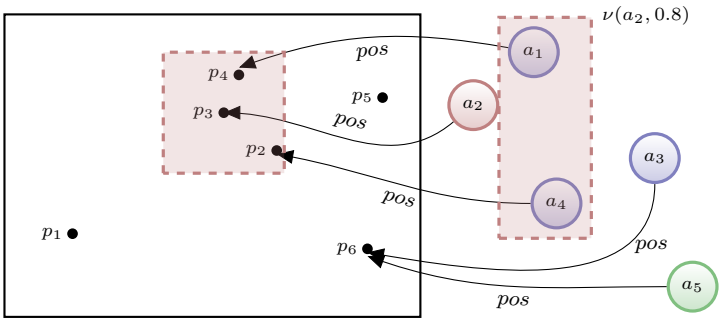


FIGURE 3. *Principe du pattern* `AgentSet`. *L'environnement (espace dessiné à gauche) où sont situés les agents (représentés à droite) n'est pas concrétisé : il se limite à la donnée de la fonction* **pos**. *Le calcul des voisins d'un agent (ici* $\nu(a_2, 0.8)$*, avec une distance de Tchebychef) nécessite un test de distance effectué sur tous les autres agents*

3.2. Le pattern `StandardGrid`

3.2.1. Principe

On rencontre assez fréquemment des environnements basés sur un espace discret organisé sous la forme d'un **réseau** (l'exemple le plus simple étant des coordonnées entières dans un espace de dimension k, $E = \mathbb{Z}^k$). On peut alors voir les positions discrètes des agents comme des « cases » et l'implémentation de tels environnements consiste à ajouter à la solution précédente une structure de type *grille*, i.e. un ensemble de *cellules* qui constituent un pavage régulier de l'environnement (cf. figure 4a). Dans le cas de $\mathbb{Z}^k$, cela se fait aisément à partir d'un tableau à k dimensions contenant des ensembles d'agents. Pour chaque point (ou cellule) p de l'environnement, on dispose ainsi d'un accès direct à tous les agents situés en ce point, autrement dit on explicite par cet attribut la fonction $cont_t$ réciproque de pos_t (cf. figure 4b). On dispose par ailleurs d'un moyen simple d'accéder aux points adjacents à p puisqu'un tel réseau définit en fait un système de coordonnées : le calcul des points voisins de p (`neighborhood` dans l'algorighme 3, ligne 3) est donc immédiat (boucle sur des coordonnées).

3.2.2. Avantages et inconvénients

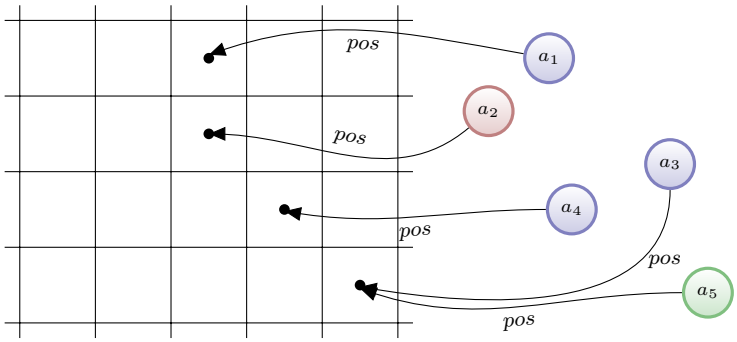
L'accès direct aux agents situés en un point accélère considérablement la recherche des voisins d'un agent a_i, puisque ceux-ci sont nécessairement dans la même cellule que a_i ou dans les cellules adjacentes. Pour calculer les voisins de n agents dans un rayon r (algorithme 3), le calcul est en $\mathcal{O}(n.r^k)$, puisqu'il faut examiner $\mathcal{O}(r^k)$ points de l'environnement seulement (ainsi pour un voisinage de Moore avec $k = 2$ on examine au plus un carré de $(2r + 1)^2$ cellules). En revanche, le déplacement d'un agent nécessite des mises à jour plus compliquées pour maintenir la cohérence entre pos_{t+1} et $cont_{t+1}$ (algorithme 4).

Algorithme 3 : Calcul des voisins d'un agent avec le *pattern* `StandardGrid`

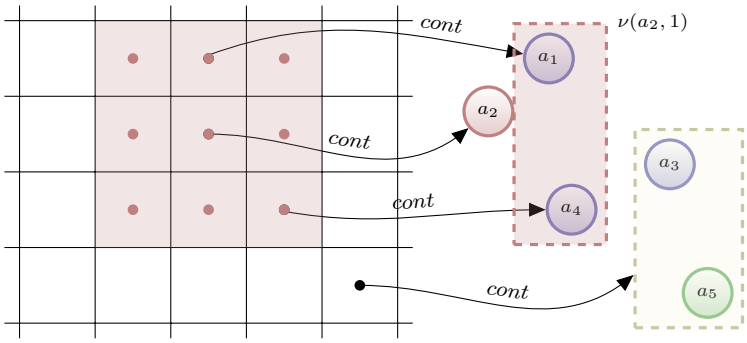
1 $\forall a_i \in \mathcal{A}_t$ **neighbors**(a_i, r):
2 $\mathbf{N} \leftarrow \emptyset$
3 **neighborhood** $\leftarrow \{p \in E \mid d(pos_t(a_i), p) \leq r\}$
4 **for** $c \in$ **neighborhood do**
5 $\quad \mathbf{N} \leftarrow \mathbf{N} \cup cont_t(c)$
6 **end**
7 return $\mathbf{N} \setminus \{a_i\}$

3.2.3. Usages

Cette méthode s'applique assez naturellement dès que l'on doit représenter un environnement à coordonnées entières et que le *pattern* `AgentSet` s'avère trop coûteux. C'est le cas si le rayon moyen $\bar{r}$ dans lequel les agents recherchent leurs voisins



(a) Position des agents



(b) Contenu des « cellules » et calcul des voisins

FIGURE 4. *Principe du pattern* `StandardGrid`*. Tous les agents ont des coordonnées discrètes (4a). Il est donc facile d'écrire (par exemple sous forme d'un tableau) la fonction* $cont$ *réciproque de la position, qui donne pour chaque point (ou case) de l'environnement la liste des agents qui y sont situés (4b), et de s'en servir pour calculer les voisins d'un agent (ici* $\nu(a_2, 1)$*)*

Algorithme 4 : Déplacement d'un agent avec le *pattern* `StandardGrid`

1 **`move_to`**(a_i, p):
2 $cont_{t+1}(pos_t(a_i)) \leftarrow cont_t(pos_t(a_i)) \setminus \{a_i\}$
3 $cont_{t+1}(p) \leftarrow cont_t(p) \cup \{a_i\}$
4 $pos_{t+1}(a_i) \leftarrow p$

vérifie : $\bar{r}^k \ll n$. C'est également une solution particulièrement adaptée à la représentation des champs, notamment lorsque ceux-ci peuvent faire l'objet d'une approximation sur des coordonnées entières (c'est le cas fréquemment des phéromones) : même s'ils sont donnés en extension, il suffit de les stocker dans un tableau à k dimensions. On peut alors envisager en outre une parallélisation du calcul comme proposé par Michel (2013). De même, la gestion d'obstacles est facilitée, que ce soit sous la forme d'agents (en « condamnant » l'accès à leur cellule), de champs ou de la distance (topologie des cellules).

Ce procédé par « grille » ne se limite évidemment pas à un pavage carré ; les simulations en géographie font un usage fréquent des hexagones (Sanders *et al.*, 1997), qui constituent également un réseau de $\mathbb{R}^2$ et où l'on peut donc donner des coordonnées entières à toute cellule hexagonale. Cette méthode s'applique également aux environnements basés sur des graphes (labyrinthes, sites web, ontologies...) : dans ce cas l'espace est un ensemble de places ($\mathcal{E} = \{p_1, ..., p_N\}$) qui peut se représenter par un tableau de N ensembles d'agents, et l'on peut représenter les distances entre ces points par une matrice des distances (δ_{ij}) où $\forall i, j\ \delta_{ij} = d(p_i, p_j)$. L'existence de cette matrice donne un accès direct aux places situées dans un rayon r autour de la place $pos_t(a_i)$.

Enfin, notons que ce *pattern* se simplifie dans un cas fréquent : si deux agents ne peuvent occuper la même position (fonction pos_t injective), alors il suffit de disposer d'un tableau d'agents au lieu d'un tableau d'ensembles d'agents.

3.3. Le pattern `AggregateGrid`

3.3.1. Principe

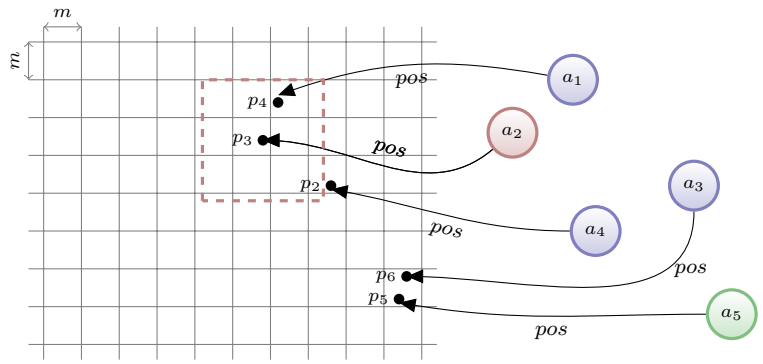
Le *pattern* précédent ne peut s'appliquer directement lorsque l'environnement est continu (cf. figure 5a). Néanmoins, à défaut de réifier la fonction $cont_t$, on peut en construire une approximation discrète en « projetant » les positions continues dans un espace discret $\mathbb{E} \subset E$. Il faut pour cela définir une fonction $\boldsymbol{cell}_m : E \to \mathbb{E}$ qui discrétise E suivant un maillage de pas m (autrement dit il faut : $\forall c \in \mathbb{E}, \exists c' \in \mathbb{E}, c' \neq c$ tel que $d(c, c') \leq m$). Ainsi en prenant par exemple $E = \mathbb{R}^2$, on peut choisir assez naturellement $\mathbb{E} = \mathbb{Z}^2$ et la fonction de discrétisation la plus immédiate s'écrit :

$$\begin{array}{rcl} cell^m : \mathbb{R}^2 & \to & \mathbb{Z}^2 \\ p = (x, y) & \mapsto & (\lfloor \frac{x}{m} \rfloor, \lfloor \frac{y}{m} \rfloor) \end{array}$$

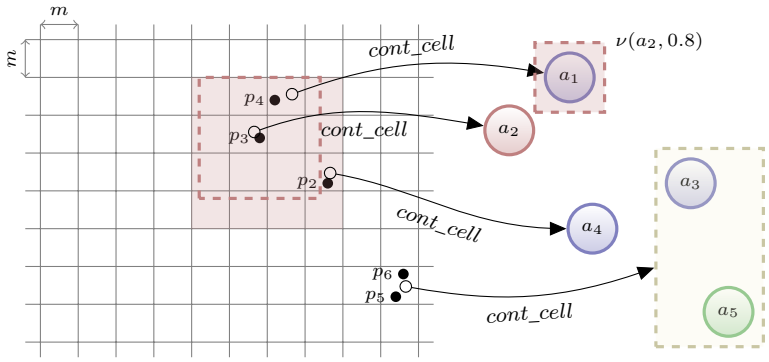
où $\lfloor u \rfloor$ désigne la partie entière par défaut de u. On peut alors définir dans le cas général une forme « agrégée » de $cont_t$ donnant l'ensemble des agents situés dans la « cellule » c :

$$\begin{array}{rcl} cell\_cont_t^m : \mathbb{E} & \to & \wp(\mathcal{A}_t) \\ c & \mapsto & \{a \in \mathcal{A}_t \mid cell^m(pos_t(a)) = c\} \end{array}$$

Il reste à réifier la fonction $cell\_cont_t^m$ comme précédemment, au moyen d'un tableau `cell_cont` à k dimensions contenant des ensembles d'agents (les agents contenus



(a) Position des agents



(b) Contenu des « cellules » et calcul des voisins

FIGURE 5. *Principe du pattern* `AggregateGrid`*. Les agents ont cette fois des coordonnées continues (5a), sur lesquelles on peut construire une discrétisation selon un maillage de taille arbitraire m, ce qui permet une agrégation $cont\_cell$ donnant pour chaque « cellule » de ce maillage, les agents qui y sont situés (5b). La recherche des voisins d'un agent (ici, $\nu(a_2, 0.8)$) repose sur l'exploration des cellules adjacentes (qui donne a_1 et a_4) puis la vérification de la distance (qui élimine a_4)*

dans la cellule $cell_m(p)$: cf. figure 5b). Là encore, le calcul de voisinage entre cellules (algorithme 5, ligne 4) est direct puisqu'on peut travailler sur des coordonnées.

3.3.2. Avantages et inconvénients

Cette discrétisation permet de profiter de l'accélération computationnelle d'une grille, tout en autorisant un maillage arbitraire m de l'espace continu. Le coût de la recherche des voisins de n agents dans un rayon r est en $\mathcal{O}(n.(\lceil \frac{r}{m} \rceil)^k.\bar{q})$ (où $\lceil u \rceil$ est la partie entière par excès de u), $\bar{q}$ désignant la *densité* des agents, i.e. le nombre moyen d'agents dans chaque cellule (cf. algorithme 5). En revanche comme précédemment la mise à jour du contenu des cellules est complexifiée (cf. algorithme 6).

Algorithme 5 : Calcul des voisins d'un agent avec le *pattern* `AggregateGrid`

$\forall a_i \in \mathcal{A}_t$ **neighbors**(a_i, r):
N $\leftarrow \emptyset$
$c_0 \leftarrow cell^m(pos_t(a_i))$
neighborhood $\leftarrow \{p \in \mathbb{E} \mid d(c_0, p) \leq \lceil \frac{r}{m} \rceil\}$
for $c \in$ **neighborhood do**
  for $a_j \in cell\_cont_t^m(c), a_j \neq a_i$ **do**
    if $d(a_i, a_j) \leq r$ **then**
      N $\leftarrow$ **N** $\cup \{a_j\}$
    end
  end
end
return **N**

Algorithme 6 : Déplacement d'un agent avec le *pattern* `AggregateGrid`

move_to(a_i, p):
$cont\_cell_{t+1}(pos_t(a_i)) \leftarrow cont\_cell_t(pos_t(a_i)) \setminus \{a_i\}$
$cont\_cell_{t+1}(p) \leftarrow cont\_cell_t(p) \cup \{a_i\}$
$pos_{t+1}(a_i) \leftarrow p$

3.3.3. Usages

Le *pattern* `AggregateGrid` s'applique dès lors que l'on doit représenter un environnement continu contenant un nombre important d'agents. Un point très intéressant de cette méthode concerne les environnements de taille finie, car si les agents occupent l'espace de façon homogène, il est possible de calculer un maillage qui maintienne le temps de calcul en-dessous d'un seuil fixé. En effet, si l'on considère un environnement « hypercubique » $\mathcal{E} = [0, W]^k$, discrétisé en $C = \lceil \frac{W}{m} \rceil^k$ cellules, la densité est $q = \frac{n}{C}$ et le calcul se fait donc en $\mathcal{O}(n^2.(\frac{a}{b})^k)$, en posant $a = \lceil \frac{r}{m} \rceil$ et $b = \lceil \frac{W}{m} \rceil$. L'encadrement des parties entières donne : $\frac{r}{W+m} \leq \frac{a}{b} \leq \frac{r+m}{W}$. Autrement dit, si l'on est capable d'estimer le rayon de perception moyen des agents ainsi que le nombre d'agents présents en moyenne dans le système, le coût en $\mathcal{O}(n^2)$ peut être contrebalancé par un facteur d'accélération $g = (\frac{b}{a})^k$ qui est plus grand que la valeur $G = (\frac{W}{\bar{r}+m})^k$. On peut donc, en ajustant le maillage, garder par exemple un temps de calcul proportionnel au nombre d'agents en choisissant $G > n$, soit $m \leq m^\star$ avec $m^\star = \frac{W}{\sqrt[k]{n}} - \bar{r}$. Le tableau 1 donne les valeurs $m^\star$ obtenues pour diverses valeurs du nombre d'agents et du rayon de perception moyen, à taille (W), et dimension (k) fixées. Le tableau 2 donne quant à elle le gain relatif $\log_{10} \frac{G}{n}$ obtenu en prenant un maillage « naïf » $m = 1$, ce qui permet d'estimer le coût de la perception.

Bien évidemment ces calculs ne concernent que la perception des voisins, en pratique il doivent être ajustés pour tenir compte des mises à jours nécessaires lorsque les agents se déplacent (i.e. changent de cellule dans l'espace discrétisé) : cela dépend de la vitesse (moyenne) de déplacement des agents, ainsi que de l'efficacité de la suppression et de l'ajout d'un agent dans les structures de données qui représentent le contenu d'une cellule.

TABLEAU 1. *Valeur du maillage $m^\star = \frac{W}{\sqrt[k]{n}} - \bar{r}$ qui garantit un temps de calcul en $\frac{n^2}{g}$, avec $g \geq n$ (n étant le nombre d'agents attendu dans le système), pour diverses valeurs du rayon de perception moyen $\bar{r}$; on a pris ici $W = 10^3$ et $k = 2$. Les valeurs manquantes correspondent à $m^\star \leq 0$, ce qui signifie que la densité des agents entraîne un coût de calcul supplémentaire (on perd la majoration de g) — cf.tableau 2.*

n \ $\bar{r}$	0.5	1	2	5	10	50	100
10^2	99,5	99	98	95	90	50	0
10^3	31,1	30,6	29,6	26,6	21,6		
10^4	9,5	9	8	5			
10^5	2,7	2,2	1,2				
10^6	0,5						

TABLEAU 2. *Gain relatif maximum $\log_{10} \frac{G}{n} = \log_{10} \frac{1}{n} \cdot (\frac{W}{\bar{r}+m})^k$ sur le coût de perception des voisins pour un maillage « naïf » $m = 1$, en fonction du nombre n d'agents attendu dans le système et du rayon de perception moyen $\bar{r}$ (le temps de calcul est majoré par $\frac{n^2}{G}$); on a pris ici $W = 10^3$ et $k = 2$.*

n \ $\bar{r}$	0.5	1	2	5	10	50	100
10^2	3,65	3,4	3,05	2,44	1,92	0,58	-0,01
10^3	2,65	2,4	2,05	1,44	0,92	*-0,42*	*-1,01*
10^4	1,65	1,4	1,05	0,44	*-0,08*	*-1,42*	*-2,01*
10^5	0,65	0,4	0,05	*-0,56*	*-1,08*	*-2,42*	*-3,01*
10^6	*-0,35*	*-0,6*	*-0,95*	*-1,56*	*-2,08*	*-3,42*	*-4,01*

On peut évidemment appliquer ce *pattern* pour un environnement discret pour lequel un maillage « naturel » de 1 serait inefficace (par exemple parce que l'on aurait $\bar{r} \gg 1$). Comme dans le *pattern* `StandardGrid`, il est également très facile de représenter par cette méthode divers champs, sous réserve que la discrétisation par un maillage m soit réaliste par rapport à l'échelle spatiale du champ. Enfin, comme précédemment on peut étendre cette méthode à des pavages variés (hexagones dans le plan par exemple), y compris, si la répartition des agents dans l'espace est très hétérogène, à toute partition de l'espace (en veillant à garder pour chaque cellule c un accès direct aux cellules adjacentes) : tessellation de Voronoï, *quadtrees* et autres structures récursives (Gosper, 1984), etc.

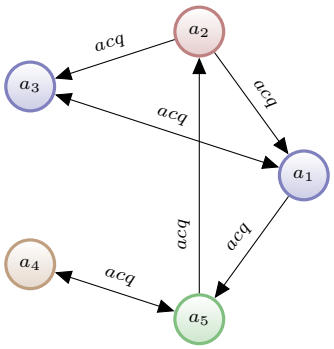


FIGURE 6. *Principe du pattern* `SocialNet`. *Comme dans le pattern* `AgentSet`, *l'environnement n'est pas concrétisé dans la mesure où sa topologie se réduit aux relations d'accointances entre les agents. Le calcul des voisins d'un agent est immédiat si on se limite à une distance de 1, ce qui est un cas très fréquent (dans cet exemple on a* $\nu(a_2, 1) = \{a_1, a_3\}$*)*

3.4. Le pattern `SocialNet`

Reste enfin à traiter une situation associée en règle générale aux environnements « sociaux », c'est-à-dire dans lesquels on considère en premier lieu les relations d'accointances entre agents.

3.4.1. Principe

Chaque agent est doté d'un attribut **`acquaintances`** correspondant à l'ensemble des agents qu'il connaît et avec lesquels il peut interagir directement. Cela revient à dire que la fonction pos_t est bijective puisque ce qui compte, ce ne sont pas les places où se trouvent les agents, mais *les relations d'accessibilité* de ces places : autrement dit les relations d'accointances définissent la matrice d'adjacence d'un graphe (orienté si elles ne sont pas réciproques) :

$$\forall i, j, e_{ij} = \begin{cases} 1 & \text{si } a_j \in \text{acquaintances}_t(a_i) \\ 0 & \text{sinon} \end{cases}$$

Deux situations se présentent donc :

1. Si l'agent ne peut interagir qu'à distance 1, la solution est triviale : la fonction **`neighbors`**(a_i, r) n'est définie que pour $r = 1$ et elle retourne l'attribut **`acquaintances`**. L'environnement est alors « virtuel » car totalement distribué dans les listes d'accointances des agents : c'est ce qu'on appelle parfois un « SMA purement communiquant » selon la terminologie proposée par Ferber (1995).

2. Si l'agent peut interagir à une distance plus grande (par exemple s'adresser aux accointances de ses accointances, ne serait-ce que pour les inclure dans ses propres accointances), il faut théoriquement passer par une fonction récursive qui peut être

coûteuse (si chaque agent possède en moyenne q accointances, la recherche des « voisins » dans un rayon r est en $\mathcal{O}(q^r)$). Dans ce cas, il est préférable de réifier la fonction de distance au moyen d'une matrice (δ_{ij}) où $\forall i, j\ \delta_{ij} = d(pos_t(a_i), pos_t(a_j))$. Cette matrice peut être calculée à partir de la matrice d'adjacence mentionnée ci-dessus, par exemple en appliquant l'algorithme de Floyd-Warshall (Floyd, 1962 ; Warshall, 1962) comme décrit dans l'algorithme 7. Le calcul des voisins utilise exactement le même algorithme que dans le *pattern* `AgentSet` (algorithme 1), à ceci près que l'on a $\forall a_i, a_j : d(a_i, a_j) = \delta_{ij}$.

3.4.2. Avantages et inconvénients

Le cas trivial est d'une grande simplicité d'implémentation : s'il peut être nécessaire de disposer d'un graphe pour initialiser les relations d'accointances entre agents (par exemple construction d'un *smallworld* (Watts, Strogatz, 1998)), celui-ci n'est plus nécessaire ensuite. Dans le second cas, l'utilisation d'une matrice des distances permet d'interagir avec des voisins dans un rayon arbitraire ; en revanche les changements qui peuvent survenir dans les relations d'accointances supposent de recalculer ces distances, ce qui est en $\mathcal{O}(n^3)$.

Algorithme 7 : Calcul de la matrice de distances avec le *pattern* `SocialNet` par Floyd-Warshall

```
for i ∈ [1, n] do
    δ_ii ← 0
    for j ∈ [1, n], j ≠ i do
        if a_j ∈ acquaintances_t(a_i) then
            δ_ij ← 1
        else
            δ_ij ← +∞
        end
    end
end
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            δ_ij ← min(δ_ij, δ_ik + δ_kj)
        end
    end
end
```

3.4.3. Usages

Ce *pattern* convient assez naturellement aux environnements dits « sociaux », bien qu'il ne s'appuie que sur des hypothèses de nature *topologique* relativement aux relations d'accessibilité entre les agents, vus comme les nœuds d'un graphe. Cette approche peut facilement s'étendre moyennant quelques transformations aux situations

où les accointances sont associées à des *poids* (par exemple pour représenter des liens d'amitié, de confiance, de dominance, un nombre de contacts, etc. entre les agents). La question clef reste de savoir si les agents sont autorisés à interagir avec leurs seules accointances (auquel cas ce *pattern* est très efficace) ou au-delà (un des trois autres *pattern* peut alors s'avérer plus intéressant).

4. Combinaison de *patterns*

L'approche que nous défendons ici, basée sur des *patterns*, se positionne plus volontiers dans la ligne esquissée par Schelfthout *et al*. (2002), que dans la tentative de construire une abstraction de première classe unique, complexe, monolithique capable de traiter de toutes les préoccupations logicielles simultanément, comme proposé par Weyns *et al*. (2007). Sans préjuger de l'intérêt de tels modèles intégrés pour l'analyse, la conception ou l'implémentation d'un SMA, nous préférons suivre une approche orthogonale qui au contraire vise à une séparation claire des diverses préoccupations (dans le même sens qu'en génie logiciel). Aussi nous défendons l'idée qu'il peut être plus pertinent de *combiner plusieurs de ces patterns fondamentaux*.

4.1. Un exemple concret

Par exemple, dans une situation où des robots doivent explorer un bâtiment, ils sont évidemment situés dans un environnement continu qui modélise l'espace physique et où ils peuvent indiquer la localisation des obstacles (et d'autres entités), et dans le même temps peuvent échanger de l'information avec leurs pairs (i.e. le réseau de leurs accointances). Aussi, « l'environnement » de ces robots est en réalité la combinaison de deux *patterns*, `AggregateGrid` et `SocialNet`. Les voisins d'un robot sont composés, d'une part, des entités perçues dans l'espace tridimensionnel qui l'entoure, et d'autre part, d'autres robots qui peuvent être joints par une communication radio (cf. figure 7).

4.2. Patterns et plateformes : l'exemple de NetLogo

En général les simulations multi-agents sont construites en réponse à un problème spécifique, qui souvent appelle (implicitement) un seul *pattern* : par exemple une simulation en sciences sociales s'appuiera exclusivement sur des réseaux d'accointances, un automate cellulaire est un cas de *pattern* `StandardGrid`, etc.

Les plateformes, en revanche, proposent en général de multiples solutions logicielles pour permettre des simulations diversifiées. Malheureusement, il n'y a en général aucune *explicitation* des caractéristiques et des usages possibles de ces solutions. Notre propos ici est de plaider en faveur d'une telle *explicitation*, pour laquelle notre approche par *patterns* est une démarche classique de Génie Logiciel.

Il est intéressant de noter par ailleurs que certaines plateformes de simulation, au premier rang desquelles NetLogo (Wilensky, 1999 ; Sklar, 2007), fournissent en fait

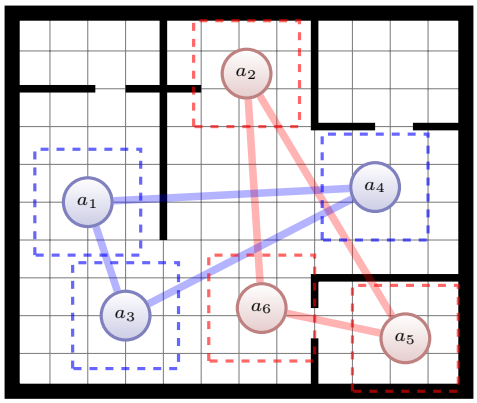


FIGURE 7. *Un exemple de situation impliquant la combinaison d'un pattern* `AggregateGrid` *et d'un pattern* `SocialNet`*, ici avec deux équipes de robots*

une implémentation des quatre *patterns* que nous proposons, sans les avoir néanmoins identifiés comme tels :

1. l'`AgentSet` est un type de données natif dans le langage NetLogo, qui contient la collection de toutes les instances d'agents d'une famille donnée ; ainsi par exemple pour créer 50 agents répartis au hasard, puis calcul (et colorisation) du voisinage à distance 1 de l'agent 0 :

```
create-turtles 50 [set color blue setxy random-xcor random-ycor]
ask turtle 0
  [ask turtles with [ distance myself <= 1 ] [ set color red ]]
```

2. le *pattern* `StandardGrid` est fourni par l'existence dans NetLogo de *patches*, qui constituent une discrétisation de l'environnement en cellules carrées de côté 1, et de primitives permettant d'accéder de façon immédiate à la cellule de l'agent (`patch-here`) ainsi qu'aux cellules adjacentes (soit dans un voisinage de Moore : `neighbors`, soit dans un voisinage de Von Neumann : `neighbors4`) ; l'exemple précédent s'écrit dans ce cas :

```
; les agents sont placés sur des coordonnées discrètes
create-turtles 50 [set color blue move-to one-of patches]
ask turtle 0
  [ask turtles-on (patch-set patch-here neighbors)
       [ set color red ]]
```

3. le *pattern* `AggregateGrid` est réalisé par le fait que chaque agent mobile (*turtle*) dans NetLogo, doté de coordonnées continues, est associé à un unique *patch* : ce système de *patches* assure donc la réalisation simultanée des deux *patterns*. ;

```
create-turtles 50 [set color blue setxy random-xcor random-ycor]
ask turtle 0
  [ask (turtles on-patches (patch-set patch-here neighbors))
       with [ distance myself <= 1] [ set color red ]]
```

```
; qui s'écrit aussi de façon plus concise :
;  [ask turtles in-radius 1 [ set color red ]]
```

4. le *pattern* `SocialNet` est fourni par la capacité à construire des liens (*links*) entre *turtles* et l'existence de primitives qui permettent de détecter les voisins associés par ces liens (`link-neighbors`); ici :

```
create-turtles 50 [set color blue setxy random-xcor random-ycor]
; création des liens d'accointance
ask turtles [ create-links-with (n-of 10 other turtles) []]
ask turtle 0
  [ask link-neighbors [ set color red ]]
```

De plus, ces concepts NetLogo ne sont en rien liés à un usage physique ou social particulier, et sont conçus pour pouvoir être utilisés simultanément pour répondre aux différents besoins des modèles de simulation.

Ainsi, nous voyons que notre approche basée sur les *patterns* conduit à une unification des concepts d'environnement physique et d'environnement social, qui d'ordinaire sont considérés comme distincts. Par ailleurs, elle pousse le concepteur à décomposer « l'environnement » d'un SMA, vu habituellement comme un tout relativement complexe, en autant d'environnements atomiques que nécessaire, chacun doué de sa propre raison d'être et d'une implémentation spécifique.

5. Discussion

Dans cette dernière section, nous cherchons à comparer notre définition de l'environnement à nos *patterns* par rapport à deux approches bien connues et très différentes, à savoir AGRE (Ferber *et al.*, 2005) de Ferber d'une part, et l'abstraction défendue par Weyns et Holvoet (2007) d'autre part. Nous voulons montrer comment nos propositions permettent de construire un cadre conceptuel plus simple et un cadre opérationnel plus précis, en dissociant mieux les diverses préoccupations qui sous-tendent ces travaux.

Dans AGRE (Ferber *et al.*, 2005), l'environnement s'ajoute à la méthodologie Agent-Groupes-Rôles existante. La notion de *monde* (*world*) est définie comme collection *d'espaces* (*spaces*) qui régulent les interactions entre agents. Chaque espace peut être soit un *groupe*, soit une *zone* (*area*). Ainsi cette approche maintient une séparation forte entre les environnements sociaux (groupes) où les agents peuvent échanger des messages selon leurs *rôles*, et l'environnement physique (unique), où les agents agissent au moyen de leur unique *corps*. Si nous appliquons nos définitions à AGRE, la distinction entre *zones* et *groupes* disparaît derrière le concept unifié d'environnement. Les groupes sont représentés par des graphes de rôles dont la topologie est définie par des protocoles d'interaction (*pattern* `SocialNet`), tandis que les zones sont représentées par un environnement souvent euclidien, et en général basé sur l'usage d'une grille (`StandardGrid` ou `AggregateGrid`).

Le modèle proposé par Weyns et Holvoet (2007) est une sorte de « modèle OSI » dans lequel l'environnement est défini comme un système en couches qui vont de l'infrastructure système à la médiation des interactions. Cette approche repose sur trois niveaux principaux : la couche de base, qui donne accès au contexte de déploiement ; la couche d'abstraction qui vise à faire écran aux primitives de bas niveau (par exemple, une requête SQL dans une base de données appartient à la couche de base, tandis qu'un artefact « agenda » lui correspondrait dans la couche d'abstraction) ; enfin, la couche de médiation des interactions, qui régule l'accès à des ressources partagées et sert de support aux interactions. Le modèle de référence qui est proposé par Weyns et Holvoet (2007, figure 4) détaille les principaux modules qui composent cette notion d'environnement : perception, observation et traitement des données, normes, interactions, communication...

Nous rejoignons Weyns sur l'importance à donner à l'environnement dans la conception et l'implémentation d'un SMA ; néanmoins sa proposition constitue un bloc monolithique où l'on ne distingue pas comment les normes, les interactions et les communications peuvent être organisées pour structurer le système. Plus précisément, la distinction entre couche d'abstraction et couche de médiation semble liée à la sémantique de l'information qui y est traitée, et paraît fortement dépendante du domaine d'application (la robotique mobile).

Si nous appliquons notre définition au modèle de Weyns, la perception et les modules d'interaction qui sont définis séparément seraient réifiés respectivement comme une distance dans un environnement euclidien continu (*pattern* `AggregateGrid`), et une distance sociale (*pattern* `SocialNet`). De plus, la réalisation de l'infrastructure de médiation qui régule les échanges entre agents est indépendante de la nature (physique ou sociale) de l'environnement.

Ainsi, sans préjuger de l'intérêt de ces modèles pour l'analyse ou l'implémentation des SMA, on constate que notre approche a pour effet d'unifier les concepts ordinairement disjoints d'environnement physique et d'environnement social, et de décomposer l'environnement, vu d'ordinaire comme un tout assez complexe, en autant d'environnements que nécessaire, dont l'implémentation est univoque.

6. Conclusion

La notion d'environnement, bien que centrale dans la conception de SMA, souffre d'une absence de définition univoque et applicable à de nombreux contextes d'usages. Dans cet article, nous proposons de considérer l'environnement comme l'expression du positionnement spatial ou social des agents et comme porteur d'informations dont la granularité est trop fine pour être représentées par des agents. La notion d'environnement est donc définie comme un espace, muni d'une quasidistance et de fonctions de placement et de contenu, permettant de déterminer les perceptions des agents.

Le passage de l'environnement conceptuel à l'environnement opérationnel peut alors être explicité à l'aide d'un nombre restreint de *design patterns* qui possèdent

chacun des propriétés les rendant plus ou moins pertinents selon les critères que l'on souhaite optimiser. Ainsi, nous avons identifié dans le cadre de la simulation quatre patterns d'environnements correspondant à des situations-types fréquentes pour lesquelles il existe une « bonne » solution logicielle :

– `AgentSet`, qui consiste à doter chaque agent d'une position, l'environnement se réduisant à une collection d'agents ;

– `StandardGrid`, qui représente l'environnement sous la forme d'un espace discret organisé sous la forme d'une grille ;

– `AggregateGrid`, qui permet de représenter des environnements continus en projetant les positions dans un espace discret ;

– `SocialNet`, qui représente l'environnement uniquement à travers les accointances des agents.

Dans ce cadre, nous défendons l'idée qu'un agent appartient généralement à plusieurs environnements (spatiaux ou sociaux). On signalera que cette démarche est confortée par les travaux que nous menons dans le domaine de la simulation multi-niveau, où nous avons déjà montré l'intérêt de décomposer un système complexe en environnements multiples pour modéliser de façon plus simple la variation de comportements des agents selon le niveau auquel ils opèrent (Picault, Mathieu, 2011).

De façon plus générale, la recherche systématique et l'explicitation de *patterns* dans les SMA, que ce soit pour l'analyse (Maudet *et al.*, 2015) ou la conception (Mathieu, Secq, 2012 ; Mathieu *et al.*, 2015), nous semblent constituer une démarche fructueuse et une contribution à un effort nécessaire de formalisation des concepts fondamentaux des systèmes multi-agents.

Bibliographie

Bautin A., Lucidarme P., Guyonneau R., Simonin O., Lagrange S., Delanoue N. *et al.* (2013). Cart-O-Matic project: autonomous and collaborative multi-robot localization, exploration and mapping. In *Proc. int. conf. on intelligent robots and systems (IROS).*

Behrens T. M., Hindriks K. V., Dix J. (2011, April). Towards an environment interface standard for agent platforms. *Ann. Math. Artif. Intell.*, vol. 61, n° 4, p. 261–295.

Colorni A., Dorigo M., Maniezzo V. (1991). Distributed optimization by ant colonies. In *Proc. 1st european conf. on artifical life (ECAL)*, p. 134–142. Elsevier.

Ferber J. (1995). *Les systèmes multi-agents. vers une intelligence collective*. InterÉditions.

Ferber J., Michel F., Báez J. (2005). AGRE: Integrating environments with organizations. In D. Weyns *et al.* (Eds.), *Proc. 1st int. workshop environments for multi-agent systems (E4MAS)*, vol. 3374, p. 48–56. Springer.

Ferber J., Müller J.-P. (1996). Influences and reactions: A model of situated multiagent systems. In *Proc. 2nd int. conf. on multiagent systems (icmas)*, p. 72–79. AAAI Press.

Floyd R. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, vol. 5, n° 6, p. 345.

Gamma E., Helm R., Johnson R., Vlissides J. (1994). *Design Patterns, elements of reusable object-oriented software*. Addison Wesley.

Gosper R. (1984). Exploiting regularities in large cellular spaces. *Physica D: Nonlinear Phenomena*, vol. 10, n° 1–2, p. 75–80.

Helleboogh A., Vizzari G., Uhrmacher A., Michel F. (2007). Modeling dynamic environments in multi-agent simulation. *J. Auton. Agents and Multi-Agent Systems (JAAMAS)*, vol. 14, n° 1, p. 87–116.

Kubera Y., Mathieu P., Picault S. (2010). Everything can be agent! In W. van der Hoek *et al*. (Eds.), *Proc. 9th int. joint conf. on auton. agents and multi-agent systems (AAMAS)*, p. 1547–1548.

Kubera Y., Mathieu P., Picault S. (2011). IODA: An interaction-oriented approach for multi-agent based simulations. *J. Auton. Agents and Multi-Agent Systems (JAAMAS)*, vol. 23, n° 3, p. 303–343.

Latombe J.-C. (1991). *Robot motion planning*. Kluwer Academic Publishers.

Macal C., North M. (2010). Tutorial on agent-based modelling and simulation. *Journal of Simulation*, vol. 4, p. 151–162.

Mamei M., Zambonelli F. (2006). Theory and practice of field-based motion coordination in multiagent systems. *Applied Artificial Intelligence*, vol. 20, n° 2–4, p. 305–326.

Mathieu P., Picault S., Secq Y. (2015). Design patterns for environments in multi-agent simulations. In Q. Chen *et al*. (Eds.), *Proc. 18th conf. on principles and practice of multi-agent systems (prima)*, p. 678–686. Springer.

Mathieu P., Secq Y. (2012). Environment updating and agent scheduling policies in agent-based simulators. In J. Filipe, A. Fred (Eds.), *Proc. 4th int. conf. on agents and artificial intelligence (ICAART)*, p. 170–175.

Maudet A., Touya G., Duchêne C., Picault S. (2015). Patterns multi-niveaux pour les SMA. In L. Vercouter, G. Picard (Eds.), *23e journées francophones sur les systèmes multi-agents (JFSMA)*, p. 19–28. Cépaduès.

Michel F. (2013). Intégration du calcul sur GPU dans la plate-forme de simulation multi-agent générique TurtleKit3. In S. Hassas, M. Morge (Eds.), *21e journées francophones sur les systèmes multi-agents (JFSMA)*, p. 189–198. Cépaduès.

Odell J., Van Dyke Parunak H., Fleischer M., Brueckner S. (2003). Modeling agents and their environment. In *Proc. 3rd int. conf. on agent-oriented software engineering (AOSE)*, p. 16–31. Springer.

Okuyama F., Bordini R., Rocha Costa A. da. (2005). ELMS: An environment description language for multi-agent simulation. In D. Weyns *et al*. (Eds.), *Proc. 1st int. workshop environments for multi-agent systems (E4MAS)*, vol. 3374, p. 91–108. Springer.

Payet D., David D., Sébastien N. (2009). Auto-génération d'environnement : l'exemple d'infinite forest. In Z. Guessoum, S. Hassas (Eds.), *17e journées francophones sur les systèmes multi-agents (JFSMA)*, p. 165–174. Cepaduès.

Picault S., Mathieu P. (2011). An interaction-oriented model for multi-scale simulation. In T. Walsh (Ed.), *Proc. 22nd int. joint conf. on artificial intelligence (IJCAI)*, p. 332–337. AAAI.

Ricci A., Viroli M., Omicini A. (2007). CArtAgO: A framework for prototyping artifact-based environments in MAS. In D. Weyns *et al*. (Eds.), *Proc. 3rd int. workshop environments for multiagent systems (E4MAS)*, vol. 4389, p. 67–86. Springer.

Russell S., Norvig P. (1995). *Artificial intelligence. a modern approach*. Prentice Hall.

Sanders L., Pumain D., Mathian H., Guérin-Pace F., Bura S. (1997). SIMPOP: a multi-agents system for the study of urbanism. *Environment and Planning B*, vol. 24, p. 287–305.

Schelfthout K., Coninx T., Helleboogh A., Holvoet T., Steegmans E., Steegmans E. *et al*. (2002). Agent implementation patterns. In *Workshop on agent-oriented methodologies, 17th annual acm conf. on object-oriented programming, systems, languages, and applications (OOPSLA)*, p. 119–130.

Simonin O., Gechter F. (2005). An environment-based methodology to design reactive multi-agent systems for problem solving. In D. Weyns *et al*. (Eds.), *Proc. 2nd int. workshop environments for multi-agent systems (e4mas)*, vol. 3830, p. 32–49. Springer.

Sklar E. (2007). NetLogo, a multi-agent simulation environment. *Artificial Life*, vol. 13, n° 3, p. 303–311.

Steen L., Seebach J. (1995). *Counterexamples in topology*. Dover Publications. (2nd ed.)

Van Dyke Parunak H. (2011). Between agents and mean fields. In D. Villatoro *et al*. (Eds.), *Proc. int. workshop on multi-agent based simulaiton (MABS XII)*, vol. 7124. Springer.

Warshall S. (1962). A theorem on boolean matrices. *Journal of the ACM*, vol. 9, n° 1, p. 11–12.

Watts D., Strogatz S. (1998). Collective dynamics of 'small-world' networks. *Nature*, vol. 393, n° 6684, p. 440–442.

Weyns D., Holvoet T. (2007). A reference architecture for situated multiagent systems. In D. Weyns *et al*. (Eds.), *Proc. 3rd int. workshop environments for multi-agent systems (E4MAS)*, vol. 4389, p. 1–40. Springer.

Weyns D., Omicini A., Odell J. (2007). Environment as a first class abstraction in multiagent systems. *J. Auton. Agents and Multi-Agent Systems (JAAMAS)*, vol. 14, n° 1, p. 5–30.

Weyns D., Parunak H., Michel F., Holvoet T., Ferber J. (2005). Environments for multiagent systems. State-of-the-Art and research challenges. In D. Weyns *et al*. (Eds.), *Proc. 1st int. workshop environments for multi-agent systems (E4MAS)*, vol. 3374, p. 1–47. Springer.

Wilensky U. (1999). *Netlogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. (`http://ccl.northwestern.edu/netlogo/`)