Design patterns for environments in multi-agent simulations

Philippe Mathieu, Sébastien Picault, Yann Secq

▶ **To cite this version:**

Philippe Mathieu, Sébastien Picault, Yann Secq. Design patterns for environments in multi-agent simulations. Principles and Practice of Multi-Agent Systems, PRIMA 2015, 18th International Conference, Oct 2015, Bertinoro, Italy. pp.678-686, 10.1007/978-3-319-25524-8_51 . hal-01248799

HAL Id: hal-01248799
https://inria.hal.science/hal-01248799

Submitted on 15 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Design patterns for environments in multi-agent simulations

Philippe Mathieu, Sébastien Picault, and Yann Secq

SMAC team, CRIStAL UMR CNRS 9189
University of Lille – Science and Technology
59655 Villeneuve d'Ascq cedex, France
{philippe.mathieu,sebastie.picault,yann.secq}@univ-lille1.fr
http://cristal.univ-lille.fr/SMAC

Abstract. In the field of multi-agent based simulation (MABS), the concept of environment is omnipresent, though poorly defined. We argue here that depending on the modeling of space and of relations between agents, only a few efficient implementations can be set up. We aim at formalizing the core functions of environments, so as to highlight the computational answers to possible modeling choices. This unifying approach leads to the identification of four paradigmatic *Design Patterns*, associated with specific *atomic environments*, which can be composed in order to tackle complex situations.

Keywords: Environments; Modeling; Parsimony; Engineering; Design Patterns

1 Introduction

In this paper, we focus on the definition and on the implementation of the concept of "environment" in MABS. It is generally admitted that agents act "in an environment" [1, 2], and most of the time "environments are simply *assumed* as given" [3]. Indeed everybody is *supposed to know* what an environment is, but this implicit reference to common sense makes the environment the neglected concept in MABS design. Hence, the term itself refers to heterogeneous realities, from an abstract description of an "habitat" (in an ecological sense) where the agents "live", to the hardware/software execution infrastructure of a platform [4] and interoperability issues [5, 6]. Some exceptions are found in Agent-Based Simulation, where the issue is crucial [7, 8]. Several criteria for characterizing environments have been proposed by [2, ch. 2] (e.g. accessibility, determinism, static vs. dynamic, continuous vs. discrete), yet a close reading shows that they are not exactly environment features, but rather peception, cognition and action capabilities *of the agents* that are situated in it.

In contrast, [4] addresses crucial issues such as eliciting the link between agents and environments, the need for a taxonomy of environments, or the essential role of topological relations. Indeed, in order to act each agent needs to

know on which other agents it can perform actions (including communication). As the agent has only a limited (local) perception, it can (must) interact only with its *neighbors. Thus, the main issue is: how does an agent determine its neighbors? Essentially, the environment* ought *to be an answer to this question.*

Our goal in this paper is to search for a minimal, formal definition of what the term "environment" can mean in a MABS, and to show how to reduce the diversity of environments found in existing MABS to a few combinable *patterns*, as suggested by [9]. We argue that the choice of appropriate techniques for modeling environments only depends on a very limited number of criteria, as in the classical Design Patterns used in software engineering [10].

In the following, *each entity of the simulation model is represented by an agent* as suggested in [11, 12]. We also assume *discrete time*, so that, at each time step, the MABS contains a set of agents, denoted by $\mathcal{A}_t$. At this stage we do not assume any special hypothesis regarding the capabilities of action, perception or cognition of those entities. From the environment indeed, "objects", "artifacts" or "agents" are undiscernible (proactivity being an internal feature). Obviously though, *everything* is not meant to be reified by agents: depending on the analysis level required in the model, there is a *granularity level, beyond which would-be entities are modeled by an aggregated information* (see § 2.2).

In the next section, we propose a minimal abstract definition of a MABS environment; then, we describe four patterns induced by modeling and implementation choices; finally, we explain how those patterns may be composed for building usual MABS environments.

2 Formalizing the concept of environment

We advocate that there are two main roles devoted to the environment in a MABS: first, *to place agents* so as to define neighborhood and accessibility relationships; second, *to provide spatial information*, which reflects underlying levels that would not be reified through agents.

2.1 First purpose: placing agents

We suppose a set E composed of "places" which can host agents, and can be *arbitrarily chosen* w.r.t the nature of the world to be modeled. To determine accessibility relations from one place to the others, we also suppose a distance function $\mathbf{d} : \mathbf{E} \times \mathbf{E} \to \mathbb{R}^+ \cup \{+\infty\}$, so that (E, d) is a *metric space* in which the distance induces a *topology*. This minimalistic description covers many situations, e.g. the usual 3D space ($E = \mathbb{R}^3$) with the euclidean distance; a 2D grid with the Moore neighborhood ($E = \mathbb{Z}^2$, d being the Chebychef distance); a continuous 2D "toric" space of width w and height h (quotient space $E = \mathbb{R}^2/(w\mathbb{Z} \times h\mathbb{Z})$); an acquaintance network (graph, i.e. a set E of nodes, endowed with the geodesic distance); etc. "Non situated" agent corresponds to a singleton $E = \{p\}$, endowed with the trivial distance $\forall x, y\ d(x, y) = 0$. Generally only a subset $\mathcal{E} \subset E$ is used in the simulation.

Since the first purpose of the environment is to locate agents, $\mathcal{E}$ must be endowed with a function designed to provide the agents with a *position* at each time step: $\boldsymbol{pos}_t : \mathcal{A}_t \to \mathcal{E}$. Then, an *environment* can be defined as *a subset $\mathcal{E}$ of a metric space (E, d) endowed with a family of functions $(pos_t)_{t\in[0,T_{\max}]}$ which give the successive positions of agents within $\mathcal{E}$.* Those functions can often be specified only in *extension*; besides, each agent a is responsible for handling the value of $pos_t(a)$ (e.g. as an attribute). Moreover, the transition between pos_t and pos_{t+1} is itself the consequence of the behavior of each agent. Nevertheless, this function-based description also covers cases where positions can be specified in *intension* (e.g. trajectories following physical laws). We also define the *reciprocal function* of the position, i.e. the *content* of a point of the environment, as follows: $\forall p \in \mathcal{E}, \boldsymbol{cont}_t(p) = \{a \in \mathcal{A}_t \mid pos_t(a) = p\}$.

From this formalization, we claim that two features can discriminate between several kinds of environments and their possible implementation: first, the *nature of the metric space* (E, d) (continuous, lattice, graph, singleton...), and second, the *decision (or need) to provide explicitly those functions*, especially $cont_t$. Indeed, the behavior of agents crucially depends on their *neighborhood function*, devoted to *identifying their neighbors*, i.e. with whom they can interact, namely $\nu_t : \mathcal{A}_t \to \wp(\mathcal{A}_t)$. Among an infinite variety of possible functions, a simple example is the *topological ball* centered on the position of a and of radius r: $\nu_t^r(a) = \{a' \in \mathcal{A}_t \setminus \{a\} \mid d(pos_t(a), pos_t(a')) \leq r\}$. The *computational cost* of computing a neighborhood may require a time/space compromise between algorithmic efficiency and data structures. Thus a given neighborhood function may be implemented through several forms, as explained in section 3.

2.2 Second purpose: providing information

Depending on the modeling scale, some underlying levels may require a macroscopic approximation. For instance, pheromones, as entities (molecules) should be represented by agents (endowed with physico-chemical behaviors: brownian motion, evaporation, degradation). But, in practice, they are usually aggregated numerically (like the concentration in chemistry) and mapped to a discretized space (the cells of a grid), in the mere intend of reducing the computational cost of handling a large number of entities, but at the expense of approximating diffusion phenomena. The MAS literature does not agree on any convention for naming such a purely informational representation. The concept of *field*, already used in reactive navigation or for pheromones (*Potential Fields* [13] and *Mean Fields* [14]) seems a convenient way of modeling *any kind of "information" afforded by the environment* (e.g. gravitation forces, temperature, obstacles) and *not reified by agents*. A field is thus a function which maps points of $\mathcal{E}$ to any set of information pieces $\mathcal{I}$; e.g. the pheromone level in grid cells would be: $\boldsymbol{phero}_t : \mathbb{Z}^{\boldsymbol{2}} \to \mathbb{R}^+$, defined in *extension* and changing at each time step through a classical diffusion-evaporation algorithm [15]. A field can sometimes be defined in *intension*, e.g. the gravity field in the usual 3D space $\boldsymbol{g} : \mathbb{R}^3 \to \mathbb{R}^3$ which maps each point to an acceleration vector.

We especially emphasize this representation of spatial information through field functions, because the modeling decision (or needs) of representing those functions in an explicit way or not, leads to specific constraints regarding the implementation of the environment.

3 Four fundamental patterns

We intend to show that a limited number of implementation families can be identified for MABS environments, depending on the nature of the metric space (E, d), and on the choice of reifying (or not) $cont_t$. Our point here is not to aim at a normative approach, nor at a comprehensive listing of cases, but rather at describing recurrent typical usages and their distinctive features. We introduce below the four fundamental *patterns* we have identified, with their principles, advantages and drawbacks. We show how to handle efficiently the search for neighbors and the motion of agents, i.e. how to actually compute the values of the functions ν_t^r and pos_{t+1} for each agent (resp. `neighbors(a,r)` and `move_to(a,p)`). Finally, we indicate when each pattern is best suited.

3.1 The "AgentSet" pattern

Principle. Each agent is endowed with an attribute `pos` containing $pos_t(a)$. The environment is reduced to a set of agents ($\mathcal{A}_t$) and the distance function d.

Algorithm 1: Calculation of the neighbors of an agent (`AgentSet` pattern)

```
neighbors(a_i, r):
N ← ∅
for a_j ∈ A_t, a_j ≠ a_i do
    if d(pos_t(a_i), pos_t(a_j)) ≤ r then
        N ← N ∪ {a_j}
    end
end
return N
```

Advantages and drawbacks. Only the function pos_t is explicitly reified. Conversely, the computational complexity of neighbor perception in the simulation is in $\mathcal{O}(n^2)$, where $n = |\mathcal{A}_t|$. Indeed, each agent has to check its distance to the $(n-1)$ other agents (algorithm 1). However, when an agent moves, it only has to modify its `pos` attribute as follows: $\texttt{move\_to}(a_i, p) : pos_{t+1}(a_i) \leftarrow p$.

Usages. The `AgentSet` pattern proves convenient for implementing plain environments containing a *small number of agents*, otherwise finding the neighbors is costly. It is also suited for fields given in intension, since the corresponding functions have just to be implemented together with the set of agents and the distance function.

3.2 The "StandardGrid" pattern

Principle. In many models, the environment to build is a discrete space organized as a geometric *lattice*, indexable by a system of discrete coordinates of dimension k (such as $E = \mathbb{Z}^k$). The discrete positions can also be seen as "cells". To implementation such environments, the structure of the latter pattern is endowed with a *grid*, i.e. a tessellation of the environment, through a k-dimension array containing sets of agents. This explicit reification of function $cont_t$ provides a direct access to all agents situated on any point (or "cell") p. Besides, all points adjacent to p are retrieved easily, either by their coordinates or by cross-references between cells: hence, an immediate calculation of the neighbor points of p (`neighborhood` in algorithm 2).

Algorithm 2: Neighbors calculation (`StandardGrid` pattern)

`neighbors`(a_i, r):
$\texttt{N} \leftarrow \emptyset$
$\texttt{neighborhood} \leftarrow \{p \in E \mid d(pos_t(a_i), p) \leq r\}$
for $c \in$ `neighborhood` **do**
  $\texttt{N} \leftarrow \texttt{N} \cup cont_t(c)$
end
return $\texttt{N} \setminus \{a_i\}$

Algorithm 3: Movement of an agent (`StandardGrid` pattern)

`move_to`(a_i, p):
if $pos_t(a_i) \neq p$ **then**
  $cont_{t+1}(pos_t(a_i)) \leftarrow$
  $cont_t(pos_t(a_i)) \setminus \{a_i\}$
  $cont_{t+1}(p) \leftarrow cont_t(p) \cup \{a_i\}$
end
$pos_{t+1}(a_i) \leftarrow p$

Advantages and drawbacks. Direct access to all agents on a point provides a considerable speed-up of neighbors search, since they are necessarily on the same cell as a_i or on adjacent cells. Finding the neighbors of n agents within a radius of r (algorithm 2), has a cost in $\mathcal{O}(n.r^k)$ (e.g., with a Moore neighborhood and $k = 2$, neighbors are in a square of $(2r + 1)^2$ cells). Conversely, moving an agent requires more complicated updates than in the previous pattern, in order to ensure consistency between pos_{t+1} and $cont_{t+1}$ (algorithm 3).

Usages. This pattern fits environments based on *integer coordinates*, especially if the average perception radius $\bar{r}$ of the agents (within which they search their neighbors) fulfils $\bar{r}^k \ll n$. It is also well suited to *represent field functions* even when given in extension (yet, assuming that those fields are defined on discrete coordinates): the values of the field on each point are held by a k-dimension array as well. This pattern also handles obstacles easily, when represented either by agents (which prevent the access to their cell), or by fields, or by the distance function (topology of the cells). Noteworthy, this grid-based approach is obviously *not limited to square tessellations*; many simulations e.g. in geography use hexagons [16], which also constitute a lattice of $\mathbb{R}^2$. We can also mention a very frequent *simplification* of this pattern: when the model assumes that two agents cannot occupy the same position, then the grid is simply a k-dimension array of agents (instead of a set of agents).

3.3 The "AggregateGrid" pattern

Principle. The previous pattern cannot be used when the environment is based on a continuous space, because of the practical impossibility to provide exhaustive adjacency relations. Yet, if the function $cont_t$ cannot be reified as such, a discrete approximation can be build by *"projecting" the continuous positions in E on a discrete space $\mathbb{E} \subset E$*. Therefore, a discretization of E, depending on a mesh size m, must be defined: $\boldsymbol{cell}_m : E \rightarrow \mathbb{E}$. In other words we need: $\forall c \in \mathbb{E}, \exists c' \in \mathbb{E}, c' \neq c$ such that $d(c, c') \leq m$. For instance, taking $E = \mathbb{R}^2$, it is quite natural to choose $\mathbb{E} = \mathbb{Z}^2$; then a simple discretization function would be: $\forall (x, y) \in \mathbb{R}^2, cell^m(x, y) = (\lfloor \frac{x}{m} \rfloor, \lfloor \frac{y}{m} \rfloor)$ (where $\lfloor u \rfloor$ denotes the floor of u). An "aggregate" form of the function $cont_t$ can then be defined, so as to calculate the set of all agents situated in a "cell" c: $\forall c \in \mathbb{E}, cell\_cont_t^m(c) = \{a \in \mathcal{A}_t \mid cell^m(pos_t(a)) = c\}$.

Now, the function $cell\_cont_t^m$ can be reified exactly like in the previous pattern, through a k-dimension array `cell_cont` which contains sets of agents (the set of all agents situated in cell $cell_m(p)$).

Algorithm 4: Neighbors calculation (`AggregateGrid` pattern)

```
neighbors(a_i, r):
N ← ∅
c_0 ← cell^m(pos_t(a_i))
neighborhood ← {p ∈ 𝔼 | d(c_0, p) ≤ ⌈r/m⌉}
for c ∈ neighborhood do
    for a_j ∈ cell_cont_t^m(c), a_j ≠ a_i do
        if d(a_i, a_j) ≤ r then
            N ← N ∪ {a_j}
        end
    end
end
return N
```

Algorithm 5: Movement of an agent (`AggregateGrid` pattern)

```
move_to(a_i, p):
if cell^m(pos_t(a_i)) ≠ cell^m(p) then
    cont_cell_{t+1}(pos_t(a_i)) ←
    cont_cell_t(pos_t(a_i)) ∖ {a_i}
    cont_cell_{t+1}(p) ←
    cont_cell_t(p) ∪ {a_i}
end
pos_{t+1}(a_i) ← p
```

Advantages and drawbacks. Due to the discretization of the continuous space, this pattern benefits from the computational speed-up of a grid structure (cf. algorithm 4). At the same time, the arbitrary size of the mesh m allows a fine tuning of this discretization if needed: the computational cost of neighbors detection for n agents, within a radius of r, is in $\mathcal{O}(n.(\lceil \frac{r}{m} \rceil)^k.q)$ (where $\lceil u \rceil$ denotes the ceiling of u and q the *density* of agents i.e. the average number of agents in each cell). Again, the counterpart is a more complicated technique for updating the content of cells when agents move (cf. algorithm 5).

Usages. This pattern is relevant for modeling continuous environments containing a large number of agents. It can be also useful for implementing a *discrete* environment where the "natural" mesh size of 1 would be quite inefficient (i.e.

when $\bar{r} \gg 1$). Like in the `StandardGrid` pattern, the representation of fields is quite straightforward, provided that the mesh size m leads to a realistic approximation w.r.t. the spatial scale of the field. This approach can be extended to other tessellation methods (e.g. hexagons in the plane), including partition methods suited for heterogeneous spatial distribution of agents (e.g. Voronoi, quadtrees, etc.): the key principle is that each cell must have direct access to adjacent cells.

3.4 The "SocialNet" pattern

Principle. Each agent owns an attribute `acquaintances` which contains the set of all agents it *knows* and can interact with. Then pos_t is bijective, i.e. the key issue is not knowing the places where the agents are situated, but rather the accessibility relations between places. Here, the *topology* of the environment comes first, because acquaintances relations are equivalent to the adjacency matrix of a graph (possibly directed if some acquaintances are not reciprocal): $\forall i, j, e_{ij} = 1$ if $a_j \in \text{acquaintances}_t(a_i), 0$ otherwise.

Thus, two very different situations may occur. If agents are allowed to interact *with their own acquaintances only*, the `neighbors`(a_i, r) function is defined only for $r = 1$, and trivially returns the `acquaintances` attribute. The environment can be considered "virtual" (or "purely communicating" [1]), being distributed through the acquaintances lists. On the contrary, if agents are allowed to interact *within a wider range*, then the cost of the corresponding breadth-first search of "neighbors" within a radius of r, is in $\mathcal{O}(\bar{q}^r)$ (with $\bar{q}$ the average size of acquaintances lists). Instead, the distance function can be reified through a distance matrix (δ_{ij}) where $\forall i, j\ \delta_{ij} = d(pos_t(a_i), pos_t(a_j))$. It gives the shortest paths in the graph (computed using e.g. the Floyd-Warshall algorithm [17]). Then, the determination of the neighbors follows the same method as in the `AgentSet` pattern (algorithm 1).

Advantages and drawbacks. The trivial case is quite simple to implement. In the second situation, using a distance matrix provides the ability for any agent to interact with neighbors within an arbitrary radius. Yet, if changes occur in the acquaintances relations, this matrix has to be updated by recomputing the shortest paths, which is in $\mathcal{O}(n^3)$ in the Floyd-Warshall algorithm.

Usages. This pattern is naturally suited to "social" environments though it actually only assumes *topological* hypotheses regarding the accessibility relations between agents, regarded as nodes of a graph. Obviously, the approach can be extended with few transformations in situations where acquaintances are associated with *weights*.

4 Combining patterns

The pattern-based approach to environments we defend here, is rather a continuation of the method sketched in [9], than an attempt to build a unique, complex,

monolithic first-class abstraction which is able to deal with all concerns at the same time as proposed in [18]. Without prejudging the interest of such integrated models for the analysis, design or implementation of MAS, we prefer to follow an orthogonal approach which ensures on the contrary a clear separation of concerns (in the same sense as in software engineering). Therefore we argue that it is more advisable to *combine several of those fundamental patterns*.

For instance, in a case where simulated robots have to explore a building, they are indeed situated in a continuous environment where they find obstacles (other entities), and at the same time may exchange information with their peers (the network of their acquaintances). Thus, "the" environment is actually a combination between the "AggregateGrid" and the "SocialNet" patterns. The neighbors of a robot is composed of, on the one hand, the entities perceived in the 3D surrounding space, and on the other hand, other robots that can be reached through radio communications.

Noteworthy, a multi-agent based simulation platform like NetLogo [19] actually provides an implementation for each of the patterns we propose, although not explicitly identified as such. The "AgentSet" is a native data type in the NetLogo language, which contains the collection of all agents of a kind. The "StandardGrid" pattern is a built-in feature due to the existence of NetLogo *patches*, which are 1×1 square cells discretizing the environment. The "AggregateGrid" pattern is realized by the fact that each NetLogo "turtle", which is endowed with floating-point coordinates, is located on a single patch (hence patches implement the two patterns simultaneously). The "SocialNet" pattern is provided by the ability to build *links* between turtles (and use them to detect "link-neighbors"). Besides, those NetLogo concepts do not imply any special use either as physical or social features, and are meant to be used together for implementing the various needs of a simulation model.

Thus, we note that our pattern-based approach leads to a unification of concepts of physical and social environments, that are usually kept distinct. Besides, it also pushes the designer to decompose "the" environment of the MAS, which is usually seen as a rather complex whole, into as many atomic environments as needed, each one with a univocal purpose and implementation.

5 Conclusion and perspectives

In this paper, we assume two main purposes for an "atomic environment": the spatial or social placement of agents, and the provision of information that cannot be represented by agents at the chosen modeling scale. Thus, an atomic environment is defined primarily as a *space*, endowed with a distance measure, with positioning and content functions, and aimed at finding agents and neighbors of agents: those features lead to a limited number of *design patterns*. The four patterns presented here seem to cover a wide range of applications in agent-based simulation.Besides, instead of modeling a single, complex environment in a MAS, we recommend to rather combine several patterns with a clear separation of concerns and purposes.

References

1. Ferber, J.: Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence. Addison Wesley (1999)
2. Russell, S., Norvig, P.: Artificial Intelligence. Prentice Hall (1995)
3. Okuyama, F., Bordini, R., da Rocha Costa, A.: ELMS: An environment description language for multi-agent simulation. In Weyns, D., et al., eds.: E4MAS'04. Volume 3374 of LNCS. Springer (2005) 91–108
4. Weyns, D., Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems. State-of-the-Art and research challenges. In Weyns, D., et al., eds.: E4MAS'04. Volume 3374 of LNCS. Springer (2005) 1–47
5. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: A framework for prototyping artifact-based environments in MAS. In Weyns, D., et al., eds.: E4MAS'06. Volume 4389 of LNCS. Springer (2007) 67–86
6. Behrens, T.M., Hindriks, K.V., Dix, J.: Towards an environment interface standard for agent platforms. Ann. Math. Artif. Intell. **61**(4) (April 2011) 261–295
7. Helleboogh, A., Vizzari, G., Uhrmacher, A., Michel, F.: Modeling dynamic environments in multi-agent simulation. J. Auton. Agents and Multi-Agent Systems (JAAMAS) **14**(1) (2007) 87–116
8. Weyns, D., Holvoet, T.: A reference architecture for situated multiagent systems. In Weyns, D., et al., eds.: E4MAS'06. Volume 4389 of LNCS. Springer (2007) 1–40
9. Schelfthout, K., Coninx, T., Helleboogh, A., Holvoet, T., Steegmans, E., Steegmans, E., Weyns, D.: Agent implementation patterns. In: Workshop on Agent-Oriented Methodologies, 17th Annual ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). (2002) 119–130
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison Wesley (1994)
11. Kubera, Y., Mathieu, P., Picault, S.: Everything can be agent! In: 9th Int. Joint Conf. on Auton. Agents and Multi-Agent Systems (AAMAS). (2010) 1547–1548
12. Picault, S., Mathieu, P.: An interaction-oriented model for multi-scale simulation. In: 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI), AAAI (2011) 332–337
13. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers (1991)
14. Van Dyke Parunak, H.: Between agents and mean fields. In Villatoro, D., et al., eds.: MABS XII. Volume 7124 of LNCS. Springer (2011)
15. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: 1st European Conf. on Artifical Life (ECAL), Elsevier (1991) 134–142
16. Sanders, L., Pumain, D., Mathian, H., Guérin-Pace, F., Bura, S.: SIMPOP: a multi-agents system for the study of urbanism. Environment and Planning B **24** (1997) 287–305
17. Floyd, R.: Algorithm 97: Shortest path. Communications of the ACM **5**(6) (1962) 345
18. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. J. Auton. Agents and Multi-Agent Systems (JAAMAS) **14**(1) (2007) 5–30
19. Wilensky, U.: Netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1999) `http://ccl.northwestern.edu/netlogo/`.