The TurtleKit Simulation Platform: Application to Multi-Level Emergence

Fabien Michel, Gregory Beurier and Jacques Ferber

LIRMM Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier.
C.N.R.S. - Université Montpellier II, 161 rue Ada 34392 Montpellier Cedex 5 - France
`{fmichel, beurier, ferber}@lirmm.fr`
WWW home page: `http://www.lirmm.fr/~{fmichel, beurier, ferber}`

Abstract. This paper presents the TurtleKit simulation platform. This platform relies on the combination of a Logo simulation model with high level programming langages.

1 Introduction

Within the framework of computer simulation, the multi-agents approach is an attractive alternative to equation-based models[9]. Agent based simulations (ABS) allow to model individuals, their behaviours and their interactions. As this field of research is constantly growing, there is today plenty of tools and platforms which provide different means to develop, execute and analyse this kind of complex systems. The majority of the existing platforms have been designed according to a particular area of research such as ecology (Cormas[2], Echo[6]), robotic (MissionLab[7]), ethology (LiveWorld[13]) or multi-agent coordination(Mass[5]). Thus they propose simulation tools (agent architecture, environment model, etc.) which are designed for a particular kind of problem.

In this framework, some platforms are not dedicated to a research field but address a particular audience. For instance, the platform Swarm [12] defines a generic simulation mechanism and affords an important library of environment, agent architectures, simulation tools, etc. The underlying idea is to give to the user advanced programming tools that will enable him to develop a simulator suited to his particular problem. So this platform is designed for advanced users. At the opposite, other platforms such as StarLogo [11] or NetLogo [10] are related to a specific model of simulation. Using a Logo based simulation language, they are designed to quickly develop simulations for educational purposes. Thus they intentionally focus on simplicity and do not afford advanced programming tools for extending the platform possibility. They are designed for newbie users.

In this paper, we present the TurtleKit platform which relies on a logo-based simulation model. Unlike StarLogo [11] or NetLogo [10], TurtleKit proposes a Logo programming approach while giving all the possibilities afforded by high level programming languages such as Java and Python. Dedicated to advanced users, the underlying idea of this open source platform is to enable the use of the Logo simulation model (environment, agent architecture, etc.) while giving several programming means to extend the platform possibilities. Through an application of multi-level emergence, we show how this platform can be used and extended.

2 TurtleKit Overview

2.1 Logo Programming Approach

The Logo language, a dialect of Lisp, was designed in the late sixties at the Massachusetts Institute of Technology (MIT). The main motivation was to provide a learning tool that does not require programming skills while giving immediate graphical results. The principle is to manipulate a graphical animat, a *turtle*[1], typing simple commands to make it move and draw shapes on the screen. For instance the *forward 10* command makes the turtle moves forward 10 steps. Logo programs are usually collections of small procedures that define turtle behaviours which can be combined to achieve more complex behaviours.

In platforms like StarLogo or NetLogo, the idea is to provide a simulation environment made of several turtles, and thus to model multi-agent systems (MAS). TurtleKit is directly inspired of this approach. In this kind of simulation model, turtles are agents that live on a two-dimensional world. The environment is spatially discretised into *patches* which define the agents' location and contain local environmental properties such as the ground's colour, pheromone concentrations and so on. Turtles have the ability to interact with the world, other turtles and patches, using *turtle commands*: they can move, perceive their local environment and modify the properties of patches and other turtles.

So, implementing a simulation with MultiLogo-like languages mainly relies on defining new *procedure commands* which are a collection of primitive commands and of other *procedure commands*:

```
to wiggle
    fd 1
    rt random 50
    lt random 50
end
```

For instance the *wiggle* procedure defines a turtle behaviour which consists in using the *right turn (rt)*, the *left turn (lt)*, the *forward (fd)* and the *random* primitive commands to define a random walk. Then, this procedure as so defined can be used into another new procedure to define a more complex behaviour and so on. Most of the time, the user will define a "top level" procedure such as *do-it* which defines the agent's behaviour as a logical sequence of procedures. Then,

[1] Originally a robotic creature that sat on the floor and could be directed to move around by typing commands at the computer.

this “top level” procedure is invoked on all the turtles to start the simulation process. Additionally, a simulation entity named *observer* can also interact with the environment using *observer commands*. This feature is also used to monitor the activity of the turtles and patches.

2.2 Using High Level Programming Languages

The main advantage of Multi-Logo platforms relies on the fact that they proposes a complete simulation oriented programming language that focuses on simple agent/environment models. This programming language provides an exhaustive list of primitive commands and complex behaviours can be easily defined starting from simple primitive commands.

However, these platforms have been designed for educational purposes and not for performing large scale simulations. Notably, they intentionally do not afford advanced programming tools that can be used to extend the platform's possibilities since they only allow to program the actors of the simulation (turtles, patches and the observer). For instance, it is not possible to program the platform to display the world according to different system point of views at the same time. As we will see in section 4, this kind of feature is very appreciable and having different graphical representations of the world is a feature that cannot be ignored when exploring behaviour of complex systems [2]. In the same way, these platforms do not provide code reuse facilities like inheritance since they focus on simplicity and target a particular audience.

Developing TurtleKit, our main goal is to bridge the gap between the use of a Multi-Logo simulation model and all the possibilities of high level programming languages. Hence, TurtleKit is an open source platform written in java. It provides a default set of classes that can be used or extended regarding the simulation requirements. Thus programming facilities such as inheritance are naturally available.

2.3 Using Agent Oriented Programming

Initially, the development of the TurtleKit platform was made to validate the simulator building tools that we have implemented in the MadKit[2] multi-agent platform [8]. Since MadKit is a generic deployment environment for developing MAS, it provides support for building organisations, communication among agents, debugging, system probing, distribution and so on. Thus TurtleKit naturally benefits of these features. Moreover, the TurtleKit platform is a MAS itself. In fact, every module of TurtleKit is a regular MadKit agent and has the ability to communicate with any other agent using messages, including the turtles, the scheduler, the displayers and the monitor tools. To keep the simplicity of the MultiLogo approach, all these features are encapsulated and can be ignored when developing a simulation with TurtleKit. However they are always available and can be used to extend the default tools of TurtleKit.

[2] madkit

2.4 Turtle Behaviour Implementation

In TurtleKit, every kind of turtle is implemented in a separated java class that inherits of the super class *Turtle* where the basics logo primitives are defined. So, implementing turtle behaviours simply consists in defining logo procedures using the Java programming syntax. For instance the code of the *wiggle* procedure is defined as following:

```
public void wiggle()
{
      fd(1);
      turnRight(random(50));
      turnLeft(random(50));
}
```

Turtles are written in Java and thus it is possible to use all the programming facilities afforded by this high level programming language (inheritance, code reuse, input/output possibilities, code libraries, etc.). So the complexity of a turtle behaviour is not limited by the characteristic of the logo programming language. Moreover the default turtle of TurtleKit is also a *MadKit* agent that can send messages or play a role in an organisation thanks to the MadKit platform characteristics. For instance in the following procedure, the turtle sends a message to all the turtles that play the *infected* role in the *simulation* group.

2.5 Turtle Activation Structure

In TurtleKit every turtle can achieve a collection of actions in a single simulation step. Indeed, an agent's behaviour is defined as an automaton that represents a succession of atomic behaviours which are a list of logo primitives. Thus, when activated, a turtle executes all the primitives associated with an atomic behaviour. Then it returns the name of the atomic behaviour that the turtle wants to do next. The figure 1 illustrates this mechanism with the termite behaviour.

For instance, the *findNewPile* behaviour code is entirely executed when the turtle's turn comes. Then in the next simulation step, the turtle will execute *findNewPile* or *find empty patch.* The code of the *findNewPile* behaviour is defined as following:

2.6 The Environment Model

Although the environment is discretised into a patch grid, turtles move in a continuous way. For instance, a turtle that has a heading of 45 degree and that moves of one unit (*fd(1)*), will have its coordinate, x and y, increased by $\sqrt{2}$. Thus it is possible to make the turtle doing particular moves like a perfect circle for instance. The patch grid is rather used to define spatial areas that define

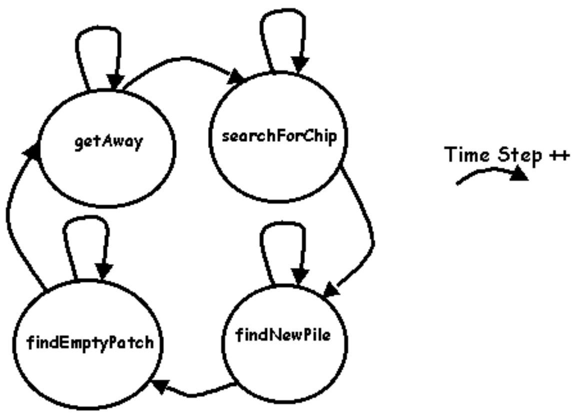

Fig. 1. The termite's behaviour as an automaton

```
public String findNewPile()
{
      if (getPatchColor() == Color.yellow)
          return("findEmptyPatch");
      else
      {
          wiggle();
          return("findNewPile");
      }
}
```

the local environment of turtles. Thus turtles use the patch grid to act on their environment, using commands like *setPatchColor(Color.yellow)* and to localize other turtles. Moreover, the turtles are able to depose an object (any Java object) on a patch. The patch grid also defines a diffusion model used to simulate the diffusion (and/or the evaporation) of information in the environment (see section 4).

3 Platform's default tools

3.1 Observers

As in SWARM, it is possible to probe the model parameters during the simulation. However, in TurtleKit, the agent which is in charge of this job, a subclass of the Observer class, is also a regular MadKit agent and can freely send messages to any agent available within MadKit. For instance, an observer can probe the model variables and then send them to an agent which is specialised in drawing line charts or to another one that handles data backup or analysis. Furthermore the user can easily extend the observer class to develop a particular graphical interface according to the simulation requirements. Additionally an observer can create a probe according to a group/role couple to extract organisational characteristic. For instance, a probe can be defined to spy only the turtle who play the role *infected*. It is also allowed to launch as many observers as required in one simulation and then as many analysis tools (see section 4.3).

3.2 Viewers

To graphically represent the turtle's world, we have implemented an observer specially designed for graphical representations. By default, the platform affords a viewer that displays the world like it is done in StarLogo/NetLogo (patches' colour and turtles' colour and position). However, it is easy to extend the default viewer to define a new point of view of the system: the user can first probe the patch's (or turtle's) state in order to represent it the way he wants. This can be done simply by overriding the *paintPatch* and *paintTurtle* methods of the default viewer. Then, through multiple representations, using several specialised viewers, it is possible to graphically extract many aspects of the simulated model. For instance, TurtleKit affords the opportunity of having a viewer that displays the diffusion of a particular variable in the environment, while another is showing a complex representation of the world (see section 4.2).

3.3 The Scheduler Agent

The *Scheduler* agent has been designed to schedule the activity of the different actors which are present in the simulation model: environment, observers, viewers and turtles. This agent can be ignored when using the platform in its default configuration. However it can be overridden to achieve a particular scheduling. For instance it is possible to activate a particular kind of turtle twice per turn or to activate the display only if a particular condition is verified.

3.4 The Python Command Center

The use of the java language has one shortcoming: the code must be compiled and it cannot be interpreted at runtime. On another hand, the possibility of interacting with the model during its execution is a very desirable feature when exploring complex systems. To propose this feature in TurtleKit, the platform also provides the possibility of using the Python language to implement turtles and/or dynamically interact with them. So, we have implemented a *python command center* which allows to take control of the turtles by typing simple commands within an interpreter window or by loading a predefined script. Notably, this feature allows to dynamically execute compiled java behaviour during a regular simulation to interact with the simulation model (see section 4.4). It is also possible to dynamically create procedures which can be associated with a button like in StarLogo/NetLogo.

4 Application: Multi-level Emergence

4.1 Project Overview

The understanding of emergence is capital in the study of complex systems whether they are biological, physical, informational, software, etc. In artificial ones this phenomenon must be controlled to better design intelligent systems. In nature, emergence exists generally in the form of *multi-level emergent structures* (also called multiple emergence) [4]. It is the production of emergence in a system composed of subsystems which are themselves the product of passed emergences. We have proposed in [1] a Multi-Agent System model of this type of emergence based on Artificial Life approach. It consists in considering agents as live-based forms (architectures and organization). For example, ant societies have been copied to build self-organized systems [3].

Our goal was to provide a framework to study multi-level emergence by modelling a generic reactive Multi-Agent System [1]. This model is based on a classic MAS approach distinguishing environment, interaction and agent model. The environment is a discrete rectangular surface representing a torrus world. The environment acts primarily as an interaction gradient between agents. The management of information is based on a biological approach (as insect pheromones). The agents emit and perceive information called "Interaction Pheromones" which are propagated in the environment via a diffusion model. The agents are reactive entities that interact via emission and perception of Pheromones in the environment. Structuration of emergence is produced by specific interactions between agents. In fact, each agent has a state that modifies the agent's perception and emission abilities and limits interactions between specific agents.

The behaviour of an agent consists in four distinct stages: **perception**, **emission**, **mutation** and **move**. These stages are realised at each turn. Perception and emission consist on a drop of information in environment and on an analysis of information locally perceived. Mutation permits the modification of a state of an agent, depending on perception of pheromones, and favours the structuration of the multi-level emergence. The moving stage consists in an analysis of perception to determine the appropriate movement.

An implementation of such a model means the use of complex tools of simulations. TurtleKit platform has been chosen because it permits to develop this kind of simulation with the facility of Java language. Moreover TurtleKit provides a complete set of tools for extracting and analysing information from simulations.

The theoretical model has been developed by simulating each step of its conception on TurtleKit. To define the multiple emergent MAS, several kinds of agents has been implemented. The first ones has been design to produce an emergence with structuration on the first level only. Then some new agents, developed in new Java Classes, have been mixed with the first ones in simulations to distinguish specific structuration factors. To this end, a scheduler has been used to differentiate the role of dynamic model factors, like information diffusion or agent movement. For instance, some test has been made to produces several pheromones diffusion between to move of agents. It showed us that the inertia of information propagation in the environment is fundamental in the emergence phenomenon. This observation would have been tricky to obtain without a total control of the simulation scheduling. Finally, numerous simulations involving heterogeneous agents, modified scheduling, real-time intervention, etc., permitted us to design the final model.

4.2 Using the TurtleKit Viewers

One of these important tasks for the analysis of the system was to observe and to characterise the emergent phenomenon. To this end we used a lot of viewers with the following representation:

– Agents as function of their states
– Agents depending on their position in emergent structures (for each level of emergence)
– Agents depending on their behaviour stage
– Pheromones of each type
– Attractive and Repulsive zone for each kind of agents
– Empty zone (zone where quantity of information is under the agents sensibility threshold)
– Superposition zone of two or more specific pheromones
– Global view of information

The figure 2 shows a simulation represented according to four different points of view:

1. a view of pheromones emitted by core agents.
2. a view on agent as function of their position in emergent structures.
3. a global view of information.
4. a view of attractive pheromones of structures at level 1.

More specifics observations have been made to determine the pattern of structuration of emergence. Figure 3 shows another viewer which has been configured to represent agents (turtles) in different colours depending on their states and role in the emergent structuration process. In the center of this figure, a core agent is represented in the lightest gray attracting agents in dark gray. These agents attract the agents in black. A circular structure is thus clearly shown with a clear colour semantic. The Viewers permitted to express the role of information too. It is illustrated on these two examples.

The figure 4 shows a Viewers parameterized to represent different pheromones involved in the circular structure formation. In light gray, in the center of the "cell", is represented a repulsive zone for black agents and around the structure, in gray, is represented attractive information for black and dark gray agents.

The figure 5 shows another aspect of the use of viewers. In these two pictures, the amount of information, including each type of pheromones emitted by agents, is represented in white. These Viewers permitted us to analyse the complexity characteristics of the system with an interactionnal and informational

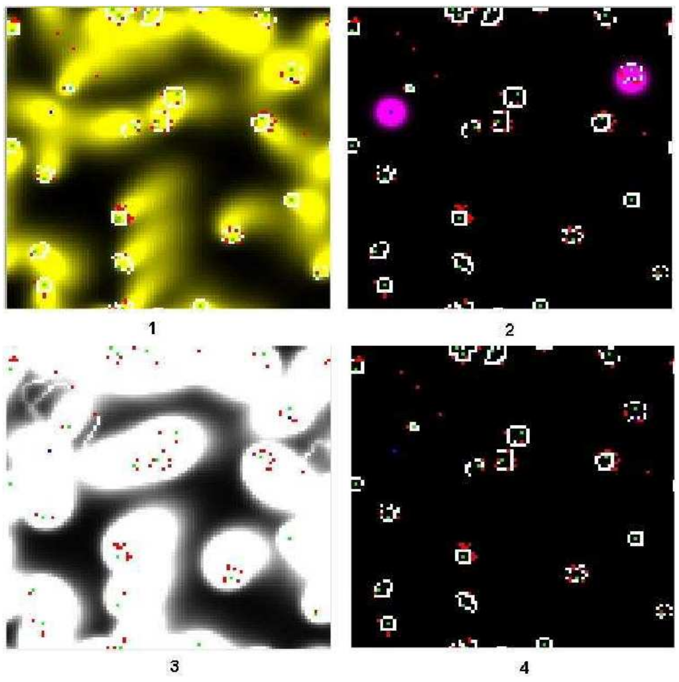

Fig. 2. multiple representations of the world using four different TurtleKit viewers

point of View.

4.3 Using the TurtleKit Observers

However, TurtleKit has been used in a more analytic way to study the system behaviour. To this end, the observer has been used to extract more mathematical information. We used a lot of them with the following tasks:

- to count specific agents (in a particular position, with a particular state or behaviour)
- to measure distances between agents
- to make calculation on amounts of pheromones (average, maxima, standard deviation)
- to compute on zone covered by information

The figure 6 shows a mathematical measure of the structural complexity based on an average of distances between agents involved in the same circular emergent structures. This has been made by implementing a specific observer that made the appropriate measurement. Then, this observer has been configured to send this data to a line charts agent that is supplied in TurtleKit. This kind

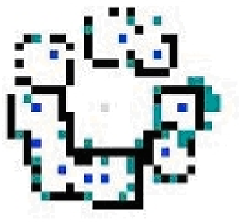

Fig. 3. monitoring the level of agents

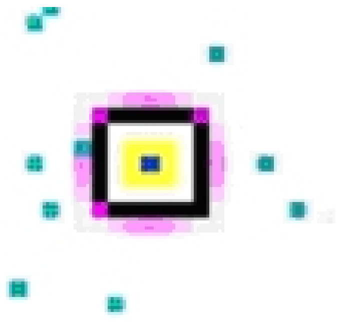

Fig. 4. monitoring circular structure formation

of measurement permitted us to analyse complex behaviour of the implemented model.

4.4 Using the Python Command Center

One of the last analyses was the study of the autopoietic behaviour of the system. For this study it was important to interact with agents during simulations to test the robustness of the system. It has been made by using python command call on Java agent Behaviour. This is illustrated on the figure 7. It represents a measure made by an observer and drawn by a line charts agent. During the simulation, three disturbances have been made using python language and thus interacting directly on the agent to modify their tasks. It permitted to extract precious information on the robustness of the system.

5 Conclusion

In this paper we have presented the TurtleKit platform which relies on the use of a Logo programming approach. This platform aims at providing to advanced users the simplicity of a Logo simulation model while proposing flexibility, modularity and extensibility. Through a concrete application, we have shown, for

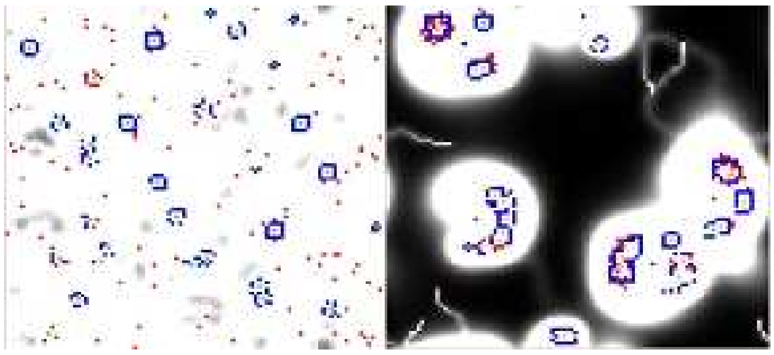

Fig. 5. interactionnal and informational point of view

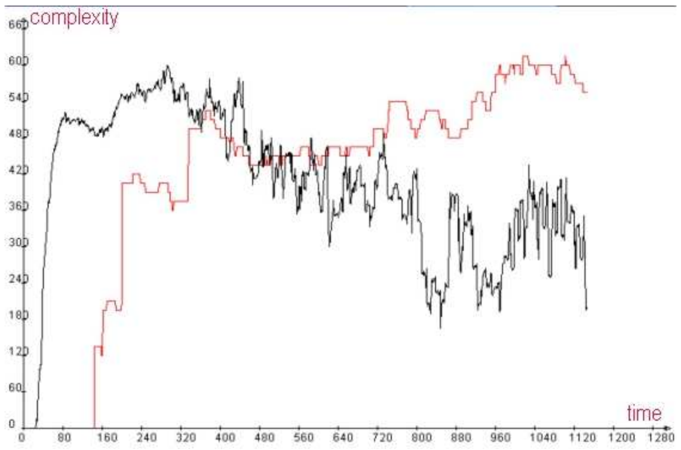

Fig. 6. Measuring structural complexity

instance, the importance of using multiple views of one single simulation in order to extract the required information. Moreover, using one class per agent has facilitated the development process of the *multi-level emergence application* since it is possible to use several different kinds of agent at the same time in one simulation.

The TurtleKit platform and its complete documentation is available for download on the MadKit web site (It is integrated in the MadKit distribution), http://www.madkit.org. It features a user guide, the TurtleKit api, tutorial simulations and advanced projects.

References

1. Gregory Beurier, Olivier Simonin, and Jacques Ferber, *Model and simulation of multi-level emergence*, in the 2^{nd} IEEE International Symposium on Signal Pro-

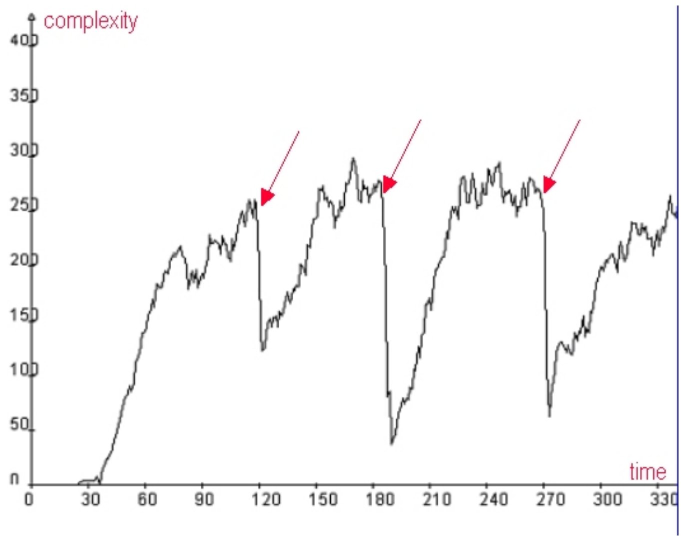

Fig. 7. Analysing the system's autopoietic behaviour using dynamical perturbations

cessing and Information Technology (Marrakesh, Morocco), December 2002.

2. Franois Bousquet, I. Bakam, H. Proton, and Christophe Lepage, *Cormas: common-pool ressources and multi-agent systems*, Lectures Notes in Artificial Intelligence (1997).
3. A. Drougoul, B. Corbara, and D. Fresneau, *Manta: New experimental results on the emergence of (artificial) ant societies*, in Artificial Societies: the computer simulation of social life (N. Gilbert and R. Conte, eds.), UCL press, London, 1995, 1995.
4. F. Heylighen, *Self-organization, emergence and the architecture of complexity*, in the First European Conference on System Science, 1989, pp. 23–32.
5. B. Horling, V. Lesser, , and R. Vincent, *Multi-agent system simulation framework*, In 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, EPFL (Lauzanne, Switzerland), August 2000.
6. P. T. Hraber, T. Jones, and S. Forrest, *The echology of echo*, Artificial Life **3** (1997), no. 3, 165–190.
7. D. MacKenzie, R.C. Arkin, and R. Cameron, *Multiagent mission specification and execution*, Autonomous Robots **4** (1997), no. 1, 29–52.
8. Fabien Michel, Jacques Ferber, and Olivier Gutknecht, *Generic simulation tools based on mas organization*, Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi Agent World MAMAAW'2001 (Annecy, France), 2-4 May 2001.
9. H. Van Dyke Parunak, R. Savit, and R. L. Riolo, *Agent-based modeling vs. equation-based modeling: A case study and users' guide*, Proceedings of the 1st Workshop on Modelling Agent Based Systems, MABS'98 (Paris, France), 1998.
10. *The netlogo system*, http://ccl.northwestern.edu/netlogo/.
11. *The starlogo system*, http://education.mit.edu/starlogo/.
12. http://www.swarm.org/.

13. M.D. Travers, *Programming with agents: New metaphors for thinking about computation*, Ph.D. thesis, Massachusetts Institute of Technology, 1996.