A Multi-environment Multi-agent Simulation Framework for Self-organizing Systems

Maíra Athanázio de Cerqueira Gatti and Carlos José Pereira de Lucena

Departamento de Informática – PUC-Rio,
Rua Marques de São Vicente, 225, 4o andar RDC
Rio de Janeiro, RJ, Brazil
{mgatti,lucena}@inf.puc-rio.br

Abstract. This paper introduces a multi-environment simulation framework for building self-organizing multi-agent systems. From an engineering point of view, the multi-environments approach brings the necessary modularity and separation of concerns to build self-organizing multi-agent systems that address hierarchy, interoperability and multi-aspects problems and domains. Our framework provides higher abstractions and components to support the development of self-organizing systems with multiple environments, which can be situated or not. Furthermore, the framework provides a coordination component and self-organizing mechanisms to be instantiated and flexibility to evolve the framework with more complex ones.

Keywords: Multi-Environment, Multi-Agent Systems, Self-organization, Coordination, Framework.

1 Introduction

Depending on each agent type being developed, the environment types vary. The environment defines its own concepts and their logics and the agents must understand this in order to perceive them and to operate. The environment might be accessible, sensors give access to the complete state of the environment or inaccessible; deterministic, the next state can be determined based on the current state and the action, or nondeterministic, and so on.

Each application domain has its own view of what is an environment and what are the functionalities implemented by an environment. In current approaches, each time a different aspect of the application domain is identified this aspect is then appended to the environment in an ad hoc manner. As a result, the environment centralizes all the different aspects of the targeted application.

In particular, for a situated environment, an additional element characterizes this agent-environment relationship: the localization function is specifically provided by the situated environment. In a situated environment, one can define the location of an agent in terms of coordinates within the environment [4].

During the last years, there has been significant research in the field of self-organization in computer science. Several definitions [1-4] and mechanisms have been examined in order to understand how computing can model self-organizing systems and how self-organizing systems can empower computer science.

G. Di Tosto and H. Van Dyke Parunak (Eds.): MABS 2009, LNAI 5683, pp. 61–72, 2010.

In particular, self-organizing systems - where each component of a system acquires and maintains information about its environment and neighbors without external control and where the emergent system behavior may evolve or change over time [1-4], a self-organizing system has a structurally distributed environment; in other words, at any point in time, no centralized entity has complete knowledge of the state of the environment as a whole. Furthermore, a designer may decide to model environments using various underlying structures. For example, an environment can be modeled as a graph, a discrete grid, a continuous space or a combination of these. In addition, to achieve performance in a cluster or computer grid, or even because of the domain application, the environment can be distributed from a processing perspective if it is designed to be executed in a distributed network. So, the more choices for environment structures, the broader its application in the field of multi-agent simulation systems.

That said, we propose a multi-environment simulation framework for building self-organizing multi-agent systems, called MESOF. From an engineering point of view, the multi-environment approach brings the necessary modularity and separation of concerns to the building of self-organizing multi-agent systems that address hierarchy, interoperability and multi-aspect problems and domains. MESOF provides higher abstractions and components to support the development of self-organizing systems with multiple environments, which can be situated or not.

2 A Multi-environment Framework

The process of building such a self-organizing system with a multi-environment framework that merges several aspects is made clearer at both the design and implementation levels. So, the agents can exist in several and independent environments.

Each environment is concerned only with a specific aspect and can be developed independently from other environments. Existing environments do not need to be redefined or modified. The environment has a dual role as a first-order abstraction: (i) it provides the surrounding conditions for agents to exist [30], which implies that the environment is an essential part of every self-organizing multi-agent system, and (ii) the environment provides an exploitable design abstraction to build multi-agent system applications.

At the conceptual meta-model of the multi-environment multi-agent simulation framework proposed in this work, we have the simulator engine that schedules the main environment. All the agents and sub-environments on the main environment are scheduled by the main environment and added to the simulator engine depending on their states. The environment state is dynamic and if one agent leaves the environment or moves, the environment state changes. Moreover, the environment provides the conditions under which agents exist and it mediates both the interaction among agents and their access to resources.

Moreover, the environment is locally observable to agents and if multiple environments exist, any agent can only exist as at most one instance in each and every environment. In self-organizing systems, the environment acts autonomously with adaptive behavior just like agents and interacts by means of reaction or through the propagation of events.

We classify the events as: (i) emission: signal an asynchronous interaction among agents and their environment. Broadcasting can be performed through emissions; (ii) trigger: signal a change of agent state as a consequence of a perceived event. For instance, an agent can raise a trigger event when perceiving an emission event that changed its state; (iii) movement: signal an agent movement across the environment; (iv) reaction: signal a synchronous interaction among agents, however without explicit receiver. It can be a neighbor of the agent or the environment; and (v) communication: signal a message exchange between agents with explicit receivers (one or more).

Each of these events may be raised by actions performed by agents or by the environment and updates their states. In self-organizing mechanisms [4], the way in which the agents interact is of paramount importance and most of the benefits involved in self-organization are actually due to the right way of interacting. When we talk about interaction and coordination we refer to any kind of mechanisms allowing some agents to orchestrate and influence each other's activities.

Among all the possible interaction mechanisms, MESOF supports uncoupled and anonymous ones. Uncoupled and anonymous interaction can be defined by the fact that the two interaction partners need neither to know each other in advance, nor to be connected at the same time in the network. Uncoupled and anonymous interaction has many advantages. It naturally suits open systems, where the number and the identities of components are not known at design time. It also suits dynamic scenarios in that components can also interact in the presence of network glitches and temporary failures. Moreover, it fosters robustness, in that components are not bound to interact with pre-specified partners but rather interact with whoever is available. Summarizing, uncoupled and anonymous interaction is suited in those dynamic scenarios where an unspecified number of possibly varying agents need to coordinate and self-organize their respective activities. Not surprisingly, this kind of interaction is ubiquitous in nature; cells, insects and animals adopt it to achieve a wide range of self-organizing behaviors [4].

Therefore, the taxonomy created of the events in our framework relies on what and how information is being communicated: explicit or implicit interaction, directly to the receiver, propagation though neighbors, and so on. Moreover, the agent may react in a different way according to the information type.

2.1 The Architecture

In this subsection we describe the architecture, hot spots and the coordination component of a basic and general multi-environment simulation framework for self-organizing systems implemented on top of MASON [5-7]. MASON is a fast, discrete-event multi-agent simulation library core in Java. It contains both a model library and an optional suite of visualization tools in 2D and 3D, which we extended to provide a continuous space and the coordination component regarding the environment approach. For the sake of brevity, we shall neglect a full description of MASON—the interested reader can refer to the official MASON documentation [7]—though some of its main aspects are presented throughout.

The entire core simulation engine provided by MASON was reused and is a frozen spot. MESOF (Fig. 1) provides higher abstractions and components to support the development of self-organizing systems with multiple environments.

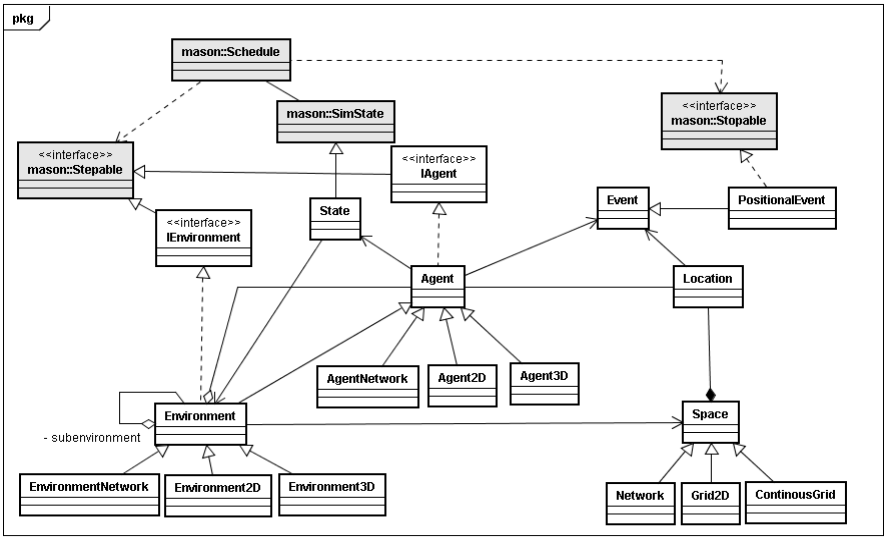

Fig. 1. The Architecture Class Diagram Overview

The first hot spot is the core architecture for a simulation without visualization. There is one interface for agents and one for environments. We provide two classes (*Agent* and *Environment*) that implement those interfaces and they provide a set of reusable behaviors that can be used for any application to be instantiated. There is a main environment, and the agents' interactions are encapsulated by the main environment and by sub-environments which have active behavior and can interact with other agents or other environments. In a discrete event simulation system, an entity is allowed to behave from time to time. These slices of time are called steps. So, a basic entity will usually implement the step method where it will perform its activities. The agent or environment has a set of events, i.e., the information provided for the self-organizing mechanism and implicit coordination, to handle on each time step of simulation. An agent can behave and execute actions on the environment where it resides or on itself.

The Environment manages the schedule of its sub-environment and agents when it is started by the simulator. And, for each time step, it manages the entrance of agents or environments and schedules each new entity. The entities being scheduled can be both executed in all modes provided by the MASON library, i.e., sequential types and parallel sequence.

The second hot spot is the core architecture of the situated multi-environment multi-agent simulation framework for a discrete 2D/ 3D and continuous 3D simulation. For instance, the framework provides the Agent2D and Environment2D classes for situated environment using a discrete 2D double point grid, which is represented

by the class *Grid2D*. This class handles the addition, removal and search of agents and events in a double point location. The environment uses the grid to realize the several strategies for self-organizing mechanisms, such as the atomic ones Replication, Death, Diffusion, and Aggregation [4], or the combined ones as Gradient Fields and Pheromone Path [9], for instance, as it will be further explained in the Coordination Component (next section).

Regarding the 3D environment, the framework provides a 3D continuous space through the *ContinuousGrid* class, and the entities are represented by a triple (x, y, z) of floating-point numbers. All the agent-environment relationships and simulation schedule described for a non-situated environment is reused in these components.

2.1.1 The Coordination Component

Coordination is defined as the management of the communication between agents – coordination defines how agents interact. In self-organizing systems this interaction occurs through the environment (e.g. gradient fields, pheromones) [9], which is why the development of an environment for coordination is important. Another motivation for developing a coordination component that supports more than message passing is the emergence of highly decentralized and dynamic distributed systems. A middleware that supports an environment where agents can interact indirectly through intentional events, for example by leaving objects in an environment for other agents to see, is more scalable and convenient for the application developer [9].

For a non-situated environment, on which the environment manages the agents and events on a graph (for a peer-to-peer network, for instance) the coordination is achieved using a neighborhood in a graph (Figure 2a). Each event to be fired by an agent or by the environment will be located in a node and, if desired, propagated to the neighbors according to the rules specified.

Regarding a situated environment, the coordination is achieved using a neighborhood in 2D/ 3D and discrete/ continuous grid (Figure 2b and 2c). Moreover, there is a specific type of event, called Positional Event, which can be propagated instead of a regular event. The Positional Event has a time to live in the environment. Therefore, if an agent takes too many time steps to reach the source location of the event, it might have disappeared. This is useful for the Diffusion pattern, for instance, and for its combination with other patterns.

MESOF also provides a set of neighborhood lookups for each environment type such as: get agents at a node/ position, get agents within distance, get available nodes/positions within distance, get events at location, and so on. In addition, the environment uses the Template Method and Strategy design patterns [11]. They provide reusable classes and methods that can be implemented for the propagations rules that depend on the self-organizing mechanism increasing reuse. Also, we provide two self-organizing mechanisms in the framework that can be reused. They have already been refactored from the applications developed: the Diffusion and Evaporation mechanisms. We are still adding mechanisms to the framework, such as the Gradient Field, which is the combination of the Evaporation, Aggregation and Diffusion [9].

Another important concept of the framework that allows the coordination component to be flexible and fast is that the grids of the situated environments are sparse fields. Therefore, many objects can be located in the same position and different search strategies exist for each type of entity: sub-environment, agent, event or positional event.

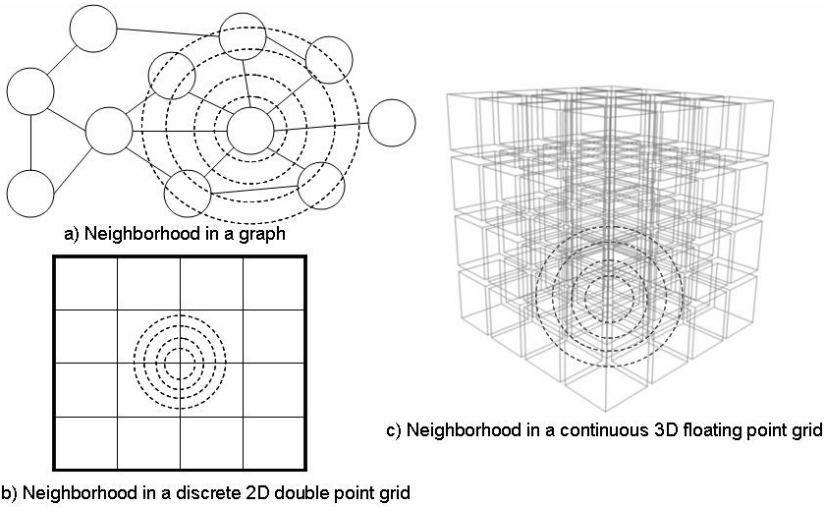

Fig. 2. *a)* Graph: each agent or sub-environment can be located in a node and perceives its neighbors; *b)* 2D double point grid: each agent or sub-environment can be located in a discrete 2D double point position in the grid; *c)* 3D continuous grid: each agent or sub-environment can be located in a 3D floating point grid

3 Framework Evaluation

The design of the multi-environment framework here proposed and its coordination component is the result of many iterative cycles of designing and refactoring. We have been developing self-organizing systems in different domain areas with the goal to develop a novel and suitable software engineering for these types of systems. Our main application areas are: distributed autonomic computing and biological systems.

Autonomic Application Networking. The autonomic application networking provides a platform that aids developing and deploying network applications by providing reusable software components. These components abstract low-level operating and networking details (e.g. I/O and concurrency), and provide network applications with a series of runtime services. We want to achieve two macro properties: scalability and adaptation.

Each application service and middleware platform is modeled as a biological entity, analogous to an individual ant in an ant colony. An application service is designed as an autonomous and distributed software *agent*, which implements a functional service and follows simple biological behaviors such as replication, death, migration and energy exchange. In this way, agents may implement a grid application or Internet data center application on a wired network.

A middleware platform is a *non-situated sub-environment*. It runs on a network host and operates agents (application services). Each platform implements a set of runtime services that agents use to perform their services and behaviors, and follows biological behaviors such as replication, death and energy exchange. Similar to biological entities, agents and platforms in our case study store and expend energy for living. Each

agent gains energy in exchange for rendering its service to other agents or human users, and expends energy to use network and computing resources. Each platform gains energy in exchange for providing resources to agents, and continuously evaporates energy to the network environments. Agents expend more energy more often when receiving more energy from users. Platforms expend more energy more often when receiving more energy from agents. An abundance of stored energy indicates higher demand for the agent/platform; thus the agent/platform may be designed to favor replication in response to higher energy level. A scarcity of stored energy (an indication of lack of demand) may cause the death of the agent/platform.

The exchange energy behavior drives all other behaviors. The exchange energy behavior of the application service AS is *coordinated* with the exchange energy behavior of the platform P: whenever the AS stores or releases energy, P also perceives and stores or releases energy. And whenever one of them is in a higher demand, they fire an *emission type event* that will activate the respective replication behaviors, contributing to a positive feedback loop.

Automated Guided Vehicles. Regarding the well known Automated Guided Vehicles (AGV) case study, an AGV warehouse transportation system uses multiple computer guided vehicles AGVs, which move loads (e.g. packets, materials) in a warehouse. Each AGV can only carry out a limited set of local actions such as move, pick up load, and drop load. The goal is to efficiently transport incoming loads to their destination. The AGV problem is dynamic: many lay-outs, loads arrive at any moment, AGVs move constantly and fail, obstacles and congestion might appear, etc. AGV movement should result in feedback with each other and the environment.

Each station is a *non-situated environment*. Therefore, there are three different types of sub-environments in the main environment, which is a *situated 2D environment* and manages the factory layout where the stations are. The vehicles are *agents* that move from one station to another in the *main environment*.

The dispatching and routing requires a mechanism that enables aggregation and calculation of extra information while flowing through intermediate stations. The *gradient fields pattern* allows the pick up stations to generate gradients while they have loads to be delivered and the intermediate stations also propagate them with information about obstacles and congestions. And the agents follow the gradient.

For instance, at the implementation level, when a pickup station (PS) executes the *Pick Up Behavior* and is dispatching loads, it fires the *load_gradient emission event*, which is *propagated* in the environment. The PS station remains in the dispatching load state while a vehicle does not pick up the load. On the other hand, a vehicle is following the *load_gradient event* propagated by the environment. When it finds a different load gradient, it decides which one to follow according to distance and to avoid obstacles. Once the vehicle chooses one load, it follows the gradient through the intermediate stations until reaching the pick up station. When the vehicle is at the pick up station, it picks up the load to be routed to one drop off station and fires the *AGV_pick_up_load* reaction event. The PS station thus reacts to this event looking for more loads to dispatch. This feedback loop is executed while there are loads to be dispatched.

Stem Cells. In developmental biology, cellular differentiation is the process by which a less specialized cell becomes a more specialized cell type [12],[13],[14]. The stem

cells can be specialized into several kinds of cells, such as heart cells, skin cells or nerve cells. We are interested in the case of a stem cell that differentiates into a mature neuron.

In the computational model, there are three kinds of cells: multi-potent cells are cells with a high power of differentiation that can give rise to several other cell types; neuron progenitor cells are cells able to self-differentiate into a neuron; and non-neuron progenitor cells are cells able to self-differentiate into any kind of cell but neuron's types.

The simulator presents the macro scale to the users by means of a visualization area (3D) [15] that represents the niche where the cells evolve in their life-cycles. In another scale, each phase of cell life-cycle has a 3D graphical representation (Figure 2), presenting the state of the main entities involved in the process. These graphical representations, besides presenting a phase of the life-cycle, show the capacity of differentiation of the cell by means of colors.

The niche is the *main environment* and manages and regulates a *3D continuous space*. Each protein and intracellular entity is a *3D agent*. Each cell is a *3D sub-environment* and contains the intracellular entities. One novel behavior in this computational model is how the cells perceive the environment, i.e., the neighbor's cells, proteins, etc. If the protein is in a specific radius scope, then the cells attract the protein to its surface and the protein is bound to the cell until the specific regulatory network inside the cell is activated. Once it is activated, the protein leaves the cell and becomes inactive. Only active proteins can be bound. The intracellular entities must perceive each other and, if the protein is inactive, i.e., not bound to any other substance or inactive or truncated, then it can bind. The rules for binding vary according to the physicochemical attraction reactions specified for each entity.

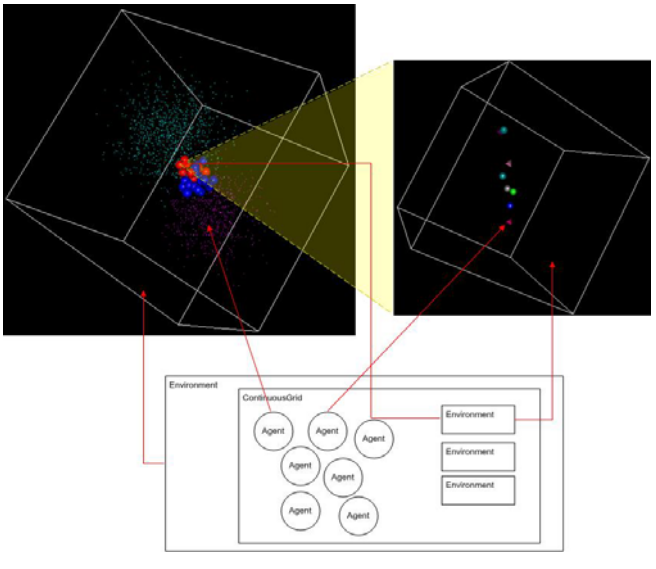

Fig. 3. The Stem Cell multi-environment case study. Each cell is a 3D situated sub-environment with entities (agents) running on it.

For instance, when *Cyclin D* – which is an intracellular entity – is inactive, if it perceives the *event synthesize_cyclin_D*, it executes the action *activate_cyclin_D* as a reaction. Then it changes its state to active, and fires an event *cyclin_D_synthesized* to the environment. Once the *CDK2* is active, it perceives the last event and changes its state to *BINDED TO CYCLIN*. The same happens to the Cyclin. Once bound, they enable the activation of the CDK-Cyclin complex behavior that regulates the cell cycle.

In order to model the 3D spatial self-organization [15] of the stem cells, we must find an empty space among its neighbors that minimizes the cell effort. Pushing the smallest amount of neighbor cells minimizes the cell effort. Thus, it can be accomplished by choosing the nearest empty space in its neighborhood followed by the pushing behavior. The problem was reduced to the 3D Nearest Neighbor problem [17], which is well studied in the field of Computational Geometry combined with the strategy similar to a Ray Tracing technique [16].

Framework Performance

Regarding the stem cell domain – which is the most complex case study - in order to achieve an increase in order, thousand of agents have to be running at the same time. A computer with the following features was used: Intel(R) Core (TM) 2 CPU T5200@ 1.60 GHz, 2GB of RAM. We executed the simulation in a 3D grid with dimensions 50×50×50 that allows visualizations of up to 100,000 cells and 500,000 extra cellular proteins. For the case for the cell visualization, we need less than 100 entities.

Our current solution does not take advantage of a parallel computation environment, although we are working on the problem [18],[19]. Even though the number of cells running together and the time of execution in the simulation for a single computer were satisfactory, we need to increase this number and to achieve this goal we are distributing the framework and application in a cluster architecture with eight QUAD CORE Q9300 processors.

4 Related Work

There are two categories for environment related work: those that emphasize the agent component and downplay the environment [20-21],[23-25], and those that consider the environment an important component of the system and decouple it from the agents. The former does not fully value an environment as a first order entity. For instance, in Jade [21], the agent environment is reduced to a mechanism for passing messages around to other agents, such as matchmakers and brokers. This is also the norm in the FIPA standard [22].

Several researchers have shown that coordination through an environment has interesting properties: Mamei et al. [26] provide an environment that allows agents to coordinate their movements in a mobile network; Parunak [27] describes several optimization algorithms for which an environment is needed; Brueckner [28] has developed an environment for the application of ant algorithms in manufacturing control; coordination of parallel processes (agents) through tuplespaces [29] can be seen as early (and ongoing) work to provide an environment wherein agents can interact.

A recent work [30], [31] evaluated five tools that acknowledge the importance of the environment in a multi-agent-based simulation system. These are NetLogo [32], MASON [5-7], Ascape [33-36], RePastS [37], and DIVAs [39].

5 Conclusions and Future Work

The design of the multi-environment framework here proposed and its coordination component is the result of many iterative cycles of designing and refactoring. As a result we think that the framework is both easy to learn and expressive at the same time. Regarding a situated environment, the coordination is achieved using a neighborhood in 2D/ 3D and a discrete/continuous grid. Another main contribution of the framework consists of providing reusable self-organizing mechanisms to be instantiated and the flexibility to evolve the framework with more complex ones. To date, the literature does not present any architectural self-organizing pattern reuse at implementation level.

There are also two features currently being developed in the framework: an autonomic experimental verification which uses online planners [40], and a transparent middleware for parallelization and distribution in a cluster using a virtual space [18-19]. The experimental autonomic verification method may autonomously analyze the emergent behavior and would be used to eventually refine the models as design feedback.

Acknowledgments. This work was supported by MCT/CNPq through the "*Grandes Desafios da Computação no Brasil: 2006-2016*" (Main Computational Challenges in Brazil: 2006-2016) Project (Proc. CNPq 550865/2007-1).

References

1. Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A.: Self-organization in multi-agent systems. The Knowledge Engineering Review 20(2), 165–189 (2005)
2. Visser, A., Pavlin, G., van Gosliga, S.P., Maris, M.: Self-organization of multi-agent systems. In: Proc. of the International workshop Military Applications of Agent Technology in ICT and Robotics, The Hague, the Netherlands (November 23-24, 2004)
3. Di Marzo Serugendo, G., Fitzgerald, J.S., Romanovsky, A., Guelfi, N.: Generic Framework for the Engineering of Self-Adaptive and Self-Organising Systems. CS-TR-1018 (2007)
4. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. J. Syst. Archit. 52(8), 443–460 (2006)
5. MASON George Mason University, `http://cs.gmu.edu/_eclab/projects/mason/`
6. MASON documentation, `http://cs.gmu.edu/~eclab/projects/mason/docs/#docs`
7. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: MASON A New Multi-Agent Simulation Toolkit, Department of Computer Science and Center for Social Complexity. In: Proceedings of SwarmFest, Michigan, USA (2004)

8. Gardelli, L., Viroli, M., Omicini, A.: Design Patterns for Self-Organizing Multiagent Systems. In: 2nd Int. Workshop on Eng. Emergence in Decentralised Autonomic Systems (EEDAS 2007), To be held at the 4th IEEE Int. Conf. on Autonomic Computing (ICAC 2007), Jacksonville, Florida, USA, June 11 (2007)
9. De Wolf, T.: Analysing and engineering self-organising emergent applications, Ph.D. Thesis, Department of Computer Science, K.U. Leuven, Leuven, Belgium, p. 183 (May 2007)
10. Weyns, D., Boucké, N., Holvoet, T.: A field-based versus a protocol-based approach for adaptive task assignment. AAMAS 17(2), 288–319 (2008)
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
12. Loeffler, M., Grossmann, B.: J. Theor. Biol. 150(2), 175–191 (1991)
13. Loeffler, M., Roeder, I.: Cells Tissues Organs 171(1), 8–26 (2002)
14. Lord, B.I.: Stem cells, pp. 401–422. Cambridge Academic Press, London (1997)
15. Faustino, G.M., Gatti, M.A.C., Bispo, D., de Lucena, C.J.P.: A 3D Multi-Scale Agent-based Stem Cell Self-Organization. In: SEAS 2008 - Fourth Workshop on Software Engineering for Agent-Oriented Systems, Capinas. XXV SBES (2008)
16. Glassner, A.: An Introduction to Ray Tracing. Academic Press, London (1989)
17. Smid, M.: Closest-Point Problems in Computational Geometry. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, ch. 20, pp. 877–935. North-Holland, Amsterdam (2000)
18. Motta, P., de Gatti, M.A.C., de Lucena, C.J.P.: Towards a Transparent Middle\ware for Self-Organizing Multi-Agent Systems on Clusters. In: The Third International Workshop on Massively Multi-Agent Systems: Models, Methods and Tools (MMAS 2009) at AAMAS 2009 (2009)
19. Valeriano, A., Motta, P., Gatti, M., Lucena, C.: Requisitos Funcionais para um Midleware Paralelo e Distribuído de Sistemas Multi-Agentes Auto-Organizáveis. Monografias em Ciência da Computação, DI, PUC-Rio (2009)
20. Giunchiglia, F., et al.: The tropos software methodology: Processes, models and diagrams. Technical Report Techincal Report No. 0111-20, ICT - IRST (2001)
21. Bellifemine, F., Poggi, A., Rimassa, G.: Jade, A FIPA-compliant Agent Framework. In: Proceedings of PAAM 1999, London, UK (1999)
22. FIPA: Foundation for intelligent physical agents, `http://www.fipa.org/`
23. DECAF, `http://www.cis.udel.edu/~decaf/`
24. Graham, J., Windley, V., McHugh, D., McGeary, F., Cleaver, D., Decker, K.: Tools for Developing and Monitoring Agents in Distributed Multi Agent Systems. In: Workshop on Agents in Industry at the Fourth International Conference on Autonomous Agents, Barcelona, Spain (June 2000)
25. JACK: Documentation, `http://www.agentsoftware.com/products/jack/documentation_and_instructi/jack_documentation.html`
26. Mamei, M., Zambonelli, F.: Self-maintained distributed tuples for field-based coordination in dynamic networks. In: The 19th Symposium on Applied Computing, SAC 2004 (2004)
27. Parunak, V.: Go to the Ant: Engineering principles from natural multi-agent systems. Annals of Operations Research 75, 69–101 (1997)
28. Brueckner, S.: Return from the Ant - Synthetic Ecosystems for Manufacturing Control. PhD thesis, Humboldt University Berlin (2000)
29. Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.): Coordination of Internet Agents: Models, Technologies, and Applications. Springer, Heidelberg (2001)

30. Arunachalam, S., Zalila-Wenkstern, R., Steiner, R.: Environment Mediated Multi-Agent Simulation Tools: A Comparison. In: Proc. of IEEE Workshop on Environment-Mediated Coordination in Self-Organizing and Self-Adaptive Systems, Venice, Italy, October 20-24 (2008)
31. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent based Simulation Platforms: Review and Development Recommendations. Simulation 82 (2006)
32. Wilensky, U.: NetLogo for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL, http://ccl.northwestern.edu/netlogo/Center
33. Ascape, http://ascape.sourceforge.net/index.html
34. Ascape documentation, http://ascape.sourceforge.net/index.html/#Documentation
35. Parker, M.T.: What is Ascape and Why Should You Care. Journal of Artificial Societies and Social Simulation 4(1) (January 2001)
36. Inchiosa, M.E., Parker, M.T.: Overcoming design and development challenges in agent-based modeling using ASCAPE. Proceedings of National Academy of Sciences (PNAS) of United States of America 99 (May 2002)
37. RePastS, http://repast.sourceforge.net/
38. North, M.J., Tatara, E., Collier, N.T., Ozik, J.: Visual Agent-based Model Development with Repast Simphony. In: Proc. of the Agent 2007 Conf. on Complex Interaction and Social Emergence, Argonne National Laboratory, Argonne, IL USA (November 2007)
39. Mili, R.Z., Steiner, R., Oladimeji, E.: DIVAs: Illustrating an Abstract Architecture for Agent-Environment Simulation Systems. Multi agent and Grid Systems, Special Issue on Agent-oriented Software Development Methodologies 2(4) (2006)
40. Soares, B.C.B.A., Gatti, M.A.C., Lucena, C.J.P.: Towards Verifying and Optimizing Self-Organizing Systems through an Autonomic Convergence Method. In: The Fourth Workshop on Software Engineering for Agent-Oriented Systems, Capinas, XXV SBES (2008)