MCMAS: A Toolkit to Benefit from Many-Core Architecure in Agent-Based Simulation

Guillaume Laville[1], Kamel Mazouzi[2], Christophe Lang[1], Nicolas Marilleau[3], Bénédicte Herrmann[1], and Laurent Philippe[1]

[1] FEMTO-ST Institute,
CNRS / Université de Franche-Comté, France
[2] Mésocentre de calcul de Franche-Comté,
Université de Franche-Comté, France
[3] UMI 209 - UMMISCO,
Institut de Recherche pour le Développement (IRD),
UPMC, France

Abstract. Multi-agent models and simulations are used to describe complex systems in domains such as biological, geographical or ecological sciences. The increasing model complexity results in a growing need for computing resources and motivates the use of new architectures such as multi-cores and many-cores. Using them efficiently however remains a challenge in many models as it requires adaptations tailored to each program, using low-level code and libraries. In this paper we present MCMAS a generic toolkit allowing an efficient use of many-core architectures through already defined data structures and kernels. This toolkit promotes few famous algorithms (diffusion, path-finding, population dynamics) which are ready to be used in an Agent Based Model. For other needs, MCMAS is based on a flexible architecture and can easily be enriched by new algorithms thanks to development features. The use of the library is illustrated with two models and their performance analysis.

Keywords: Multi-Agent Systems, Parallel Computing, GPGPU.

1 Introduction

Multi-agent systems (MAS) are often used to describe large complex systems as biological, geographical or ecological ones. Increasing the size or precision of agent based simulations is a common way to obtain more accurate results, but it also results in increased computation costs. Since the raise of CPU computation power is not sufficient to absorb these expanding modeling needs, parallel architectures are now becoming a required mean to gain performance. However, their use requires fundamental improvement in model runtimes provided by platforms such as NetLogo [1] or GAMA [2]. Several projects as D-MASON [3] or Repast-HPC [4] have introduced distributed computing techniques into MAS in order to make use of parallel CPU architectures. The goal of these works is to accelerate or enlarge the simulations.

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 544–554, 2014.

Last years have seen the emergence of GPU or many-core cards based on massively parallel SIMD (Single Instruction Multiple Data) architectures. GPGPU computing is already used in various domains as linear solvers, video streaming or image processing. Most of these applications are based on matrix data structures well-adapted to parallel processing. Since most of personal computers are equipped with a GPU card, taking advantage of these GPU and many-core architectures has become one way of speeding up Agent Based Simulations. MAS are however often characterized by less-regular data accesses and unpredictable behaviors due to algorithms offering multiple execution branches or random aspects. They thus do not fit the single instruction flow model and adaptations in their execution workflow are thus required to allow agents to run concurrently.

Some works have already demonstrated the gain of using GPU to run MAS in various domains as cellular automaton [5], mobile agent path finding [6], genetic processes [7] or life science [8]. These works present specific adaptations of existing MAS to GPU. In these models individual behaviors driven by mathematical laws (path finding) or equations (cellular automatons) can be considered as the application of the same process on each individual (Single Program, Multiple Data or SPMD execution). This approach does not however work for the majority of MAS, and algorithmic adaptations are often required. Note that the full-GPU approach sometime limits the possible use in MAS and that an hybrid approach, based on CPU plus GPU, may fit a larger number of MAS.

The FLAME-GPU [9] platform proposes an all-in-one solution to run MAS on GPU. The framework relies on a detailed XML description of the agents and on C-like code fragments to support several target architectures. This approach implies that the MAS is developed for the framework and thus cannot (re)use an existing model nor interface with other MAS runtimes.

The contribution of the paper is a toolkit, called MCMAS (Many Core MAS), that provides functions to facilitate the implementation of MAS simulations on many-cores architectures and better exploit their computing power. The library can be interfaced with existing agent platforms to acts as a wrapper for GPU code that allows integration of optimized model parts within other simulators. We present in section 2 the MCMAS library. Its interface and extension facilities are illustrated on two use cases: the collembola model and the mior model. The model performance results are detailed in section 3. We then conclude on the possibilities of the MCMAS platform and its possible application to other models or use cases in section 4.

2 MCMAS

Porting a MAS on GPU requires a model adaptation to benefit from this architecture. We developed the MCMAS library to facilitate the implementation of models on GPU and many-core architectures. To achieve this MCMAS proposes a set of commonly used functions and data structures that simplify the implementation of new models and allows the integration of new functionalities in the shape of plugins. Note that MSMAS has yet only be tested on GPU and on few

CPU cores but it should also match many-core architectures as Intel Xeon Phi as it is based on a SIMD execution model.

2.1 Basic MCMAS

Choosing a programming language is the first step to adapt an agent model to a GPU platform. On the one hand the Java programming language is often used for the implementation of MAS due to its large availability and its high-level object-oriented programming. On the other hand GPU platforms only offer dedicated languages, CUDA or OpenCL, so that a model implementation requires some part of GPU-specific development or the use of GPU enabled libraries. MCMAS offers a higher level Java interface linked with OpenCL GPU-code through the JOCL library [10] (see figure 1). We decided to retain OpenCL for its portability and the possibility to run programs on both CPU and non-GPU many-core architectures such as the Xeon Phi.

OpenCL provides access to CPU or GPU threads using an asynchronous interface. This library is based around three main concepts. The *kernel* represents a program to be executed on the GPU. The *work-item* is analogous to the concept of thread on CPU. The *work-group* is a set of work-items that share memory. An OpenCL execution consists in running the same kernel on numerous work-items. Synchronization operations, as barriers, can only be used across the same work-group. Data used by the work-items can be stored in local (high speed) or global (low speed) GPU memory. Since the size of this local memory is often limited to a few hundred of kilobytes, choosing this number often implies a compromise between the model synchronization or data requirements and the available resources. In the case of agent based simulations, each agent can thus naturally be mapped to a work-item. Work-groups can then be used to represent groups of agents or simulations sharing common data (as the environment) or algorithms (as the background evolution process). This process is described in more details in the case of the MIOR model[11].

Similar data structures are used by whole classes of MAS. One such example is the grid, which can be either integrated in the algorithm, as in SugarScape [5] where each cell represents the fundamental unit of modeling, or used to discretize a continuous environment as path-finding simulations [12]. These grids can be considered as 2d or 3d matrices representing agents' data or their environment. Another data representation often encountered in MAS is the usage of coordinate systems to position agents in the simulation space [13].

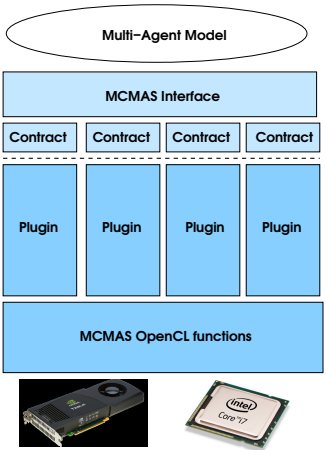

Fig. 1. MCMAS general architecture

These common structures lead to the usage of similar algorithms in many simulations, such as distance computation in 2d or 3d space, diffusion processes, reduction operations, SIMD transformations applied to each cell of the model. These kinds of processing can be parallelized and executed on the GPU, leading to possible performance gains, without heavy modifications to the model scheme.

To facilitate the addition of new functions, MCMAS is based on a dynamic architecture allowing the registration of new plugins at runtime. This allows two usage approaches: (i) the adaptation of the whole MAS model on GPU by writing a new implementation of the algorithm, a **model parallelization**, or (ii) the usage of many-core optimized primitives in an existing MAS model, a **process parallelization**. This dual extensible architecture allows the designer to either use already developed plugins or to roll its own solutions, based on the parallelized operations required by its model.

The MCMAS interface is based on the `MCMASContext` object. This context contains all the data (for instance device context and execution queue) required to run any MCMAS low-level operation and plugins. The context can be created using a wide variety of constructors, to allow the customization of the environment depending of the available execution resources and the functionalities needed: profiling, debugging... This MCMAS context can then be either used to initialize MCMAS plugins using the `newInstance()` method, or to directly call low-level OpenCL operations, using the accessors provided for the underlying OpenCL objects.

MCMAS provides a standard set of data structures (grids, vectors, structures) as a mean to pass data to plugin methods. Methods are provided to facilitate the translation of existing Java data structures to these formats. Each plugin is also free to define its own data structures, either for its own usage or for general use. A springy data architecture has been designed to expand the existing MCMAS collection.

2.2 Process Parallelization Use Case

The Collembola model is an agent based model focusing on landscape biodiversity. It reproduces the diffusion of arthropodes life forms (collembola) across plots of an identified territory of the Morvan. This environment is constituted of forested, cultivated and artificial areas. The model goal is to study the impact of modifications of this environment on biodiversity. The model subdivides each plot in surface units and follows the evolution of the number of individuals in time. This evolution can be decomposed into four steps: (i) *arrival* of new individuals, (ii) *reproduction* of each parcel population, (iii) *diffusion* between cells, and (iv) *selection* of surviving individuals. The algorithm contains two costly operations which can be parallelized: diffusion, reduction of cell properties to obtain global data. The **reproduction** process represents an update of each plot population to take into account the new individuals. It consists of a walk-through on all the cells to update their population and is implemented using the reduction plugin of MCMAS. The **diffusion** process represents the collembola propagation on new plots. For each cell the population is computed taking into account the immigration flow from its

eight neighbors. The process is implemented using the diffusion plugin. The two costly operations use already existing MCMAS plugins to speed up their computations, with few changes in the simulation and without advanced parallelization expertise, as illustrated on figure 2.

```
1 public void run() {
2     MCMASContext context = new MCMASContext();
3     ReductionPlugin reduce = ReductionPlugin.newInstance(context);
4     DiffusionPlugin diffuse = DiffusionPlugin.newInstance(context);
5     doNewArrivals();  // Simulate new individuals' arrival on CPU
6     reduce.taggedIntReduce(...); // Launch reproduction on GPU
7     diffuse.diffuse8(...); // Launch diffusion on GPU
8     doDeath(); // Kill individuals on a not favorable terrain on CPU
9 }
```

Fig. 2. Usage of MCMAS in the Collembola model

In both cases the GPU is used as a specialized co-processor for the existing simulation. This permits a progressive adaptation of the existing Java model to benefit from many-core architectures.

2.3 Developing a New MCMAS Plugin

A MCMAS plugin is a Java class who implements the `MCMASPlugin` interface. This interface allows plugins to be instantiated from a MCMAS context. This context will then be used by the plugin to allocate new data structures, launch operations or adapt its execution.

Beyond the basic methods required by the interface, each plugin can provide its own free-form set of operations. New MCMAS plugins and structures are packaged as Java libraries. By convention all classes belonging to the same plugin lives in a `mcmas.plugins.<plugin name>` package to maintain code isolation and facilitate the discovery of new plugins.

Most plugin-provided operations are organized around the standard GPU execution workflow: (1) OpenCL source code retrieval and compilation, (2) copy of Java data into input data structures, (3) execution of one or more kernel, (4) retrieval of output data and translation into Java data structures, and (5) resource cleanup and return from the primitive call. Memory allocations in each plugin can be managed at two levels. At instance level the memory is used for the lifetime of the plugin. At method level the memory is used for temporary copies of input and output parameters and to manage the execution progress.

2.4 Model Parallelization Use Case

The MIOR (MIcro-ORganism) [14] model simulates local interactions in a soil between microbial colonies and organic matters. The MIOR model can be used in multi-scale MAS, such as Sworm [15]. Since the evolution takes place at a microscopic scale each unit of soil corresponds to many such simulations that justify the computing cost of this process.

```
1  // Create a new MIOR model template
2  MiorWorld model = new MiorModel();
3  model.nbOM = 310; model.nbMM = 38; model.size = 200;
4  MCMASContext context = new MCMASContext(MCMASContext.GPU);
5  MiorPlugin plugin = MiorPlugin.newInstance(context);
6  // Execute 100 instances simulating 1000 steps each time.
7  int [][] CO2Values = new int[100][1000];
8  plugin.runNSimulations(model, 100, CO2Values, 1000);
```

Fig. 3. Usage of the MIOR plugin from Java code

The MIOR model is based on two types of agents: (i) the Meta-Mior (MM), microbial colonies consuming carbon and (ii) the Organic Matter (OM), carbon deposits occurring in soil. Each Meta-Mior agents exhibits two distinct behaviors. By *breathing* it converts mineral carbon from the soil to carbon dioxide CO_2, released in the soil. By *Growing* it fixes the carbon present in the environment to reproduce itself (augments its size). This last action is only possible if the colony breathing needs are covered, i.e. if enough mineral carbon is available. The model is implemented as an MCMAS plugin as described on Figure 4, to allow an easy use in a Java program (Figure 3).

```
1  public class MiorPlugin extends MCMASPlugin<MiorPlugin> {
2    // Static method implemented by all MCMAS plugins (factory)
3    public static MiorPlugin getInstance(MCMASContext context) {
4      new MiorPlugin(context.getContext(), context.getQueue();
5    }
6    private MiorPlugin(Context context, CommandQueue queue) {
7    }
8    public void runNSimulations(...) {
9    }
10 }
```

Fig. 4. Implementation of the MIOR plugin

2.5 Using MCMAS from Existing MAS Frameworks

The MCMAS library can also be used to delegate computations in existing MAS frameworks. An intermediary agent is required to translate the existing models requests into calls to MCMAS plugins, and to manage the interactions with the MCMAS platform as shown on figure 5. This translation layer between MCMAS and the MAS framework allows a transparent use of the

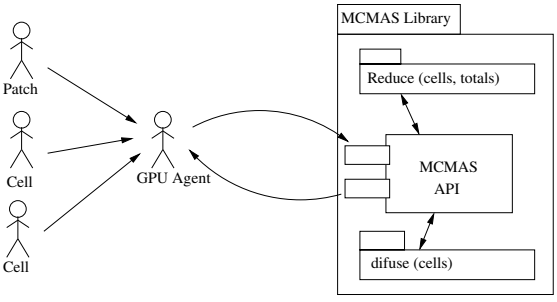

Fig. 5. MCMAS integration in MAS framework

library without disturbing the existing model architecture. For instance, in the case of GAMA, this integration can be realized by extending the agent description language with MCMAS related functions. For that, an eclipse plugin associated with GAMA and XText should be developed. In the case of the Madkit framework [16], a threaded agent should be implemented. The latter may ordered by other simulation agents through a dedicated agent communication language.

3 Experiments

The performance of the use case models is presented in this section to show the gain that can be obtained by delegating some functions or by implementing a plugin that use the MCMAS library. Note that the GPU runs are compared to the Java implementation of the model instead of their initial Netlogo implementation as it would not be fair to compare programs to interpreted code (Netlogo implementations are much more slower).

3.1 Collembola Model Results

In this part we present the performance of the Collembola model with the MCMAS library on CPU and GPU execution platforms. Due to the cost of data transfers between CPU and GPU, performance gains depend on the size of the model. Therefore we have run the simulations with different scaling factors for the agent population size. The reference execution is a sequential execution of the simulation on a standard processor, an Intel Core i7 2600K running at 3.4GHz. Since MCMAS also target multi-core platforms, using CPU-based OpenCL implementations, we execute the simulations in parallel on the same processor to point out the gain of what can be expected from CPU parallelism. Then to illustrate the performance on GPU hardware we have run the simulations on two mainstream GPU platforms: a Radeon HD 6870 and a GeForce 560Ti.

Figure 6 shows the results obtained when running the simulations. As expected the use of the MCMAS library increases the performance of the simulation. The best obtained speedup is factor of 4 between the sequential run and the GeForce run. The Radeon curve is limited to a scaling factor of 20 due to buffer size limitations imposed by the AMD-provided OpenCL runtime.

Note that the curves exhibit a odd-even pattern. Since this phenomenon is visible on distinct hardware, drivers and OpenCL implementations, it is likely due to the model decomposition process based on warp of fixed power-of-two sizes. The results show that the gain closely depends on the hardware platform.

3.2 MIOR Results

Two platforms are used to assess the performance of the complete GPU implementation of the MIOR model as a MCMAS plugin. The first one is representative of hardware available in HPC clusters: a cluster node dedicated to GPU computations with two quad-cores Intel X5550 processors running at 2.67GHz

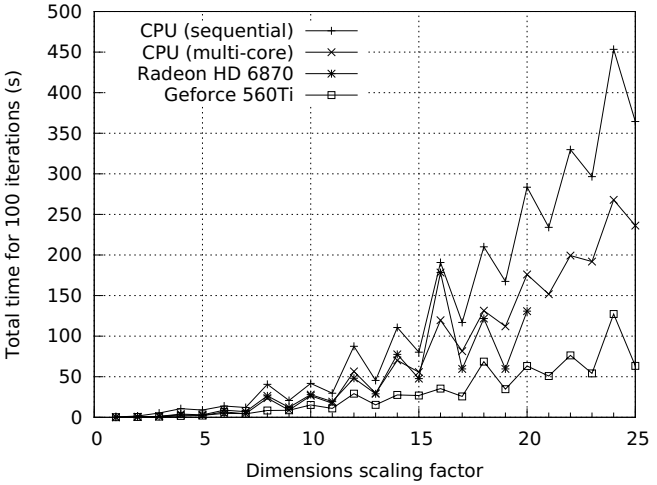

Fig. 6. Performance of the Collembola model

and a Tesla C1060 GPU device running at 1.3GHz and composed of 240 cores. The second platform is a personal computer based on an Intel Q9300 CPU, running at 2.5GHz, and a Geforce 8800GT GPU card, composed of 112 cores and running at 1.5GHz. The purpose of these two platforms is to assess the benefit that could be obtained either when a MAS modeler has access to specialized hardware as a cluster or tries to take benefit from its own personal computer.

The MIOR experiments illustrate the effect of increasing the level of adaptation of the algorithm to GPU hardware. The **GPU (naive)** implementation is a direct implementation of the existing algorithm and its data structures. The **GPU (optimized)** implementation uses compact representations of the topology and the low level MCMAS interface.

We measure the execution duration for 50 simulations on the two hardware platforms. A size factor is applied to the problem: at scale 1, the model contains 38 MM and 310 OM, while at the scale 6 these numbers are multiplied by six. The size of the environment is modified accordingly to maintain the same average agent density in the model. This scaling factor displays the impact of the size of the simulation on performance.

Figures 7 and 8 gives the performance of the simulation runs on the two platforms (the CPU curves give the performance of the local CPU). We can note that for small problems execution time for all implementations are very close. This is because the GPU implementation does not have enough threads (representing agents) for an optimal usage of GPU resources. This trend changes after scale 5 where the optimized GPU version begins to take advantage of the naive GPU and CPU implementations. This advantage continues to grow with the scaling factor and reaches a speedup of 10 at the scale 10 between the fastest single-simulation GPU implementation and the naive one.

Multiple trends can be observed on both figures. First the optimizations for the GPU hardware show a big, positive impact on performance, illustrating

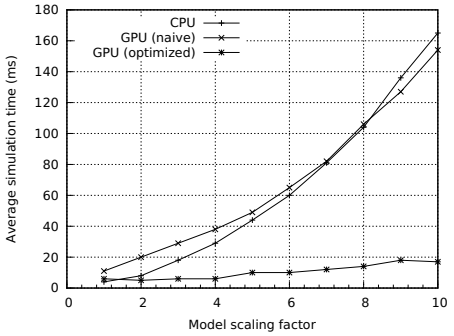

Fig. 7. MIOR performance on Tesla

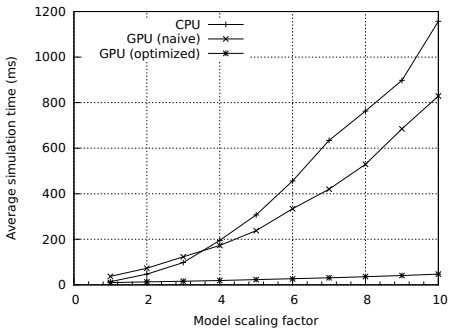

Fig. 8. MIOR performance on mainstream GPU

the strong requirements on the algorithm properties to ensure the execution efficiency. These charts also show that despite the vast difference in number of cores between the two GPU platforms the same trends can be observed in both cases. We can therefore expect similar results on other GPU cards, without the need for more adaptations.

4 Conclusion

In this article we present MCMAS, a solution to facilitate the use of many-core architectures and allow the integration of optimized model parts within agent based simulators. To achieve this, two possible approaches are supported by our toolkit: (i) use MCMAS as an optimized algorithm library; or (ii) use MCMAS as many-core runtime to develop specific algorithms or MAS. The usage can be mixed by the model designer, depending on the its model needs and of the amount of development required.

The first approach is to use the interfaces and plugins already provided by MCMAS. These plugins cover classic problems in MAS simulations as path-finding, diffusion or population dynamics. These predefined algorithms are ready to be used to accelerate one or more aspects of an existing CPU simulation, without huge changes in the existing implementation.

The second approach is to develop new plugins for MCMAS to enable the implementation of more specialized or performance-critical algorithms directly on the underlying many-cores platform. This approach implies the development of OpenCL kernels, called from MCMAS, to execute portions of the computations. Once written, these kernels can be used both on CPU, GPU or any other OpenCL supported platform: this allows the reuse of the same program on a wide range of architectures, ranging from personal computers to computing clusters, or dedicated GPU nodes, without modification or manual re-compilation.

Our main goal is now to enrich the MCMAS platform to support more MAS problems and to refine the support of new data structures to generalize the possible applications of this platform.

Acknowledgement. Computations presented in this article were realized on the supercomputing facilities provided by the **Mésocentre de calcul de Franche-Comté**.

References

1. Sklar, E.: Netlogo, a multi-agent simulation environment. Artificial Life 13(3), 303–311 (2011)
2. Taillandier, P., Vo, D.-A., Amouroux, E., Drogoul, A.: GAMA: A simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In: Desai, N., Liu, A., Winikoff, M. (eds.) PRIMA 2010. LNCS, vol. 7057, pp. 242–258. Springer, Heidelberg (2012)
3. Carillo, M., Cordasco, G., De Chiara, R., Raia, F., Scarano, V., Serrapica, F.: Enhancing the performances of D-MASON - a motivating example. In: SIMULTECH, pp. 137–143 (2012)
4. Collier, N., North, M.: Parallel agent-based simulation with REPAST for high performance computing. In: SIMULATION (2012)
5. D'souza, R.M., Lysenko, M., Rahmani, K.: Sugarscape on steroids: Simulating over a million agents at interactive rates. In: Proceedings of the Agent 2007 Conference (2007)
6. Silveira, R., Fischer, L., Ferreira, J.A.S., Prestes, E., Nedel, L.: Path-planning for RTS games based on potential fields. In: Boulic, R., Chrysanthou, Y., Komura, T. (eds.) MIG 2010. LNCS, vol. 6459, pp. 410–421. Springer, Heidelberg (2010)
7. Maitre, O., Lachiche, N., Clauss, P., Baumes, L., Corma, A., Collet, P.: Efficient parallel implementation of evolutionary algorithms on GPGPU cards. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 974–985. Springer, Heidelberg (2009)
8. Laville, G., Marilleau, N., Lang, C., Mazouzi, K., Philippe, L.: Using GPU for multi-agent soil simulation. In: PDP 2013, Belfast, Ireland, pp. 392–399. IEEE Computer Society Press (February 2013)
9. Richmond, P.: FLAME GPU Technical Report and User Guide (CS-11-03). Technical report, Department of Computer Science, University of Sheffield (2011)
10. JOCL: Java bindings for OpenCL, `http://www.jocl.org/` (June 07, 2013)
11. Laville, G., Mazouzi, K., Lang, C., Marilleau, N., Philippe, L.: Using GPU for multi-agent multi-scale simulations. In: Omatu, S., Paz Santana, J.F., González, S.R., Molina, J.M., Bernardos, A.M., Rodríguez, J.M.C. (eds.) Distributed Computing and Artificial Intelligence. AISC, vol. 151, pp. 197–204. Springer, Heidelberg (2012)
12. Fischer, L., Silveira, R., Nedel, L.: GPU accelerated path-planning for multi-agents in virtual environments. In: Proceedings of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment, SBGAMES 2009, pp. 101–110. IEEE Computer Society, Washington, DC (2009)

13. Erra, U., Frola, B., Scarano, V., Couzin, I.: An efficient GPU implementation for large scale individual-based simulation of collective behavior. In: Proceedings of the 2009 Int. Workshop on High Performance Computational Systems Biology, HIBI 2009, pp. 51–58. IEEE Computer Society, Washington, DC (2009)
14. Bousso, M., Cambier, C., Masse, D., Perrier, E.: An offer versus demand modelling approach to assess the impact of micro-organisms spatio-temporal dynamics on soil organic matter decomposition rates. In: Ecological Modelling, pp. 301–313 (2007)
15. Blanchart, E., Marilleau, N., Drogoul, A., Perrier, E., Chotte, J.L., Cambier, C.: Sworm: an agent-based model to simulate the effect of earthworms on soil structure. EJSS. European Journal of Soil Science 60, 13–21 (2009)
16. Gutknecht, O., Ferber, J.: Madkit: a generic multi-agent platform. In: Proceedings of the Fourth International Conference on Autonomous Agents, AGENTS 2000, pp. 78–79. ACM, New York (2000)