Tools for Large Scale (Distributed) Agent-Based Computational Experiments

László Gulyás[1 2], Attila Szabó[2], Richárd Legéndi[2], Tamás Máhr[1], Rajmund Bocsi[1], and George Kampis[2]

[1] AITIA International, Inc., Budapest
[2] Lorand Eotvos University, Budapest
{lgulyas, aszabo, rlegendi, tmahr, rbocsi}@aitia.ai,
kampis.george@gmail.com

Abstract. This paper discusses a collection of tools aimed at helping agent-based modelers to run large-scale computational experiments using agent-based social simulations. We address both the issue of *running simulations with millions of agents* (technically, distributing individual simulation runs over a cluster of computers) and the issue of handling enormous parameter spaces. For this latter, we address two approaches. The first is a purely computational one that helps distributing computational experiments over a cluster of computers. In connection, we also describe an on demand simulation service, where modelers can rent (potentially hundreds) of computers by the hour over the web, providing access to an *„instant computer farm"*. The second approach addressed is more statistical than computational. We briefly refer to and discuss the application of careful *designs of experiments*.

1 Introduction

The spreading of computational simulations goes in parallel with the increasing availability of computational resources. Agent-based simulations, in particular, can generate a very high demand for computing power – for several reasons. On one hand, agent-based models typically require numerous input parameters to be specified, and the parameters can have multiple different values. Exploring the parameter space of such models means large-scale computational experiments, i.e., running a large number of individual simulations each initialized with different parameter combinations. On the other hand, simulations of agent-based models often need to deal with huge amount of agents (e.g., at the order of 105-106 agents). These cannot be executed within a single PC due to performance issues (e.g., limitations of memory). While the exact numbers and limits may vary, the need for executing „real-life" simulations that is on the rise these years, imposes that that even the ever more powerful PC's are soon outgrown.

As a consequence, it is often necessary to runădistributed simulations, either on a dedicated computing cluster, on some grid infrastructure, or using some cloud computing service. However, implementing a distributed program demands more sophisticated (and scarcer) software engineering skills, that are, in practice, not available for computational social scientists.

In this paper, we discuss a collection of tools we developed to help agent-based modelers to run large-scale computational experiments using agent-based social simulations. We address both the issue of running simulations with millions of agents (technically, distributing individual simulation runs over a cluster of computers) and the issue of handling enormous parameter spaces. For this latter, we address two approaches. The first is a purely computational one that helps distributing computational experiments over a cluster of computers. In connection, we also describe an on demand simulation service, where modelers can rent (potentially hundreds) of computers by the hour over the web, providing access to an „instant computer farm". The second approach addressed is more statistical than computational. We briefly refer to and discuss the application of careful designs of experiments

2 Distributing Individual Runs (Simulations with Millions of Agents)

Complex systems are characterized by complex interactions that map to a high level of inter-process communication in complex systems simulations. On the other hand, parallel computing applications tend to be most efficient when there is no or little demand for such inter-node communication. In order to minimize the execution time of a distributed complex system simulation, the computation has to be partitioned in such a way that the communication between interacting components is minimized. In addition, the computational load of each partition, determining the time between communication events, needs to be balanced.

An ABM typically consists of a set of autonomous communicating agents, and can be described formally by giving the state-transition functions of the agents and the interaction topology of the whole system. For realistic, „real life" simulations, simulations with large numbers of agents are required. To enable this and to reach acceptable performance, the execution of the model should be distributed on several processors. Thus, the development of parallel code for multi-core, cluster-level or inter-cluster computation is a natural solution. However, the parallelization of an ABM is typically a hard task, due the high level of inter-process communication involved. Different approaches can be found in the literature. Without aiming for completeness: from solutions that bring Graphical Processing Units into play [9], through the automatic parallelization of ABMs [13] to all the techniques that can be applied to spatial (e.g. ecological) ABMs [16] [5] [11], many solutions have been proposed. However, though to different extent, all these solutions assume a level of technical expertise that, on our opinion, renders them hard to use or hard to reuse.

In [6] we reported our efforts to enable researchers to do Agent-Based Modeling (ABM) on multicore computers, computing clusters and computational grids without the need to write parallel programs. Instead of creating individual parallel solutions for every single agent-based simulation, our open-source project (gridABM, `http://gridabm.sourceforge.net/` [15]) developed a categorization of the feasible communication topologies (see below) and provided ready-to use, transparent solutions („Dry Schemas") for them to approach a general, applications-independent solution. Using our toolkit, all the modeler needs to do to obtain a distributed version of her model is the following:

1. determine the communication topology of the model,
2. choose the appropriate template,
3. fill in the associated Dry Schema. (This can be done typically by sub-classing the Dry Schemas and filling in a few abstract methods.)

The main benefit of our toolkit lies in the use of standard technologies. The transparency of parallelized code is guaranteed by the ProActive framework [1], while simulation specific code can be written using the popular RePast J package, [12]. Using our tools, modelers are only required to write small pieces of simulation specific code (that they would need to write for a sequential version as well), and nothing else.

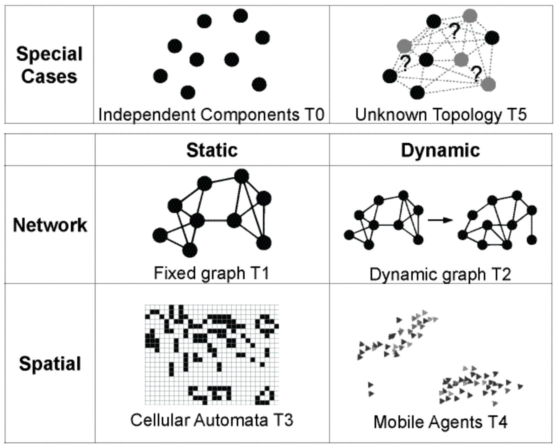


Fig. 1. The ABM communications templates identified by gridABM.

The categorization scheme consists of six categories referred to as communication templates here (see Figure 1) Template T0 is the simplest communication template. T0 allows no communication between the components, so partitioning is only constrained by load-balancing considerations. Perfectly (or „embarrassingly") parallel applications, such as parameter sweeps (see the next section), fall into this T0 category. Template T5 represents the other end of the spectrum of our communication templates. Essentially, T5 covers those complex systems for which it is difficult or impossible to determine a meaningful communication template and an a priori partitioning of the computation. („Well mixed" populations, where random interaction is assumed, belong here.) In such cases, no distribution of individual runs is possible. The only way to harness the powers of distributed computing is to use it for parameter space explorations. (In a sense, this means moving „one step up" and looking at the same system at a higher level of abstraction, where, as a parameter sweep problem, it now belongs to the T0 category.)

Template T1 describes the communication topology of a non-spatial complex system with a fixed element-to-element interaction topology (i.e., a fixed interaction network) that is known a priori. Template T2 applies to those systems whose interaction pattern is changing over time in an explicit, well-defined and well-describable fashion. In this case, the communication topology can be given as a graph and a transition function applied to it that may provide sufficient information for the system's efficient distribution. Template T3 covers classical cellular automata simulations. In this case, finding a good partitioning is straightforward. Template T4 describes those agent based simulations where agents are embedded in a metric space and interactions are local with respect to that space (e.g., mobile individuals moving in ordinary Euclidean n-space, where n is typically 2 or 3). According to our experiences, the outlined categories cover the most frequent types of ABMs. [8] Moreover, methods for partitioning them are available in the literature. This enabled us to implement reusable „Dry Schemas" for them.

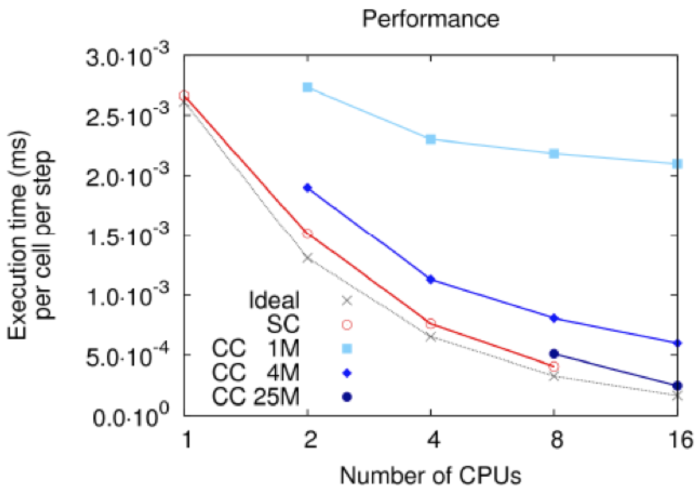


Fig. 2. Performance of a cellular automaton (CA) model of increasing size (1000x1000, 2000x2000, 5000x5000) implemented using the Dry Schema for T3. [6] The line with cross-symbols shows ideal performance (execution time of the non-parallel implementation divided by the number of processing cores). The line labeled „SC" shows single cluster runs on the QosCosGrid testbed. Legend: SC=single cluster, CC=cross-cluster (runs utilizing two geographically distant computing clusters), 1M=1 million cells (1000x1000), 4M=4 million cells (2000x2000), 25M=25 million cells (5000x5000)

To demonstrate of the power and usefulness of the approach, Figure 3 and 2 show performance statistics about two models using the gridABM package and scaling to 105 and 107 agents, respectively. Figure 3 reports on a version of the TAXSIM model of tax avoidance [14] that uses the Dry Schema for template T2. The measurements were carried out on the QosCosGrid testbed (`http://www.qoscosgrid.org/trac/qcg` [3]) using up to 50 processors. (The configuration marked by „3+35" used two geographically distant computing clusters, containing 5 and 35 nodes, respectively.) With simpler configurations, some population sizes were unmanageable, thus they are missing from the figure.

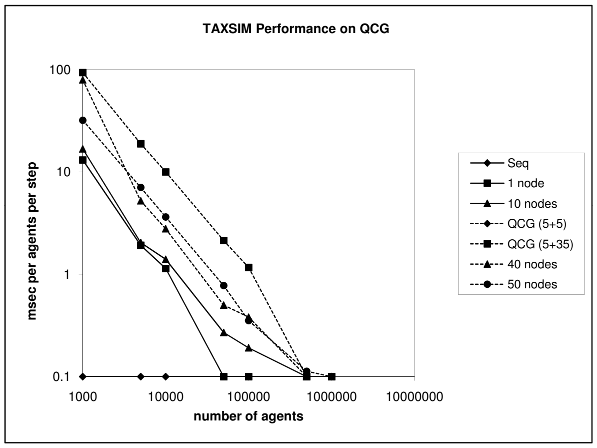


Fig. 3. Performance of the TAXSIM model using the Dry Schema for T2, executed on the QosCosGrid testbed, using up to 50 processors. The configuration marked by „5+35" involved two geographically distant computing clusters of 5 and 35 processing units, respectively. Some population sizes were unachievable for smaller computing clusters, thus their measurements are omitted from the figure.

Figure 2 shows the performance of a predator-prey simulation implemented on a CA that uses the Dry Schema for communication template T3. [6] The measurements were executed on the QosCosGrid testbed (`http://www.qoscosgrid.org/trac/qcg` [3]), using up to 16 computing cores (processors). Lines marked by CC were „cross-cluster" runs, utilizing two geographically distant computing clusters. The line with cross-symbols shows ideal performance (execution time of the non-parallel implementation divided by the number of processing cores).

Clearly, Figure 3 and 2 shows that our gridABM package is capable of executing simulations with unusually large numbers of agents, utilizing computing clusters consisted of regular PCs. Moreover, the performance statistics also indicate that the solutions scale relatively well. (Naturally, the benefits of distribution manifest especially for larger systems and for a larger number of processors.) Furthermore, it is important to emphasize that these simulations capitalize on the Dry Schemas provided by gridABM, meaning that their implementations are little more than the model definitions that would be also necessary for a sequential implementation.

3 Distributing Computational Experiments (Executing Millions of Runs)

The T0 template, discussed above, is a special case, when components of a simulation can be run independent of one another. The exploration of the parameter space of a model is a task similar to this. In many cases, this is the only task that can be easily helped by distributing the simulation. There are several different techniques available that can run programs in parallel. One possibility is to use a grid scheduler (like SGE or LSF [4]) to distribute simulation jobs on a cluster of computers. This is typically used on locally (university or company) owned data centers. A grid scheduler automates the parallel execution of jobs by allowing the user to submit jobs to queues, and deciding on the nodes the jobs should be assigned to. What the user still has to do by hand, in case of a parameter sweep setting, is the distribution of parameter combinations among the simulation jobs and the collection of the results from the nodes.

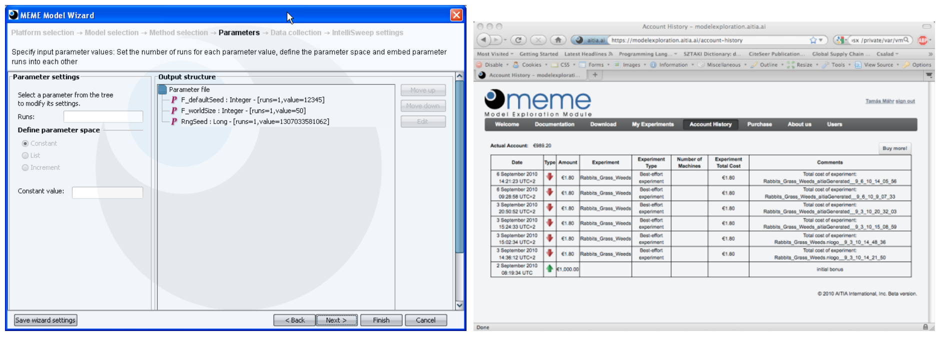
When a locally owned cluster is not available, a similar solution is possible by hiring machines from a cloud-service provider. To use a cloud service, the user has to prepare and upload a machine image that contains the software she wants to run in the cloud. Then, it is possible to start up this machine image on any number of computers providing the required computational workforce. In this case, running a distributed parameter sweep requires the same effort from the user as running it on a grid, plus the creation of a grid-based machine image (i.e. the installation of the grid software).

Both of the above mentioned parameter sweep distribution techniques have the drawback of requiring the user to manually set up the different parameter settings for the simulation jobs, and to manually collect the results. In addition, if one does not have access to a local computer cluster, there is the extra work of setting up simulation machines in the cloud. In case of a large-scale parameter-sweep experiment, this requires considerable extra work from the experimenter that could be saved by an automated tool.

Our solution to this problem is the user-friendly Model Exploration Service (MES, `http://modelexploration.aitia.ai/` [10]) that allows the user to run parameter sweep experiments of her model in a distributed fashion on a dynamic cluster of computers provisioned on demand. For this, the user only need to use her normal model development environment (FABLES, Repast J and NetLogo are currently supported, MASON support is planned). No installation or configuration of computers is required. Also, the user does not need to have a local cluster of computers at her disposal.

The user is assisted in designing, setting up, and submitting the experiment by a program running on her local computer. (This piece of software is called the Model Exploration Module, MEME, `http://meme.aitia.ai/`.) MEME connects to a subscription-based simulation service that takes care of distributing the experiment on multiple machines „in the cloud", and of collecting the results, which it also helps downloading tool for further processing.

Using MES and MEME, a distributed parameter-sweep experiment has the following workflow. First, the user sets up the experiment on her local machine using MEME. When finished, MEME uploads the simulation to the Model Exploration Server (given that the user provided her authentication tokens). The server is responsible for authenticating the user and for sending the experiment to one of the contracted cloud providers that is configured to run the cloud-end processes of the Model Exploration Service. The cloud end then runs the experiment, and returns status updates and finally the result to the server. The user can check the MES website to monitor the status of the experiment, or she can ask for an e-mail notification about the completion of his experiment. Upon completion, the user can use MEME to download the results from the server (and optionally also for further processing). Figure 4 shows screenshots of MEME (left) and the MES portal (right).



Specify input parameter values: Set the number of runs for each parameter value, define the parameter space and embed parameter runs into each other

Select a parameter from the tree to modify its settings.

Runs:

Define parameter space

[✓ Constant] List Increment

Constant value:

Parameter file
- F_defaultSeed : Integer - [runs=1,value=12345]
- F_worldSize : Integer - [runs=1,value=50]
- RngSeed : Long - [runs=1,value=1307033581062]

https://modelexploration.aitia.ai/account-history

Actual Account: €989.20

Date	Type	Amount	Experiment	Experiment Type	Number of Machines	Experiment Total Cost	Comments
6 September 2010 14:21:23 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds_aitiaGenerated__9_6_10_14_05_56
6 September 2010 09:28:58 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds_aitiaGenerated__9_6_10_9_07_33
3 September 2010 20:50:52 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds_aitiaGenerated__9_3_10_20_32_03
3 September 2010 15:24:33 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds_aitiaGenerated__9_3_10_15_08_59
3 September 2010 15:02:34 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds.nlogo__9_3_10_14_48_36
3 September 2010 14:36:12 UTC+2	↓	€1.80	Rabbits_Grass_Weeds	Best-effort experiment		€1.80	Total cost of experiment: Rabbits_Grass_Weeds.nlogo__9_3_10_14_21_50
2 September 2010 08:19:34 UTC	↑	€1,000.00					initial bonus

© 2010 AITIA International, Inc. Beta version.

Fig. 4. Screenshots of the local wizard (MEME) that helps setting up parameter space explorations (left) and the Model Exploration Service (MES) portal (right).

In [10] we have demonstrated the architecture and operation of the Model Exploration Service (MES) through an experiment using the Rabbits Grass Weeds model from the NetLogo model library. This model takes 6 parameters: the initial number of rabbits, the energy needed for a rabbit to reproduce, the energy gained by eating a patch of grass or weed, and the grow rates of grass and weed. Setting up an experiment for running in the cloud consists of a few consecutive steps. First the user has to specify the executable code of the agent-based model she wishes to experiment with. Once the executable code is located, MEME analyses the code and collects the possible input parameters, the accessible methods, and the agent attributes. Then the user is assisted in composing a parameter tree that describes the parameter combinations she wishes to test in his experiment. She can specify the parameter combinations manually, or he can make use of the built-in design of experiments methods (see below).

Subsequently, the user can define the variables she wants to record in the output file. It is possible to hook up input parameters, agent attributes, or accessible method calls of agents or the model to recorders that write the associated values to files. Additionally, MEME allows the user to use its interactive graphical interface to put together various statistics of the variables to be measured.

After the experiment is set up, the user can choose to run the experiment on the local host, on a local cluster of machines, on the QosCosGrid infrastructure [8], or on the cloud-based Model Exploration Service. If the user chooses the cloud-based service, he has to provide a username and a password that was previously registered on the Model Exploration Service portal. Finally, the user can submit the experiment to MES. Upon completion the server notifies the user via e-mail and the results are downloadable using MEME.

4 Exploring Enormous Parameter Spaces (Being Smart, Not Only Strong)

In the previous section, we have discussed the Model Exploration Service (MES) that assist modelers in exploiting distributed cloud computing resources in order to execute large-scale parameter space explorations (or sensitivity analyses). The MES is assisted by our MEME tool in setting up the experiment and loading it up to the MES server. In this section we briefly mention another very useful service of MEME that was discussed in detail elsewhere. [7]

In the scenario described above, it was the user who determined the parameter combinations to be sampled (run). MEME assisted this process by collecting all potential input parameters from the model source and offering them to the modeler. In addition to this, MEME also offers statistical tools to reduce the number of simulation runs needed for collecting enough information (i.e., samples) from a given parameter space region to judge the behavior of the simulation. These statistical methods are based on the design of experiments literature that was developed for the experimental sciences with the goal to reduce the number of costly experiments without jeopardizing the amount of knowledge that can be collected from the experiments. [2]

MEME has a growing list of classic designs of experiments, including (fractional) factorial design(s), the Box-Behnken design, the central composite design and Latin hypercube designs, as well as it offers a number of more advanced, dynamic exploration strategies, based on heuristic optimization methods. These latter include Genetic Algorithms and Iterative Interpolation.

5 Conclusions

This paper briefly introduced and discussed a set of tools that we have developed to enable and support large-scale agent-based simulations. Our tools address both typical directions of distributed computer simulations. The gridABM package makes the implementation of distributed simulations, where agents belonging to the simulation may reside on different computers (or processors), accessible to social simulators by providing „Dry Schemas" for a set of common agent communication templates. Using the Dry Schemas, modelers can focus on coding their models, while the tricks of the distributed implementation are added seamlessly by the schemas. On the other hand our Model Exploration Service (MES), that comes hand in hand with our Model Exploration Module (MEME) assists modelers in setting up large-scale parameter space explorations in a distributed fashion. In this case, individual simulations run on a single computer, but the large number of runs needed for the exploration are distributed over several computers. MES is a cloud-based „on demand" service, which means that users can rent computing facilities by the hour in order to scale up their computational capabilities. Finally, we briefly discussed another useful service of MEME that uses statistical methods to reduce the number of simulation runs needed to appropriately explore a given parameter space.

6 Acknowledgements

This work was partially supported by the Hungarian Government, under the grant KMOP-1.1.2-08/1-2008-0002. The work also benefited from the support of the European Commission via two projects (QosCosGrid IST FP6 STREP #033883, FP7-ICT-2009-5 STREP #256910). These supports are gratefully acknowledged.

References

1. Baduel, L., Baude, F., Caromel, D., Contes, A., Huet, F., Morel, M., Quilici, R.: Programming, composing, deploying for the grid. GRID COMPUTING: Software Environments and Tools pp. 205–229 (2006)
2. Box, G., Hunter, J., Hunter, W.: Statistics for experimenters: design, innovation, and discovery, vol. 13. Wiley-Interscience New York (2005)
3. Dubitzky, W., Kurowski, K., Schott, B. (eds.): Large-Scale Computing Techniques for Complex Systems Simulations. Wiley-Blackwell (In preparation)
4. Etsion, Y., Tsafrir, D.: A short survey of commercial cluster batch schedulers. Tech. Rep. 13, School of Computer Science and Engineering, the Hebrew University, Jerusalem, Israel (2005)
5. Glass, K., Livingston, M., Conery, J.: Distributed simulation of spatially explicit ecological models. In: pads. p. 60. Published by the IEEE Computer Society (1997)
6. Gulyás, L., Szemes, G., Kampis, G., de Back, W.: A modeler-friendly api for abm partitioning. Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, ASME, San Diego (August 2009)
7. Iványi, M., Gulyás, L., Bocsi, R., Szemes, G., Mészáros, R.: Model exporation module. In: Agent 2007: Complex Interaction and Social Emergence Conference. Evanston, IL, USA (November 2007)

8. Kurowski, K., de Back, W., Dubitzky, W., Gulyás, L., Kampis, G., Mamonski, M., Szemes, G., Swain, M.: Complex system simulations with QosCosGrid. Computational Science–ICCS 2009 pp. 387–396 (2009)
9. Lysenko, M., D'Souza, R.: A framework for megascale agent based model simulations on graphics processing units. Journal of Artificial Societies and Social Simulation 11(4), 10 (2008)
10. Máhr, T., Bocsi, R., Gulyás, L.: Simulation as a service: The model exploration service. 3rd World Congress on Social Simulation, Kassel, Germany (September 2010)
11. Mellott, L., Berry, M., Comiskey, E., Gross, L.: The design and implementation of an individual-based predator-prey model for a distributed computing environment1. Simulation Practice and Theory 7(1), 47–70 (1999)
12. North, M., Collier, N., Vos, J.: Experiences creating three implementations of the repast agent modeling toolkit. ACM Transactions on Modeling and Computer Simulation (TOMACS) 16(1), 1–25 (2006)
13. Scheutz, M., Schermerhorn, P.: Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. Journal of Parallel and Distributed Computing 66(8), 1037–1051 (2006)
14. Szabó, A., Gulyás, L., Tóth, I.J.: TAXSIM agent based tax evasion simulator. In: 5th European Social Simulation Association Conference (ESSA 2008). vol. (preprint). Brescia, Italy (October 2008)
15. Szemes, G., Gulyás, L., Kampis, G., de Back, W.: Gridabm - templates for distributed agent based simulation. Open Grid Forum 28, Munich, Germany (2010)
16. Wang, D., Berry, M., Gross, L.: On parallelization of a spatially-explicit structured ecological model for integrated ecosystem simulation. International Journal of High Performance Computing Applications 20(4), 571 (2006)