Pandora: A Versatile Agent-Based Modelling Platform for Social Simulation

Xavier Rubio-Campillo

Barcelona Supercomputing Center
Computer Applications in Science & Engineering
C/Gran Capita 2-4, Nexus I Building, Barcelona
Email: xavier.rubio@bsc.es

***Abstract*—The evolving field of social simulation is diversifying the degree of complexity of published works, from simple models to large scale simulations with millions of agents. In this context, available platforms are divided between the ones favoring easy to use interfaces and the development of prototyping, and others created for simulating large scenarios with high computing costs. The later group is truly diverse, based on the way executions are accelerate and the wide range of technologies that they can support. The cost of this increase in performance is a steep learning curve, as the users of these platforms need to have advanced programming skills in order to deal with code complexity. Pandora is a novel open-source framework designed to fill the gap between these approaches. A twin interface in Python and C++ offers the same interface to users developing prototypes as well as projects with High-Performance Computing requirements. The need for parallel programming knowledge is also skipped through the automated generation, during compilation time, of needed code for shared and unshared memory distribution using OpenMP and MPI. A set of different helpers (unit testing, georeferencing support) and analytical tools complement the basic framework in order to facilitate the tasks of development, testing and analyzing any type of Agent-Based Model. Pandora's flexibility is exemplified through different projects that have introduced GPU acceleration, georeferenced data and cloud computing to the framework.**

Keywords–Agent-Based Models; Social Simulation; High-Performance Computing; Parallel programming; Multi-Agent Systems

I. Introduction

The disciplines studying human behavior are being increasingly interested on the use of Agent Based Models (ABMs) as virtual laboratories to explore and validate their research hypotheses. This trend has determined the appearance of a growing number of platforms specifically designed to assist projects creating this kind of models. At the same time, the diversification of case studies and requirements has created a bifurcation between platforms created to implement prototypes and simple models and those designed to face advanced requirements, specially regarding the acceleration of the execution.

The first group of platforms tends to emphasize easy to use interfaces, and a quick learning curve that allows for producing models in a short span of time. These features are extremely interesting for a sensible part of the community of social scientists, as it allows to create, deploy and explore simulations with little or non-existent previous programming experience. The powerful capability of creating models without expert programming skills explains the success of platforms such as NetLogo [1], MASON [2] and Repast Simphony [3]. Similar features are provided by Model Driven Engineering platforms, such as INGENIAS [4] and modelling4all [5].

The drawback of this approach is scalability. Graphical tools and dynamic programming languages are not focused on this aspect of software engineering, and they tend to have serious efficiency issues. This is not relevant for simple models, but as the complexity of a model grows the idea of switching towards more efficient programming languages takes importance. This decision has a major impact in the model. Advanced programming skills are required for this task, as languages such as Java or C++ have a steep learning curve. Besides, this decision not always affect the programming language but also the platform chosen to create the ABM. In particular, the Application Programming Interface (API) of the different platforms can be extremely different, even when the programmer is replicating an existing model. Finally, the change of programming language is not enough improvement on efficiency when the models contain a large number of agents with complex behavior; in this case, the better approach is to choose a platform capable of distributing the execution of a simulation.

Parallel programming is one of the most challenging aspects of software development. In particular, the distribution of an ABM is strongly dependent on the nature of the problem to model and the properties of the system (degree of interaction between agents, complexity of behavior, etc.). There is no optimal way to distribute any kind of ABM, because their dynamics are extremely diverse. For this reason there are different strategies that could split an ABM execution amongst different computer nodes, but none of them will be universally optimal. The only possible solution is to provide a set of different techniques that can be applied to similar problems. This idea is the basis of different initiatives like GridABM [6], a framework of template solutions for distributing these type of simulations.

All this diversity of models and software platforms implies that any project involving ABMs must carefully evaluate existing options, considering its primary goals. If a wrong platform is chosen it will be difficult to fix the problem, and this could have a major impact on the success or failure of the project. This also applies when there is a high degree of uncertainty or requirements change over the span of the project, situations that can easily happen in the context of research. The Repast suite [7] is currently the only solution to this issue, as it offers a wide arrange of different tools to use, from graphical modelling to distributed executions. However this is not an optimal solution, as the most advanced tool in terms of computation (Repast-HPC) does not share the same code or language than the rest.

The discussed issues can be transformed in a set of requirements for an all-purpose ABM platform:

- Rapid prototyping in a dynamic programming language
- An alternate interface providing access to efficient version of the same functionality
- The user should be able to analyze their models with a wide array of analytical tools, including spatial analysis, statistics and visualization.
- Parallel execution, not only of different runs but also of a single, CPU-demanding, simulation.
- The switch between sequential and distributed executions should not translate into a re-implementation of the model or in learning the complexities of parallel programming.

This paper presents Pandora, an open-source ABM framework designed to deal with the varied needs of modellers and fill the gap between prototyping and advanced simulations. This innovative platform has a flexible architecture capable of providing the tools needed to create any type of ABM and execute it in any environment in a transparent way. This includes dynamic prototyping, automated parallel execution and analytical tools.

Next Section describes the different tasks involved in the creation of an ABM, that any framework should cover. Next, we describe the general structure of the platform, before focusing on the core concepts of the design fulfilling the requirements. The paper closes with a discussion on the present and future of the platform.

II. Creating an Agent-Based Model

The methodologies used to create agent-based simulations are as diverse as their goals [8]. The set of utilities and analytical tools needed to implement, explore and publish an ABM makes a strong case for using a general framework capable of providing functionality for all these tasks. This software platform should provide enough flexibility to go from exploratory models to predictive simulations using the same code. On the other hand social simulations are usually created inside interdisciplinary initiatives, and this has an important impact in the different steps of the methodology. The general work-flow for this development, independently of the platform, can be summarized as followed:

1) Definition of a research question
2) General definition of a model using Overview, Design Concepts and Details (ODD) protocol [9], Unified Modelling Language (UML) [10] or similar modelling tools)
3) Development of a prototype using a dynamic programming language
4) Exploratory visualization and verification of results
5) If needed, implementation of an efficient version of the model
6) Design and execution of a set of experiments
7) Analysis and dissemination of results

This classical modelling methodology is not the only way to create ABMs. Software engineering practices are increasingly being used in the field; for this reason, the framework should not force the user to create their models with a particular methodology, and it could be interesting to have support for agile methodologies, including the use of Test-Driven Development, intensive refactoring and collective ownership of the code of [11][12].

III. General Design

The general architecture of Pandora is shown in Figure 1. The abstract classes *World* and *Agent* are the core of the library, as they form the content of any model. The first one manages the different layers of continuous information that define the environment of the simulation. Following Geographical Information Systems modelling practices the environment is defined as a set of raster maps (bi-dimensional matrices of values), that can have read-only data and are encapsulated in instances of the class *StaticRaster*. They are complemented by *DynamicRasters*, whose values that can be modified over the span of the simulation. The class *Config* also provides the functionality needed to initialize any *ConcreteWorld*, usually from an eXtensible Markup Language (XML) file with all the parameters of a run (duration, size of the map, output directory, etc).

The *Agent* class encapsulates any entity of the model with internal state, decision making processes and behavior. Any *ConcreteAgent* needs to define at least a method *updateState* that is executed every time step as well as *serialize*, where its state will be stored in a file using the Hierarchical Data Format (HDF5) protocol [13].

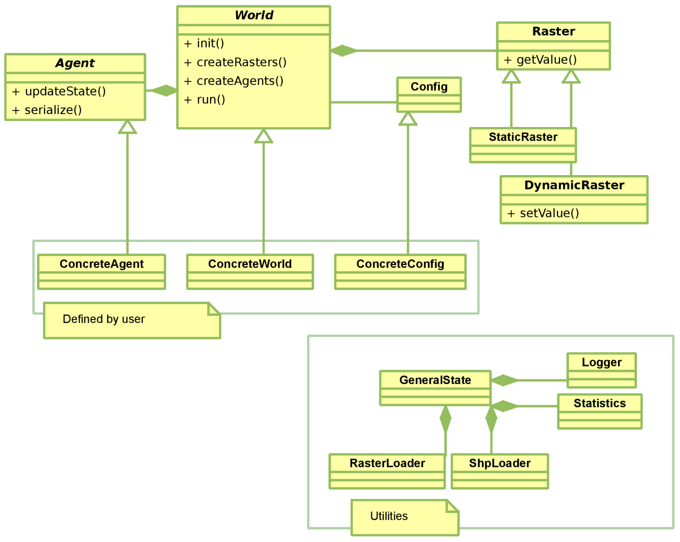

Figure 1. Class diagram of the framework

A set of different utilities and helpers provides functionality to load georeferenced data, log relevant information and generated pseudo-random numbers needed for stochastic processes. This architecture is specifically designed to host any potential ABM, from simple prototypes to complex and realistic models. Next sections define the different solutions that have been implemented to deal with the requirements, in particular (a) dynamic/static programming, (b) analytical tools and (c) scalability.

IV. Twin programming interface

One of the most important requirements for this platform is to facilitate the transition between simple and complex simulation environments. Pandora implements its functionality in C++, which can be accessed from a two-headed interface in C++ and Python. The first interface offers an efficient version that can also be distributed in a computer cluster, while the flexibility of Python allows any user with minimal programming skills to develop a model from scratch. The link between both interfaces uses the library boost-python, that provides a clear way of binding Python calls to C++ classes and functions.

As an example of the flexibility provided by this solution, a simple Agent that moves randomly can be defined in C++ as:

```
class BasicAgent : public Agent
{
public:
BasicAgent( const std::string & id ) : Agent(id) {}
~BasicAgent() {}

void updateState() {
    Point2D<int> newPosition = getPosition();
    newPosition._x += getUniformDistValue(-1,1);
    newPosition._y += getUniformDistValue(-1,1);

    if(getWorld().checkPosition(newPosition))
    {
        setPosition(newPosition);
    }
}
};
```

The same behavior is achieved in Python using this code:

```
class BasicAgent(Agent):
 def __init__(self, id):
  Agent.__init__( self, id)

 def updateState(self):
  newPosition = self.getPosition()
  newPosition._x += random.randint(-1,1)
  newPosition._y += random.randint(-1,1)

  if self.getWorld().checkPosition(newPosition):
   self.position = newPosition
```

Any code can be translated from one version to the other, as both languages are designed to implement the Object-Oriented paradigm and the interface to Pandora's functionality is exactly the same. An important side-effect of this solution is the maintainability of the framework. There is no duplicated functionality between version of the framework in different programming languages, as both interfaces call the same C++ code (i.e., complete examples can be found at [14]. This approach decreases the number of lines, thus diminishing the potential appearance of bugs in the code. In addition, both interfaces are checked by two unit testing suites, one using the basic python testing framework and the other one with the boost unit test toolbox. Finally, the analytical tools are common for both interfaces, as models implemented with any of the languages serializes result files with identical format, as they are calling the same serialization methods.

V. Multiple schedulers

ABMs are developed to explore a wide array of research questions. In essence any scenario where micro behaviors generate macro dynamics can be modelled with this technique. At the same time, there is a key component of any ABM that must be present in all possible models: the scheduler. This is the process that updates the set of agents and the environment that together form the model, and manages issues like the order of execution of the agents, and the way they interact with the other components (i.e., other agents, layers of raster maps, etc.). The heterogeneity of problems to be modelled suggest that there is no optimal scheduling algorithm. For example, a scheduler that proofs efficient in the sequential execution of simple models in a laptop will probably be incapable of managing distributed simulations with millions of agents. Besides, the way agents interact with each other is quite different, depending on the purpose of the model. For example, agents living in spatially structured models will have a completely different interaction with other entities than a model guided by social networks. The consequence of the flexibility of the ABM concept is that there is no optimal general strategy for a platform's scheduler.

The relevance of this issue grows exponentially when dealing with distributed executions [15]. Each agent needs to gather knowledge from the surrounding environment, including raster maps and other agents, before making any decisions. The agents then execute their decision-making processes and modify the environment. These mechanics translate, in terms of parallelization, in the need of sharing several layers of environmental data and agents between computer nodes, at every single time step. Furthermore, the execution of the agents' actions cannot usually be distributed within a single computer node (i.e., OpenMP), because there can be several conflicts related to agents accessing and modifying the same data at the same time.

Pandora's design follows the philosophy of versatility, combining the software engineering patterns *bridge* and *factory method* [16], as shown in Figure 2.

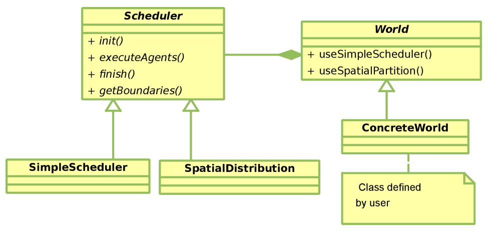

Figure 2. Class diagram of the Scheduling system

Any functionality of the base class World that needs to update the environment or the agents is delegated to the class Scheduler. Custom Schedulers implement different ways to manage these tasks without implying a change in the Application Program Interface (API) of the framework. The user only needs to specify the preferred Scheduler in the code. This is done as follows for multithreaded execution in a single computer (using OpenMP):

```
ConcreteWorld world(config,world.useOpenMPSingleNode());
```

or spatial distribution in different computer nodes using Message Passing Interface (MPI) and OpenMP:

```
ConcreteWorld world(config,world.useSpacePartition());
```

The solution also provides a clear interface for advanced users to develop their own schedulers if they need particular solutions for their models. In any case, the *Bridge* pattern allows that any Scheduler can be used in the execution of any simulation. This is particularly useful for exploring a model in different scales, as quick scenarios will use multithreading while the largest ones can be deployed in a High-Performance Computing (HPC) infraestructure.

A. Parallel execution

The only current Scheduler of Pandora designed for distributed execution is based on spatial partition. Each computer node owns a section of the entire simulated scenario, containing the different landscape data as well as the agent. This is one of the most popular ways of distributing an ABM, as is also the solution adopted by Repast-HPC [7].

The world of the simulation is evenly divided among computer nodes, and each one of them owns a section of the environment, as well as the agents located inside its boundaries. This layout is depicted in Figure 3. Information in the border between adjacent nodes (raster maps and agents) is communicated to neighbours every time step execution, in order to keep up-to-date data in the entire scenario. The size of this buffer border is defined as the maximum interaction range of any agent, being the absolute horizon of actions of any component of the simulation. The solution is scalable, given the fact that every computer node will need to communicate, at most, with 8 neighbouring nodes (if nodes own rectangular regions), independently of the total size of the simulation. On the other hand communication must be local, as agents can only communicate inside a given interaction range.

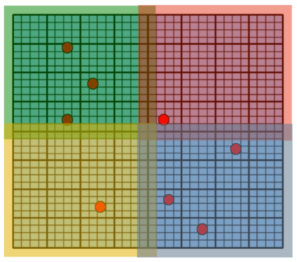
Figure 3. Spatial partitioning of an ABM. Each color represents the section of the world owned by a different computer node.

This solution does not solve to problem that two agents *living* in different computer nodes can modify the same bordering data at the same time. This potential collision can be avoided by different technique, but most of them can be computationally intensive (e.g., rollbacks). This overhead is affordable if agent behavior is CPU intensive and the possibility of conflict is low, but this is seldom the case with ABMs.

Pandora's spatial partition scheduler uses a simpler approach. The spatial section owned by a computer node is split into four equal parts numbering 0 to 3, as seen in Figure 4. The agents inside 0 sections are executed simultaneously; as they are not adjacent there is no possibility of collision between their actions. Once all 0's are finished, modified border data is sent to the neighbors, and a new section will begin its execution (1, 2 and finally 3). Once all of them are executed, the entire state of the simulation is serialized and a new time step can be evaluated.

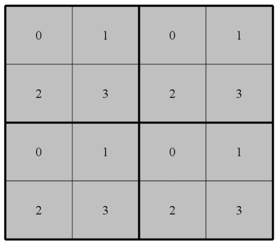

Figure 4. Each computer node is divided into four different sections, that are executed sequentially

The pitfall of this solution is that agents in section 0 will always be executed before agents in sections 1-3. Depending on the model the consequences of this effect can be nonexistent, or introduce artifacts in the outcome. As usual, a careful choice between the different strategies is needed, based on the existing scenario; Pandora provides the way to implement and use any existing algorithm designed to distribute ABMs [17][18][19].

B. Simultaneous execution of agents

Simulation performance can also be increased using the complete set of CPU cores of every computer node to simultaneously execute the agents. Again, the problem of collisions between agents' actions must be solved. Performance analysis showed that most of the execution time is spent when agents (1) gather information, (2) choose a particular set of behaviors, and (3) execute them. All ABM platforms mix these phases in a single method, executed by every agent every time step: *tick* in Netlogo [1], *step* in MASON [2] and RePast [20].

Agents do not modify anything in phases (1) and (2), as they just evaluate potential course of actions depending on existing data; if we separate them from the action's executions they can be simultaneously executed without risk of collisions.

Pandora uses this approach to split the step of an agent in three different methods. In the first one, *updateKnowledge*, an agent cannot modify the environment or other agents; it only gathers information. In the second one, *selectAction*, the agent executes her decision-making process to chooses an action Once every agent has chosen what she wants to do Pandora executes the actions of the agents sequentially. Finally, the third method that a user can specify is *updateState*, where any agent can modify its internal state evaluating the results of her actions. This cycle *Explore* - *Decide* - *Apply* allows Pandora to distribute the execution amongst different Central Processing Unit (CPU) cores of a node, as the first two methods are declared *const* in C++ to be thread-safe, while the third one is executed sequentially.

This structure seems more complicated than just defining one method but, from a theoretical point of view, the division of an agent's execution in these three phases is more consistent than the traditional ABM approach. The single method implementation mixes the different stages of an agent's cycle, that should be correctly specified while building the model (see Figure 5). Dividing the execution in these phases avoids this issue, forcing coherence during the transition from theory to code.

Finally, from a theoretical point of view this solution is more elegant, as it matches the definition of an intelligent agent [21].

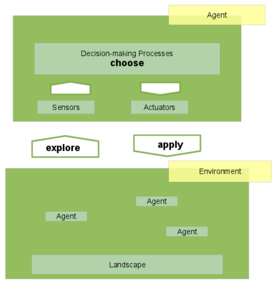

Figure 5. The execution cycle of an agent

To conclude, a performance analysis of Pandora's scheduling system [15], both in (a) supercomputers and (b) cloud HPC infraestructures. Results show again the diversity of challenges that any ABM platform need to face, as its execution in (b) is optimal when the balance between intensity of communication and CPU needs are shifted towards the later, while (a) is needed if the model has a high degree of interaction.

VI. Analysis

Any ABM framework is not complete without a set of tools to assist experiment design and analysis. Pandora's development team has created Cassandra [22], a GUI tool to help the user to perform the required tasks, from exploring a single run to the exploration of parameter space across thousands of executions.

A. Single run examination

The graphical interface allows to visualize the spatiotemporal dynamics of a model, as can be seen in Figure 6. This feature is useful to detect general patterns, as the user can check and track, at any given moment, the state of the different agents and layers of information.

B. Parameter space exploration

The *Laboratory* tool allows to define and run an experiment, based on the values that the used defines for each input parameter specified in the configuration. An HDF5 file is generated by each run, that are parsed in order to extract the needed summary statistics for the analysis.

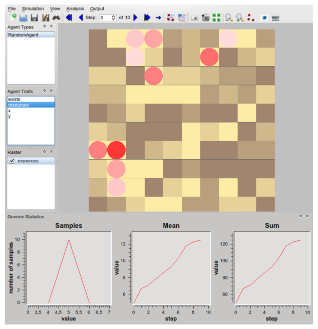

Figure 6. Spatial visualization of a single run

C. Exploratory Data Analysis

The output produced by thousands of ruins of an stochastic ABM can be extremely complex to study, and visual analytics can be an optimal choice to explore preliminary results [23]. At the present Cassandra includes two interactive tools: (a) heat maps to compare particular parameter values and (b) time series for understanding temporal dynamics, as seen in Figure 7.

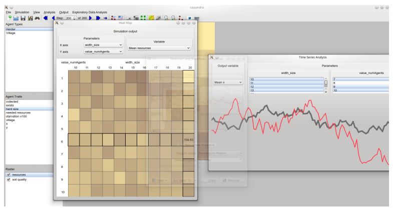

Figure 7. Exploratory visualization toolbox depicting the same set of runs with a heatmap and time series

D. Output

The final step is the use of the analytical toolbox to collect data from the dynamics of the system in different formats that can be used by different applications (i.e., Comma Separated Values for Statistical packages, GeoTiff and Shapefiles for Geographical Information Systems, etc.). Additional outputs ready to be used in publications and reports are composite mosaics encapsulating several runs in any video format, as well as Google Earth georeferenced movies.

VII. Conclusions and Future Work

The popularization of social simulation has increased in recent years the number and features of open-source ABM platforms [24]. This trend, while common and positive in any scientific software community, has been one of the reasons why replicability is scarce [25]. This issue is combined with a limited use of software engineering to control the quality of scientific code [26].

In addition, there is a technical gap between exploratory models created for theoretical research and realistic models developed for hypothesis testing and prediction. The first class of ABMs are implemented by social scientists in platforms with little or non-existent capability for multiplatform execution, while the second type of models is created by programmers with advanced programming skills in other software packages, and deployed in powerful hardware infrastructures. The paradox is that simple simulations cannot be distributed without a major coding effort, while distributed simulations are not easily executed in a standard computer. Pandora closes this gap in the increasingly diversified environment of social simulation; it provides enough flexibility to be extended by any user requirement, while maintaining its role of an all-purpose platform that can be used with any kind of model and/or infraestructure.

This versatility has been shown in a wide array of scenarios, from simple models exploring theoretical issues [27] to realistic simulations executing complex behavior for thousands of agents and parameter configurations [28][29]. Its use has also been extended through additional functionality like the use of Graphics Processing Unit (GPU) acceleration for particular agent actions [30], advanced decision-making processes using Markov Decision Processes [27] or its deployment in cloud HPC infraestructures [15]. Next steps include the development of a network-based scheduler, binary installation packages and additional visualization tools.

VIII. Acknowledgements

This research is part of the SimulPast Project (CSD2010-00034) funded by the CONSOLIDER-INGENIO2010 program of the Ministry of Science and Innovation Spain. We would like to thank the past and present member of the development team, and the anonymous reviewers for their comments. Pandora currently works under Linux and OS X operating systems and can be downloaded from http://xrubio.github.io/pandora.

References

[1] U. Wilensky, "Netlogo," Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999, retrieved: September, 2014. [Online]. Available: http://ccl.northwestern.edu/netlogo/

[2] S. Luke, "Multiagent simulation and the mason library," George Mason University, August 2011, retrieved: September, 2014. [Online]. Available: http://cs.gmu.edu/ eclab/projects/mason/manual.pdf

[3] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with Repast Simphony," Complex Adaptive Systems Modeling, no. 1:3, 2013.

[4] J. Pavón and J. Gómez-Sanz, "Agent oriented software engineering with ingenias," in Multi-Agent Systems and Applications III. Springer Berlin Heidelberg, 2003, pp. 394–403.

[5] K. Kahn and H. Noble, "The modelling4all project a web-based modelling tool embedded in web 2.0," in Proceedings of the 2Nd International Conference on Simulation Tools and Techniques, ser. Simutools '09. ICST, 2009, pp. 50:1–50:6.

[6] L. Gulyás, A. Szabó, R. Legéndi, T. Máhr, R. Bocsi, and G. Kampis, "Tools for large scale (distributed) agent-based computational experiments." in Proceedings of the Computational Social Science Society of America Annual Conference 2011, 2011.

[7] N. Collier and M. North, "Repast hpc: A platform for large-scale agent-based modeling," in Large-Scale Computing Techniques for Complex System Simulations, W. Dubitzky, K. Kurowski, and B. Schott, Eds. Wiley, 2011, pp. 81–110.

[8] B. Heath, R. Hill, and F. Ciarallo, "A survey of agent-based modeling practices (january 1998 to july 2008)," Journal of Artificial Societies and Social Simulation, vol. 12, no. 4, 2009, p. 9, retrieved: September, 2014. [Online]. Available: http://jasss.soc.surrey.ac.uk/12/4/9.html

[9] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback, "The odd protocol: A review and first update," Ecological Modelling, vol. 221, no. 23, 2010, pp. 2760–2768.

[10] H. Bersini, "Uml for abm," Journal of Artificial Societies and Social Simulation, vol. 15, no. 1, 2012, p. 9, retrieved: September, 2014. [Online]. Available: http://jasss.soc.surrey.ac.uk/15/1/9.html

[11] R. Mugridge, "Test driven development and the scientific method," in Agile Development Conference, 2003. ADC 2003. Proceedings of the, June 2003, pp. 47–52.

[12] M. Sletholt, J. Hannay, D. Pfahl, and H. Langtangen, "What do we know about scientific software development's agile practices?" Computing in Science Engineering, vol. 14, no. 2, March 2012, pp. 24–37.

[13] M. Folk, A. Cheng, and K. Yates, "Hdf5: A file format and i/o library for high performance computing applications," in In Proceedings of the 12th Conference on Supercomputing, Portland, November 1999.

[14] "Pandora: an agent-based modelling framework for large-scale distributed simulations," http://xrubio.github.io/pandora, retrieved: September, 2014.

[15] P. Wittek and X. Rubio-Campillo, "Scalable agent-based modelling with cloud hpc resources for social simulations," in Proceedings of the 4th International Conference on Cloud Computing Technology and Science, 2012, pp. 355–362.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[17] M. Scheutz and P. Schermerhorn, "Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models," Journal of Parallel and Distributed Computing, vol. 66, no. 8, 2006, pp. 1037–1051.

[18] D. Chen, G. K. Theodoropoulos, S. J. Turner, W. Cai, R. Minson, and Y. Zhang, "Large scale agent-based simulation on the grid," Future Generation Computer Systems, vol. 24, no. 7, 2008, pp. 658–671.

[19] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "Flame: simulating large populations of agents on parallel hardware architectures," in Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1633–1636.

[20] M. North, T. Howe, N. Collier, and J. Vos, "A declarative model assembly infrastructure for verification and validation," in Advancing Social Simulation: The First World Congress, S. Takahashi, D. Sallach, and J. Rouchier, Eds., 2007.

[21] S. J. Russell and P. Norvig, "Intelligent agents," in Artificial Intelligence: A Modern Approach, 2nd ed. Pearson Education, 2003.

[22] "Cassandra: a lab for agent-based model analysis," https://github.com/xrubio/pandora/tree/master/cassandra, retrieved: September, 2014.

[23] J. J. Thomas and K. A. Cook, "A visual analytics agenda," IEEE Comput. Graph. Appl., vol. 26, no. 1, Jan. 2006, pp. 10–13.

[24] C. Nikolai and G. Madey, "Tools of the trade: A survey of various agent based modeling platforms," Journal of Artificial Societies and Social Simulation, vol. 12, no. 2, 2009, p. 2, retrieved: September, 2014. [Online]. Available: http://jasss.soc.surrey.ac.uk/12/2/2.html

[25] M. Lake, "Trends in archaeological simulation," Journal of Archaeological Method and Theory, vol. 21, no. 2, 2014, pp. 258–287.

[26] D. C. Ince, L. Hatton, and J. Graham-Cumming, "The case for open computer programs," Nature, vol. 482, no. 7386, 02 2012, pp. 485–488.

[27] P. Wittek, I. Lim, and X. Rubio-Campillo, "Quantum probabilistic description of dealing with risk and ambiguity in foraging decisions," in Quantum Interaction, ser. Lecture Notes in Computer Science, H. Atmanspacher, E. Haven, K. Kitto, and D. Raine, Eds. Springer Berlin Heidelberg, 2014, pp. 296–307.

[28] X. Rubio-Campillo, J. Cela, and F. Hernàndez, "Simulating archaeologists? Using agent-based modelling to improve battlefield excavations," Journal of Archaeological Science, vol. 39, 2012, pp. 347–356.

[29] X. Rubio-Campillo, J. M. Cela, and F. Hernàndez, "Development of new infantry tactics during the early eighteenth century," J Simulation, vol. 7, no. 3, Aug 2013, pp. 170–182.

[30] P. Wittek and X. Rubio-Campillo, "Social simulations accelerated: Large-scale agent-based modeling on a gpu cluster," in GPU Technology Conference, San Diego, 2013, poster DD06.