

SNAP: Scalable Networkable ABM Platform for the Social Sciences

Christopher M. Conway ^a

IESEG, School of Management, 3 Rue de la Digue, Lille, France

Keywords: Agent-Based Modeling, Distributed Systems, Computing for Social Sciences.

Abstract: Agent-Based models (ABMs), although increasingly useful and widespread, are underused in social science. I show that extant “user-friendly” platforms are not well suited for social science research, while platforms that would support such research are not easy to use. I outline requirements for a sufficiently powerful, easy-to-use system, which requires no programming skills on the part of the user. I explain the design and development of an ABM which features a GUI and a menu of agents, statistics, and visualizations which are commonly desired. This system is robust enough for social science research; it is portable, flexible, and customizable. Users will have access to pre-designed complex and recursive agents, running distributed across all available processors, as well as user-selected geometries and time clocks. Progress and future work are discussed.

1 INTRODUCTION


Agent based modeling (ABM) has come a long way from its humble beginnings, and has proved its worth in a number of fields, from environmental sciences to genetics and epidemiology. ABM can provide preliminary results to research questions which cannot be explored directly; it can eliminate inappropriate solutions; it can illuminate emergent behavior; and it can simplify complex environments. Perhaps most usefully, ethical considerations can be ignored or resolved using “agents” before testing with live subjects, whether that is a pond ecosystem or corporate employees (Gilbert, 2007).

Because the social science field considers the behavior of human actors, ethical considerations are in play almost immediately, which constrains research. Furthermore, experiments are often difficult to replicate, since human subjects are unreliable due to many biases and research threats (such as social desirability, hypothesis guessing, or experimental interference) which may cause them to react inaccurately to a study, to which ABMs are immune (Gilbert, 2007). Thus, ABMs enable social science researchers to do considerable preliminary work, eliminating dead ends and ethical traps before the first subject is contacted.

However, ABMs are considerably underused in social science (Conte & Paolucci, 2014; Hughes et al., 2012; Jackson et al., 2017). Seeing an opportunity, I began to search for social science research, particularly in applied social psychology, which was well adapted to expansion with ABM. However, it quickly became clear that the existing platforms are not well suited for social science research, and/or are difficult to use. I realized that what was needed was an ABM which is well adapted to social science research.

We know from decades of research with the Technology Acceptance Model (TAM) (Davis, 1989) that, in order to be adopted, technology should be perceived as both easy to use and useful. Of currently available ABM platforms, those that are easy to use are not well suited to social science research, and those that support social science research are not easy to use. This contention is supported by papers reviewing ABM platforms (Abar et al., 2017).

As an experienced software engineer now working in the social sciences (MIS), I am well positioned to create an ABM for social science research. In this paper, I will first provide the requirements for a system which would be easy to use and useful for research in social science, and briefly examine major current platforms in light of those

^a <https://orcid.org/0000-0003-0108-9177>

requirements. I will then present a design for a system which I believe will meet all the requirements. Finally, I will explain my progress in implementing this platform, and what future work I am planning.

2 ABM AND THE SOCIAL SCIENCES

2.1 Making an ABM Easy to Use

Social scientists are not, generally, also experts in computer science and programming, especially systems programming. Early conversations with my social sciences colleagues suggest that they need a way to specify their ABM models simply and intuitively, without requiring knowledge of application efficiency, parallelization, synchronization, or guaranteeing consistency. As with many users of this type, this means that they need a graphic interface which will provide a simpler set of intuitive and, to the extent possible, self-explanatory tools to allow execution of typical tasks without requiring detailed knowledge of the underlying system. For ABM systems, these typical needs are as follows:

1. Select among “typical” agents and add them to a simulation run, without needing to program them
2. Extract typical data and statistics from a run, either by integrating with a common statistical program or by providing raw data in a format usable by common statistical programs
3. Initialize simulation runs easily, including options to specify multiple runs with or without parameter changes
4. Predict run time required when possible, given a set of computing resources and past history
5. Provide typically desired visualizations of a run in progress
6. Perform the above tasks from the user interface, without any required programming

2.2 Making an ABM Useful

The research domain of social sciences is humans and their interactions. This introduces complexity at both the micro and macro levels. At the micro level, each human is a complex entity with goals, desires, emotions, moods, beliefs, attitudes and behaviors. An individual is, in fact, an entire ecosystem. At the macro level, humans are embedded in the society that surrounds them. This embedding is recursive; humans are embedded in groups, which are

embedded in larger groups, and larger, until one gets to the entire population of the planet. Additionally, each individual is embedded in multiple groups, which may or may not share memberships. The study of humans is inherently multilevel and has an extremely complex geometry. Thus, the ideal ABM would require considerable flexibility and nuance in the agent specifications. In fact, the meaning of “agent” itself becomes complicated, since agents may contain other agents: individuals are embedded in groups, which are embedded in larger groups recursively. Additionally, they are often members of multiple groups, and the group membership of these groups may have non-identity and non-null intersections.

What this means in practice is a considerable increase in complexity, on several fronts.

In the first place, the addition of nuance requires that each agent make multiple choices, often among multiple options. Most ABM platforms require that these choices happen in sequence. Each additional option, or nuance, is likely to double (or more) the processing time and memory required. At its most basic level, the model is turn based. Each agent calculates and performs the appropriate action for its current state, sequentially. When all of those calculations have finished, the timer sets to the next turn, and so forth. As the model increases in complexity, whether in complexity of choice or in number of agents, the length of time required for each turn increases, and the number of turns required to reach a new equilibrium also increases.

The multilevel nature adds further complexity. The groupings may operate simultaneously with the individuals, sequentially with the individuals, or may even be in a feedback loop with the individuals, a loop which needs to reach an equilibrium before the current turn can end. This becomes more complicated the more levels are introduced, and the more complex the geometry of the groupings becomes.

Current ABM platforms primarily run as a single process, and hence on a single CPU. Any attempt to schedule other tasks risks thrashing by the CPU and/or interruption to the program. CPU speed is the hard limit for the length of time required for a complete run. However, modern systems have multiple CPUs; these CPUs are left idle by most ABM platforms. Restricting the simulation to a single CPU makes scaling to the level needed by social simulations extremely problematic.

In the second place, current ABM platforms primarily use a checkerboard model, in which each

agent occupies a specific geographical space. Many of its actions will be prompted by the change in state of its neighbors in the previous turn. Empty spaces must be evaluated at each turn, as well.

However, many human interactions take place in social space. Although this often overlaps with physical space, social space is increasingly virtual (e.g. FaceBook) and proximity is less important than relationships. For example, we are likely to be more influenced by the opinion of a relative or close friend (strong relationship) than by a neighbor or fellow commuter (strong proximity). Social space groupings are also critical and increase the dimensional space of the geometry exponentially.

Social space is difficult to map onto a standard checkerboard model, because the exact number of connections between this agent and other agents is fluid. On a checkerboard, a square has 4 direct neighbors and 4 diagonal neighbors; similar calculations apply to an n-dimensional matrix. However, in a social space, the number and strength of these connections does not align easily with a physical diagram. What's more, it is a sparse matrix; there are many, many empty spaces. If we draw a diagram showing whether a given agent is connected to each of the other agents, we will find that the number of extant connections is much lower than the number of non-connections. I may know one thousand people; I may have one million Twitter followers; but there are 6 billion people in the world and I have no connection to 5,999,000,000 of them.

In a checkerboard model, each of those 5.999 billion non-connections will take up a space, and must be processed at each turn. Thus, the ABM platform needs to support a more abstract notion of space, to allow these kind of sparse geometries to be used efficiently.

This leads to the following requirements:

1. The platform needs to be portable. As computer systems and resources change, current ABM platforms become obsolete. It should not rely on use of a particular operating system or programming language.
2. The platform needs make use of all available computing resources. This means it must make use of
 - 2.1. multiple processors and cores in a system
 - 2.2. GPUs contained in a system, and
 - 2.3. networked nodes.
3. The platform needs flexibility. In particular, this means that the platform needs to pre-specify as little as possible of a simulation.
 - 3.1. The platform must support multiple user-defined geometries.

- 3.2. The platform must permit embedding of groups of agents into other agents to enable multilevel models.
- 3.3. The platform must allow use of multiple types of time, as time may flow differently for different agents (e.g. quarterly results versus weekly sales quotas versus sudden interruptions for emergency tasks of undefined length).
4. The platform needs to be customizable and extensible by the user.
 - 4.1. It must be possible to easily modify existing components to provide new or modified functionality, or to add new components such as new types of agents, new groupings, new geometries, and new measures of time.
 - 4.2. Adding or modifying modules necessarily implies some level of sophistication in the user beyond our expected case. The sophisticated user may be familiar with specific programming languages or systems. The platform needs to allow for use of the programming language, design, and run environment which is comfortable for this more experienced user. For them, use of their preferred language and environment will also add to the platform's ease of use, since they will not need to learn a new language or system environment.

3 CURRENTLY AVAILABLE ABMS

I have taken as my starting point a recent categorization of ABM platforms. In the categorization, platforms were ranked on ease of model development and scalability. Ease of model development had three levels: easy, moderate, or hard; scalability had four levels: small, medium, large, and extreme (Abar et al., 2017). In Table 1 below, I have looked at platforms which are "easy" to develop models on, and which have scalability "large" or better. While these do not directly correspond to "ease of use" and "usefulness" as explored above, they are a good proxy for limiting the search to a manageable number. This gives three possible candidates: Altreva Adaptive Modeler, SeSAm, and NetLogo. Of these, SeSAm and NetLogo are the only general-purpose platforms; Altreva Adaptive Modeler only handles financial

market simulations, so I exclude it from consideration. Table 1 shows which requirements SeSAM and NetLogo meet.

4 BUILDING A BETTER MOUSETRAP

Clearly the available platforms do not simultaneously provide ease of use and usefulness to the social science researcher. A better solution is needed. Based in my experience in computer science and real-time, distributed, and systems programming, I propose the

following design. Two principles guide this design. First, small tools that do one thing well, and can be easily combined with other tools, are more powerful and flexible than large tools that try to do everything. Second, offload functionality to already extant system components whenever possible, as much as possible, to take advantage of domain expertise beyond my own.

1. Framework management is decoupled from the interface and the agents. This is an approach which has previously been used for reasons of scalability and performance (Bosse, 2021; Collier & North, 2013).

Table 1: Requirements and Existing Platforms.

Requirement	NetLogo	SeSAM
1. Select among “typical” agents and add them to a simulation run, without needing to program them	Yes	Yes
2. Extract typical data and statistics from a run	Yes	Yes
3. Initialize simulation runs easily, including options to specify multiple runs with or without parameter changes	Yes	Yes
4. Predict run time required when possible, given a set of computing resources and past history	No	No
5. Provide typically desired visualizations of a run in progress	Yes	Yes
6. Perform the above tasks from the user interface, without any required programming	Yes	Yes
1. Portable	Yes; requires Java Virtual Machine	Yes; requires Java Virtual Machine
2.1. Uses multiple processors and cores in a system	Only when multiple runs are made; a single ecosystem is not split across processors	Only when multiple runs are made; a single ecosystem is not split across processors
2.2 Uses GPUs contained in a system	No	No
2.3 Uses networked nodes	No, although an extension can run multiple versions of a simulation across clustered systems; again, a single ecosystem cannot be split across nodes	No, although an extension can run multiple versions of a simulation across clustered systems; again, a single ecosystem cannot be split across nodes
3.1 Allows user-specified geometries	No	Yes
3.2 Supports multilevel agents	No	Yes
3.3 Allows user-specified types of time	No	No
4.1 Allows modification of included agents and creation of new agents	Yes	Yes
4.2 Allows creation of agents in any language	Native creation in NetLogo language. Bindings for some languages exist to allow external calls; others could be added	No

2. A basic communications library abstracts all the information needed for two independent programs to send each other messages, regardless of the location of the agents (on the system, on a GPU, on another networked node) in the most efficient possible manner. Messages
3. between agents will be plain text strings (encrypted in transit, but plain text to the agent itself).
4. A directory server allows programs to find each other (similar to RPC portmapper or CORBA). The directory server will also enable broadcast and multicast messages. The directory server is simply an agent with a well-known (to the ABM platform) address.
5. All agents are self-contained programs which use the library in (2) above and the directory in (3) above to find and communicate with other agents; this creates extreme modularity. The agents are normal processes scheduled by the operating system, leveraging the expertise that goes into operating systems implementation to efficiently use the CPUs and cores available. This approach also has some precedent, though not quite to the fundamental level that I propose (Collier & North, 2013). This also allows agent parameterization to be handled by command-line arguments.
6. All framework components (GUI, visualizations, geometry, time, etc.) are themselves agents, no more or less powerful than user agents.
7. A library of typical agents is provided for ease of use (visualization, data tapping, statistics creation, etc.)
8. The communications library directory server and all included agents are written in C using POSIX standard libraries for portability.
9. Library stubs in other languages are provided, to allow agent programming in any language.
10. The design builds in security from the start. Agents will need to authenticate with each other in order to communicate with each other. Messages sent between agents will be encrypted. These are done in the provided library, and the user does not need to concern themselves with it beyond very simple decisions (e.g., what is the password for the simulation?). Encryption and authentication will be handled by POSIX standard libraries.

4.1 How Does This Design Match the Requirements?

First, a word on terminology. For this discussion, "system" means the hardware and operating system components. "Platform" means the overall ABM platform, which, with this design, may have multiple independent simulations running simultaneously. "Ecosystem" is a user-defined set of agents which are able to talk to each other in a particular simulation run. Thus, the platform may run on multiple systems, and may contain multiple ecosystems. There is no *a priori* linkage between a system and an ecosystem.

4.1.1 Ease of Use

Since the user interface is completely decoupled from the ABM platform, all usability aspects are handled by an agent library. If the user wants a GUI, they start the provided GUI agent. This interface provides for agent creation, standard data extraction, run control, statistical results, parameterization, run time predictions, and visualization via provided agents. These are each individual programs, which communicate with the GUI agent to interface with the user. A preset suite of interface agents is provided, but it is possible for the user to determine how many or few of these predefined agents they wish to use.

The fact that the agents are self-contained programs using command line arguments for parameterization makes the creation of a graphic user interface very simple. The interface will not be as efficient as an integrated GUI would be; however, since the execution time of the GUI is going to be tiny compared to the execution time of the simulation runs, this tradeoff for simplicity seems worthwhile.

4.1.2 Usefulness

Because the provided programs and library are written in C using POSIX libraries, they can easily be compiled and run on any POSIX-compliant system (including Windows, MACOS, Linux, and any UNIX system (either SYSTEM V or BSD derivatives)). As standard research practices, university resources, and technical best practices change, the entire model or individual systems can be moved to any hardware desired.

The problem of scheduling CPU resources has been well studied in the computer science field; good solutions exist and better ones are being developed (e.g., (Chen et al., 2020; Kim et al., 2020; Wang et al., 2020; Cheng et al., 2016)). An ABM platform which allows the operating system to perform scheduling

tasks can take advantage of the latest updates, without itself having to change. Thus, the operating system handles internal load balancing. External load balancing among networked nodes is an ongoing field of study in cloud computing, with experimental load balancers available (Hamdani et al., 2020). I anticipate providing an agent which will handle this in the medium-term future.

The reduced size of the ABM platform core results in low overhead. Again, since agents are simply programs, a program which runs in a GPU can easily be an agent in the ecosystem like any other. Since individual agents are self-contained, they can be easily migrated to other systems. The directory server and the communications library understand how to communicate to agents which are located on the same system, on a GPU, or on other network nodes. The library uses shared memory on a single system and either IPV6 or IPV4 to communicate to other systems.

Geometry is just an agent. The user can use a predefined agent (e.g. checkerboard), or can provide their own agent to provide the geometry. This permits use of agents which can efficiently handle the sparse space of a social network. Similarly, time is just an agent. The user can use a predefined agent (e.g. event tick) or provide their own to create a more complex idea of time, possibly including a real-time clock. The ability of agents to migrate to other systems makes dynamic reallocation of computing resources simple. All of these being individual programs, starting or not starting any individual agent is trivial.

Again, the fact that every agent is a self-contained program guarantees the maximum amount of modularity. Users can easily pick and choose which agents to include in an ecosystem.

All the ease of use functions are provided by predefined agents. The user is free to modify those or provide their own to allow for more customizability. Since each agent is its own program, the user is not limited to the language of the framework for programming their own agents. The stub libraries provided allow the user to use whatever programming language they choose to implement their own agents.

5 CURRENT STATUS OF THE WORK

I have created the basic framework to allow agent communication, creation, and removal. I also have a basic checkerboard geometry agent and event tick timer agent. I anticipate that in the near future I will

be able to reproduce Schelling's (Schelling, 1971) and March's (March, 1991) models to verify correct operation, with statistics showing scalability across CPUs and additional systems.

6 FURTHER WORK

Currently very little of the standard ease of use set of agents is complete. Additionally, "ease of use" is currently based upon my conceptions, informed by a few colleagues. I intend to formally research what the simulation community considers a full set of requirements. Much of this will be ongoing; in the spirit of agile development, it is better to get something working, and then fine tune it as users interact with it and understand better their needs, than to lock in a feature set at the very beginning.

I have not implemented use of GPUs at this point; Library bindings are currently only in C, so agents can only be written in C right now. Stub libraries will need to be provided for other languages, to include at least: Java, C++, R, and Python.

REFERENCES

- Abar, S., Theodoropoulos, G. K., Lemariniere, P., & O'Hare, G. M. P. (2017). Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review*, 24, 13–33. <https://doi.org/10.1016/j.cosrev.2017.03.001>
- Bosse, S. (2021). Parallel and Distributed Agent-based Simulation of Large-scale Socio-technical Systems with Loosely Coupled Virtual Machines. 344–351. <https://www.scitepress.org/PublicationsDetail.aspx?ID=trekToYQetc=&t=1>
- Chen, J., Soomro, P. N., Abduljabbar, M., Manivannan, M., & Pericas, M. (2020). Scheduling Task-parallel Applications in Dynamically Asymmetric Environments. 49th International Conference on Parallel Processing - ICPP: Workshops, 1–10. <https://doi.org/10.1145/3409390.3409408>
- Cheng, L., Rao, J., & Lau, F. C. M. (2016). vScale: Automatic and efficient processor scaling for SMP virtual machines. *Proceedings of the Eleventh European Conference on Computer Systems*, 1–14. <https://doi.org/10.1145/2901318.2901321>
- Collier, N., & North, M. (2013). Parallel Agent-Based Simulation with Repast for High Performance Computing. *SIMULATION*, 89(10), 1215–1235. <https://doi.org/10.1177/0037549712462620>
- Conte, R., & Paolucci, M. (2014). On Agent-Based Modeling and Computational Social Science. *Frontiers in Psychology*, 5. <https://doi.org/10.3389/fpsyg.2014.00668>

- Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3), 319–340. <https://doi.org/Article>
- Gilbert, N. (2007). *Agent-Based Models*. SAGE Publications, Inc.
- Hamdani, M., Aklouf, Y., & Chaalal, H. (2020). A Comparative Study on Load Balancing Algorithms in Cloud Environment. *Proceedings of the 10th International Conference on Information Systems and Technologies*, 1–4. <https://doi.org/10.1145/3447568.3448466>
- Hughes, H. P. N., Clegg, C. W., Robinson, M. A., & Crowder, R. M. (2012). Agent-based modelling and simulation: The potential contribution to organizational psychology. *Journal of Occupational and Organizational Psychology*, 85(3), 487–502. <https://doi.org/10.1111/j.2044-8325.2012.02053.x>
- Jackson, J. C., Rand, D., Lewis, K., Norton, M. I., & Gray, K. (2017). Agent-Based Modeling: A Guide for Social Psychologists. *Social Psychological and Personality Science*, 8(4), 387–395. <https://doi.org/10.1177/1948550617691100>
- Kim, J., Kim, J., & Park, Y. (2020). Navigator: Dynamic multi-kernel scheduling to improve GPU performance. *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, 1–6.
- March, J. G. (1991). Exploration and Exploitation in Organizational Learning. *Organization Science*, 2(1), 71–87. <https://doi.org/10.1287/orsc.2.1.71>
- Schelling, T. C. (1971). Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2), 143–186. <https://doi.org/10.1080/0022250X.1971.9989794>
- Wang, Y., Li, R., Huang, Z., & Zhou, X. (2020). An In-depth Analysis of System-level Techniques for Simultaneous Multi-threaded Processors in Clouds. *Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications*, 145–149. <https://doi.org/10.1145/3407947.3407964>
- Willinsky, J. (2005). The unacknowledged convergence of open source, open access, and open science. *First Monday*. <https://doi.org/10.5210/fm.v10i8.1265>