ABMSync: An Object-Oriented Reimplementation of PS-I

Randall Fradin (fradin@wharton.upenn.edu)

Steve Pelhan (pelhan@seas.upenn.edu)

Faculty Advisor: Professor Barry Silverman
Abstract

The purpose of this project is to create a viable option for an industry standard for Agent Based Modeling (ABM) applications. By carefully planning our approach with an emphasis on an intuitive object model, readability, and flexibility, we intend to incorporate ease of use and functionality into one Python application. The current standards have generally only been able to incorporate one of these two important aspects. However, one open-source program, Political Science-Identity (PS-I) attempts to do both. As such, we will be using this program as our guide in terms of functionality and top level usability. However, the program is not without major designs flaws. Our Senior Design project will attempt to eradicate these flaws by developing a solid base from which PS-I can be reimplemented and improved. We will do this by first implementing a base on which most ABM applications can be created and then we will recreate PS-I as a module on top of that program.

Related Work

Agent Based Modeling is a computational model that runs a simulation to produce some event driven output. According to Borschchev and Filippov, "Modeling is a way of solving problems that occur in the real world. It is applied when prototyping or experimenting with the real system is expensive or impossible." Also, "a simulation model may be considered as a set of rules (e.g. equations, flowcharts, state machines, cellular automata) that define how the system being modeled will change in the future, given its present state." It is also very important to note that "simulation is the process of model 'execution' that takes the model through (discrete or continuous) state changes over time" (Borschchev & Filippov, 2004). Agent Based Modeling (ABM) differs from other well known methods of modeling, namely, System Dynamics and Discrete Event Modeling. Whereas those two types of modeling have a large, centralized "hub" that determines how the model evolves, ABM introduces a new concept of individual agents, that each have rules for how they will evolve. However, the idea of what exactly constitutes an agent is up to debate. Borschchev and Filippov say,
"there are no universally accepted definitions in this area, and people still discuss what kind of properties an object should have to 'deserve' to be called an 'agent': pro- and re-activeness, spatial awareness, ability to learn, social ability, 'intellect,' etc...but we would like to stress just one feature of agent based models: they are essentially decentralized. (original emphasis)" (Borshchev & Filippov, 2004)

This decentralization is what gives ABM its flexibility and power.

"Agent based modeling is for those who wish to go beyond the limits of [System Dynamics] and [Discrete Event] approaches especially in the case the system being modeled contains active objects (people, business units, animals, vehicles, or projects, stocks, products, etc.) with timing, event ordering or other kind of individual behavior." (Borshchev & Filippov, 2004)

What is an Agent:

Now we know that "Agent-based models are (computational) models of a heterogeneous population of agents and their interactions. The result of the micro-level interactions can be interesting macro-level behavior like cooperation, segregation, fashion, culture, etc" (Agent Based Modeling FAQ, 2008), but it may still be unclear what an agent is. Macal and North describe an agent as "a discrete entity with its own goals and behavior" and it must be "autonomous, with a capability to adapt and modify its behavior" (Macal & North, 2006). For the purpose of the programs we have studied, an agent can represent many things including, "individuals, households, firms, and nations," and can be encoded to act in many different ways, "such as changing the shared environment by harvesting or pollution, exchange of information and resources, and by imitation." Perhaps the most important feature of ABM is the rules used to define how the agents interact. However, defining what those decision rules are can be a very difficult question. The Open ABM Consortium answers it this way, "instead of assuming selfish individuals with perfect information, we often assume agents who have simple heuristics of satisfying behavior."

What are Rules:
Rules are the ways in which agents interact with each other. How the rules are defined rest heavily on the modeler, and as such, the creation of rules needs to be a very flexible process. Professor Lustick writes:

“Toolkits such as Starlogo and AgentSheets are user friendly, but can only be used to produce a limited number of pre-cooked, ‘toy’ models and are unlikely to be of actual use to social science researchers focused on theoretically or empirically meaningful problems. SWARM, RePast, and ASCAPE, which in principle could be used to produce models of real experimental interest to social scientists, require users to be proficient in either C languages or Java” (Lustick, 2002)

His reason for creating PS-I was to avoid a heavy dependence on C or java programming in order to efficiently and effectively create these simulations. However, his creation, PS-I, relies heavily on its own version of pseudo code that is a major obstacle for a casual modeler.

Our version will attempt to create the same flexibility and power that Lustick’s PS-I has, with much less dependence on coding. By allowing users to define their own agents, agent classes, identities, and rules, we allow for increased flexibility. However, our design provide for very clear declarations of attributes, class type, and identities, without forcing the user to code the entire model in one chunk, and in the end will require custom code only for agent evolution rules. In our initial implantation we will define a model file specification that the user must follow to define all aspects of the model, but as the GUI is developed it will be simple to create a form for generating a model with the desired agents, classes, identities, and attributes, with the only the evolution rules requiring free text entry and a bit more technical knowledge. A full knowledge of Python will not be necessary however, only the ability to create conditional statements and to specify which attributes are evolving if the given condition is met, while still allowing those users with more programming-savvy to include more complicated procedures. Such a form could even include rule templates that take a given set of parameters and automatically add the desired functions to the model specification. This recognition that evolution rules are the only part of a model complex
enough to require a more sophisticated method of specification is what sets our program apart from simpler implementations that don’t allow that level of freedom, and more complicated implementations that require programming knowledge to specify all parts of a model.

**Technical Approach**

Agent based modeling in the most general sense refers to a computational model in which autonomous individuals exist or are created in a network, and experience a series of actions and interactions over time, with a goal of observing the effects the agents have on the system as a whole. PS-I has embedded in it certain assumptions about the nature of these agents, how they are networked, and how they evolve over time. For example, PS-I assumes that agents exist on a single discrete two-dimensional field of a predetermined finite size, that every location on the field is occupied by an agent, and that agents cannot move, be created, nor be destroyed during the simulation. This graph implicitly represents a directed graph, since each agent is influenced by its neighborhood of agents which exist within a certain sight radius on the field. Each agent has a class which gives it certain attributes, such as sight radius, inactive, immutable, and influence. Some of these attributes can be changed for individual agents regardless of class, while others must stay fixed for an agent of a given class. PS-I also contains a notion of identities, which can represent any number of real-world characteristics such as political affiliations, religions, or diseases. The set of identities is specific to a particular model. Each agent has a repertoire, which consists of a single activated identity and a cache of subscribed identities. This repertoire evolves over time based on the activated identities and other characteristic in an agent’s neighborhood. The goal of PS-I is generally to assess how the distribution of identities among agents on the field will change over time given the initial distribution and values of certain parameters that affect the model, such as particular thresholds that govern the evolution of agents’ repertoires. This is a high-level overview of how PS-I functions. There is plenty of other functionality currently in PS-I, such as the notion of regions on the field, rules that govern how agent classes change over time, attributes specific to particular identities, and certain statistics that are tracked over time. An example statistic specific to a single agent would be agent tension, which is the
number of other agents in an agent’s neighborhood that have an activated identity different from that of the central agent. An example identity-specific statistic is the total number of active agents activated on a particular identity. It is clearly evident that PS-I is not a general ABM application, and while implicit assumptions can be worked around, a truly extensible ABM application should try to minimize the number of assumptions about the nature of the models it is being used for.

The primary motivation for the creation of ABMSync was to have a replacement for PS-I that could be more easily understood and extended. PS-I contains certain implicit assumptions about the nature of agent-based systems which come from the specification for the Identity-Repertoire model type, so with the goal of extensibility in mind we chose to separate the program into different layers so that a more general layer which we call ABMSync would contain only a few of these implicit assumptions, and a second layer, which in the example we provide is called IDRep, would implement the rest of these assumptions to provide functionality similar to PS-I as it stands today, with a model specification file providing a link between the two layers.

There is clearly a tradeoff when trying to allow for extensibility, both with performance and with simplicity of implementation. Allowing for too many possibilities in a model will create an unnecessarily complicated model specification and will allow fewer steps in a model to be standardized. We believe that ABMSync is generalized sufficiently to allow for relatively painless future modifications as the theories underlying PS-I change. We kept the notion of identities, assuming that the primary purpose of a model is to observe the behavior of the distribution of identities among agents over time, though this is not actually necessary since identities are not necessary to creating a meaningful model. Additionally we now allow for multiple groups of identities, so that identities within a group may be treated equally, though not necessarily across groups. This is represented in the model with a dictionary mapping of group names to lists of identity instances. This could be useful if you wanted to model political affiliation and religious views simultaneously, for example. The modeler can create any number of identity attributes that may affect agent interaction and assign different values to different identities, which are stored in identity instances with a mapping of
attribute names to values. The application still assumes that agents interact on finite-sized fields, but here we allow for any number of fields in a single model, which can each contain any number of dimensions. Each field has its own regions, represented as mappings of names to lists of locations. Each field also maps locations to agents, so it is no longer necessary for a field to have all locations filled. It is not even necessary that each agent exist on each field. Agents have been generalized to have a class, and an arbitrary mapping of attribute names to values. Agent Classes have a name, and a mapping of some subset of agent attributes to values, so that when an agent is assigned to a class, those attributes that are defined by the class overwrite the attribute values of the agent. It is not necessary for the agent to maintain any of these values after class assignment has occurred.

When a step in the model occurs, a series of functions are used to process the state of the model and make changes based on certain rules. These functions come from the second layer of the system, which specifies the type of model that we are running and contains more specific rules of the type that we see currently in PS-I. As previously mentioned, the model specification file contains among other things the pointers to the appropriate functions in the second layer. This layer is contained in one or more modules, and can contain any number of several types of functions. The first thing that the model does in a time step is call all preprocessor functions specified by the model file, which take the entire model instance as input and perform some transformation. For example in IDRep, one preprocessor function is used to randomize the bias attribute of all identities before each time step. Next the model iterates over all agents, and for each agent calls all applicable rules, which are functions that take the agent as an input and govern how it will evolve. The rules are separated into groups, so that each group will stop after the first successful evolution, indicated by a return value of true. Groups can be specified to apply only to a certain agent class, or to apply to all agents. Each agent acts based on the state of all agents at the end of the previous time step, so all changes made during the processing of these rules are buffered using a set of functions contained in the model object, and at the end of iteration over all agents the buffers are cleared out and all changes are applied simultaneously. IDRep contains two rules, one that determines how an agent’s activated identity and cache change, and another that determines
When an agent will be assigned a new agent class called border. Next the model calls all postprocessor functions, which are similar to preprocessors, though IDRep currently requires none. Next the model recalculates all statistics. Statistics are a set of values that the model writer specified he wants to keep track of, which can be values particular to each agent, each identity, or the entire model. The second layer module will contain a set of functions that can be used to calculate statistics, so that the model writer can decide which ones he wants to see and specify a function pointer for each. Statistics may depend on each other, and other functions may depend on statistics as well, thus the second layer module can contain function calls to the model object which specify these dependencies so that an error will occur if required statistics are not included by the model writer. For example, in IDRep, the total_tension statistic requires the agent_tension statistic to be calculated. The second layer module will also specify which agent attributes, identity attributes, classes, identity groups, fields, regions, and random sequencers are required for the layer to function, so that the model will preemptively raise an error if a required model component is not included by the modeler. The only other type of function included in this layer is the neighborhood function. ABMSync maintains the notion of agent neighborhoods, but provides no specificity for how they are computed. The modeler specifies functions that will compute the neighborhood for a particular agent on a particular field based on the agent’s attributes, and will also specify which attributes this function depends on so that neighborhoods will only be recalculated when these attributes change. This recalculation occurs when buffers are cleared after the rule step if necessary.

Different modules can be created to allow for different types of models, each containing different rules, pre and postprocessors, statistics, and neighborhood functions. The final part of the system is the model specification. Right now ABMSync contains functionality to load a model specification from an XML file, since XML is a fairly standardized file format for representing tree-structured data. This is the part of the system that allows for different models to be created, and since it requires no computer programming knowledge it should be simple for researchers in other fields to created model specifications, especially if a model editor with a GUI is developed to edit the XML file. The file contains items such as the field
specifications, agent and identity attributes, agent classes and attribute values, identity groups with identities and attribute values, the names of modules to include and pointers to the different functions needed to run the model in those modules, and a pointer to a snapshot file. The snapshot file is comma-separated file which contains at the very least an initial set of agent classes, and then optionally locations for each agent and initial attribute values. ABMSync also contains functions to write out the currently loaded model to an XML model file and snapshot file. Loading these files back in and running them would result in the same outcome as if the simulation had continued uninterrupted from the time the model was written out. Random number generators in the model are seeded with a specified value initially and counted as they are used, thus even a theoretically nondeterministic model can be recreated deterministically and continued as if it had been uninterrupted if it is saved and reloaded.

We chose Python as our implementation language in order to make the code as readable, portable, and useful as possible. Python is a high-level partially object oriented language that is widely used for research and is generally considered one of the more readable and easy to use scripting languages. Python is especially well suited to our need for dynamic loading and storage of function pointers that are specified by strings in a model file. A high-level interpreted language creates an obvious performance tradeoff compared to PS-I’s C implementation, but the cost is not prohibitive and the benefits of stability and manageability are enormous.

**Conclusion**

ABMSync implements all of the features that we initially set out to create. It's modularity allows for an incredible range of other types of models to be created on top of it. In addition, we recreated the IDRep model used by PS-I. While the design aspect of the project was much harder then we initially expected, we were able to overcome some major complications and surpass the limitations of the other major industry standards that exist for ABM applications. However, the program can still see a could deal of improvement by having additional graphical applications built on top of it. Although this was outside the
scope of our project, the program could benefit from a model viewer and editor. With these tools in place, ABMSync could eventually become one of the foremost programs used for ABM applications.

Bibliography


The Agent Based Modeling FAQ is a simple list of frequently asked questions put forth by the Open Agent Based Modeling Consortium, "a group of researchers, educators, and professionals." Their purported goal is to "improve the way we develop, share, and utilize ABM." The website provides useful links to ABM journals, as well as some basic insight into ABM and the power that it represents. The FAQ itself however, is very limited and provides mostly an introduction to ABM and reasons for its use. It fails to detail any major flaws with ABM or describe any situations in which a different modeling approach might be better.


The Authors give an in depth introduction to System Dynamics, Discrete Event, and Practical Agent Based Modeling. They take time to examine each individual, and to compare the three approaches and speak about relative strengths and weaknesses. Their approach shows that Agent Based Modeling, while powerful, is only useful in certain situations. The goal of the paper is to inform users of the new tool of ABM and allow them to add it to their repertoire for problem solving and modeling. This paper is very skillful at detailing the type of situations that can and should be modeled using ABM and as such it is useful for describing why ABM is necessary in the first place. In addition, the authors are able to make valid
arguments as to when ABM should not be used, and thus, when the power of ABM, and subsequently our program, are limited.


This paper, published on JASSS (The Journal of Artificial Societies and Social Simulation), is written by professor Lustick and is an attempt to introduce the Agent Based Modeling community to the new program PS-I. In his paper, Lustick attempts to convince readers that his program is better than other industry standards and goes on to detail exactly why, namely citing the programs ability to define and simulate an much larger section of possible scenarios, while at the same time maintaining the high quality of performance necessary for ABM to be possible. Professor Lustick makes a very good case for why the program PS-I should be used, however he fails to point out the major flaws of the program. As such, this paper serves as a good tool to describe why PS-I makes an ideal platform for us to base our new tool on, however, we must carefully watch for Professor Lustick's personal bias and try to remove it from our analysis.


These are a set of slides published by ANL as a brief introduction to Agent Based Modeling. The slides talk about the basics of ABM ranging from what an agent is to what makes a rule. As they discuss, both of these definitions are very fluid, and differ from one model to the next. Agent Based Modeling, as they say, is a modeling system where each individual agent reacts to the other agents, and they talk about the power that this type of modeling has. This set of slides serves as a very basic level of introduction to ABM and what is has to offer the
modeling community and does not go into any depth concerning different types of modeling tools. As such, it serves only as a reference to help describe what Agent Based Modeling is.