Invited paper presented at *Complex Systems and Policy Analysis: New Tools for a New Millennium*, Arlington VA, 27 September 2000. RAND Corporation Science & Technology Policy Institute.

A General Framework for Representing Behavior in Agent Based Modeling

William N. Reynolds and David S. Dixon
Least Squares Software LLC
PO Box 91405
Albuquerque, NM 87199

grendel@leastsquares.com
ddixon@leastsquares.com

Abstract: A Specification for a simulation framework for Complex Adaptive Systems (CAS) is given. The desired framework is flexible enough to allow the technical expert to build simulations for a broad variety of CAS and accessible enough to be of use to the domain expert. The BASP modeling framework and the Archimedes simulation platform are presented and shown to satisfy the framework specification.

Keywords: Complex adaptive systems (CAS) Agent-based modeling (ABM), Military simulations, Simulation frameworks, Simulation methodologies.

1. Introduction

The study of Complex Adaptive Systems (CAS) is today one of the most exciting fields of research in basic and applied science. Political systems, stock markets, organization dynamics, economies and battlefields are all examples of systems that are poorly understood and profoundly consequential in our day-to-day lives. Predicting or even developing insight into their behaviors is a problem of tremendous practical importance.

2. Specifying a Computational Tool for Simulating Complex Adaptive Systems

Unfortunately, it is very difficult to achieve insight, and often impossible to make predictions about a complex system. Analysis, (the dictionary definition of which is "to separate into its constituent parts" [1]) is difficult or impossible on systems with nonlinear *emergent* properties. That is, systems which depend on the interactions of the constituents and which cannot be studied if the constituents are separated.

Computer simulation is a natural avenue for the study of CAS and some progress has been made in this direction [2]. Nevertheless, the difficulties here are also formidable. One the one hand, robust, accurate, *facsimile* simulations of CAS often strain available hardware and software technologies to their breaking point, and the technical, administrative and fiscal hurdles required to carry these projects through can be insurmountable. On the other hand, while progress has been made using *toy model*, or *thinking tool* approaches [3], in which skeletal representations of the constituents of a complex system are simulated on the computer, their results are tarnished by questions of fidelity and the model's inability to make quantitative predictions. Often a toy model approach is employed to suggest directions to proceed using a more complete facsimile model. While the search space for the facsimile model is reduced, the thinking tool may do little to improve the facsimile's fidelity or predictive accuracy.

By their very nature, a CAS encompasses tremendous amounts of information, some subset of which is important for determining the system's behavior. Developing a good understanding of these factors and how they affect the system is typically a specialty in itself (*e.g.* economics, political science, military science or stock trading). Domain experts who study these systems are typically the end users of software tools developed to study the systems.

A central problem in computational approaches is that the domain expertise and the technical expertise often reside in different places. The domain experts (or analysts) know what capabilities they would like a simulation to have, but do not have the technical expertise to implement the capabilities, nor do they have a good understanding of what can and cannot be done in simulation. Similarly, the technical experts can implement behaviors in the simulation, but do not have the domain expertise to specify that behavior.

This *communication gap* (or rather *communication chasm*) between the domain and technical experts lies at the heart of many unsatisfactory forays into CAS studies. To be successful, a simulation must address this problem. The domain expert must be able to easily understand, specify and modify the system's behavior. He must further be able to easily direct the technical expert on implementing new technical features that do not exist in the system [4].

This capability is much broader in scope than simply designing new graphical user interfaces (GUIs) for the system. If the underlying system is too abstract or complex for the domain expert to easily map onto his problem space, then it is of little practical use, no matter how elegant the interface. Further, the implementation of a GUI often requires a level of effort equal to or exceeding that of implementing the underlying simulation. The cost of implementing new interfaces for each new problem domain can, even for existing and proven simulation frameworks, lead to the demise of the project.

A generalized CAS simulation platform must have the following capabilities:

- **Flexibility**: The system must be flexible enough to simulate a variety of problem domains, ranging from stock markets to battlefields. There should be a well-defined migration path from one problem domain to the next that leverages the core technical formalism.

- **Accessibility**: The system must be accessible to both the domain and the technical expert. Ideally, the system would be completely under the control of the domain expert, who could introduce new features to the system easily, without any support from the technical expert. In the absence of this, the domain expert should have wide latitude in specifying the system's behavior without having to bring in the technical expert. The system should provide a framework for discussion between the domain and technical expert that "crosses the chasm" between the two disciplines.

- **Toy to Facsimile Scalability**: The system must be capable of representing the system at different levels of fidelity. It should be possible for both the domain and technical experts to quickly develop toy model prototypes. There should be a well-defined pathway for scaling up the fidelity of these toy models to full facsimile models. The system should be able to easily accommodate arbitrarily complex components if the domain expert deems it necessary and the technical expert deems it feasible.

3. The BASP ABM Modeling Framework

BASP (Behavior/Action Simulation Platform) is a framework that addresses the above requirements. Archimedes, a simulation implemented in Java for the United States Marine Corps, is an instance of BASP as an Agent Based Model (ABM) military simulation. ABM has been shown to be a robust and flexible approach to modeling CAS [5]. Obviously, there are systems for which ABM is inappropriate, and in this case the BASP framework would not be applicable.

BASP design is motivated by two principal questions:

- What are the core software components that will support model design and implementation by domain experts with minimal technical support (i.e. “coding”)?

- What is the best way to make these components effective and robust for both simple, toy models and full-scale facsimile models?

The first question is answered by examining the way in which analysts discuss their problems. Simplifying and quantifying autonomous agents leads to a natural division between a set of beliefs based on internal state and interactions with other agents, or *behaviors*, and the expression of those behaviors in the form of *actions*. This is the fundamental design philosophy of BASP. It becomes clear quickly, however, that there are only so many

general behaviors and actions that are common to all models, and then more specialized behaviors and actions are required. Fortuitously, the solution to this problem can be incorporated into the answer to the second question.

This combination of design philosophy and implementation technology is intended to give BASP the levels of flexibility, scalability and accessibility required of a generalized CAS simulation. The following paragraphs discuss some of the features in greater detail.

The answer to the second question is *aspects* – specialized software modules implemented with the BASP interface for easy and rapid integration into the BASP framework. The BASP framework imposes a lexicon on the domain expert and programmer alike, closing most of their communication gap and speeding implementation by providing much of the basic code for aspect interoperability. Scalability comes the way in which aspects, like all components of BASP, can represent behaviors or actions in very general terms of in minute detail, and can be layered to produce increasing complexity.

The BASP architecture represents both agents and the interactions between agents as software objects. Interactions between two agents are called *connections*, interactions between three agents are called *triples* and interactions involving larger numbers of agents are called *nexuses*. The system uses a template formalism for constructing these objects - the analyst builds template objects that represent types of agents, connections, triples and nexuses. When a simulation is to be run, the template objects are used to generate instances of the agents and the interactions. The interactions are directional. For example, between two Agent Templates, the analyst may specify two Connection Templates (in which each Agent Template alternatively plays the role of the *owner* and *partner*). Between three Agent Templates, the analyst may specify up to six distinct Triple Templates. Associated with these templates are Variable Templates, which represent the behavioral state of an object, and Aspect Templates, which represent a physical capability of an object. Specific instances of these templates (called Agents, Connections, Triples, Variables and Aspects), interrelate as depicted in Figure 1.

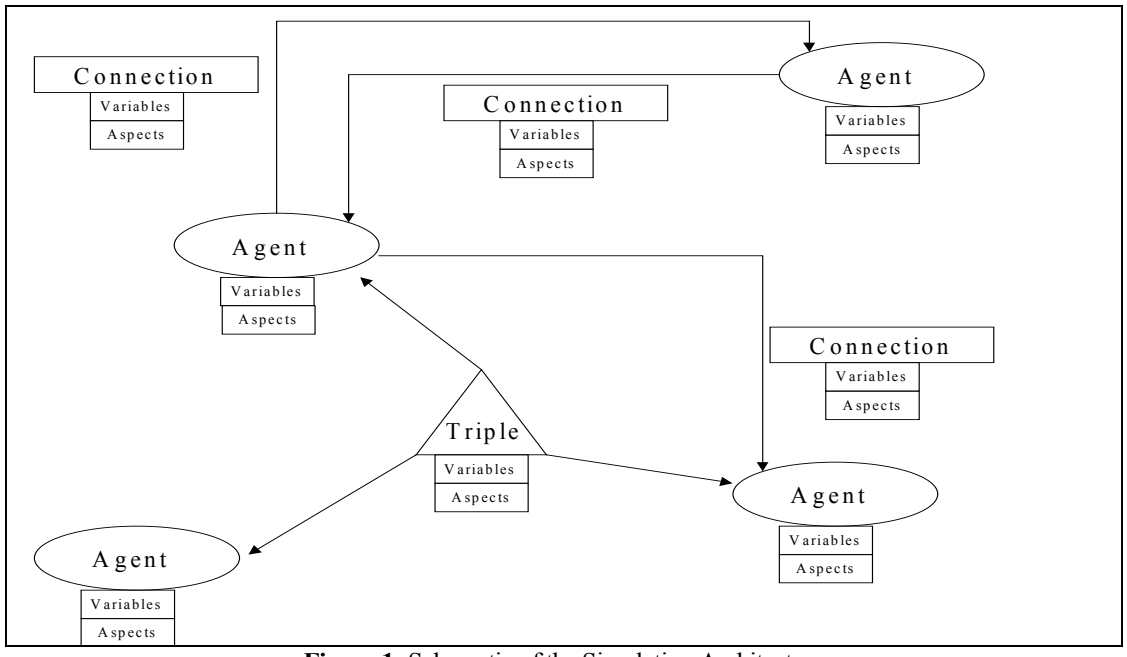

Figure 1: Schematic of the Simulation Architecture

BASP uses *heavy* agent and interaction objects; which is to say there is a great deal of breadth and flexibility in specifying the capabilities of an object. Fundamentally, the attributes of an object are divided into two regimes: *actions* and *behaviors*. Actions are the physical capabilities possessed by the Agent or interaction, while behaviors

are decisions that are made by the Agent or interaction, which may cause actions to be performed. Every Agent and interaction is endowed with a set of quantities that represent its state. This state consists of an arbitrary number of parameters. The state is divided into two domains:

- *Physical State*: This is the action part of the state. It represents the relationship of the Agent or interaction to the physical world in which the simulation is embedded. Examples of physical state include the position and velocity of an Agent in a physical environment, the distance along a Connection separating two Agents in that environment, or the capital held by an Agent in a trading simulation.

- *Behavioral State*: This represents the intent of the Agent or interaction, which may or may not be associated with physical reality. For example, the "Movement State" of an Agent embedded in a physical environment is the Agent's desired speed. The physical speed will reflect the realities of the local terrain, the Agent's movement capabilities and other physical effects. On a Connection, the "Perceived Threat" would represent the degree to which the Connection's owner Agent is threatened by its partner Agent, regardless of the actual threat that the partner poses the owner.

The behavioral state of an Agent or interaction is represented as a collection of Variables. These are instances of analyst-defined Variable Templates, one of which may, for example, divide “Distance” into five fuzzy categories, such as “Here”, “Very Near”, “Near”, “Far”, and “Very Far” representing a range of one thousand meters.

The physical state on an Agent or interaction is represented by software objects called Aspects. There is an Aspect *class* associated with every physical action in the simulation, (*e.g.* movement, combat, communication, trading, voting). Aspects encapsulate any environment-specific and model-specific code and data associated with the performance of actions. The analyst controls Aspects in two ways: by choosing which Aspect Templates to associate with an Agent or interaction template, and by invoking Aspect actions via changes in the behavioral state. New Aspect algorithms are incorporated merely by making the appropriate code available to the simulation, allowing the analyst to select the new Aspect Template.

Aspects also transmit information from the physical state to the behavioral state. For example, an Aspect on a Connection updates the behavioral variable "Distance", which, based on the analyst’s definition, may translate 750 meters into the category “Far”.

Aspects provide the “framework for discussion” required for *Accessibility* as described above. When a system needs a new physical capability, the analyst directs the technical expert in the design and implementation of a new Aspect. In this way, the system is easily extendable within a problem domain, and new problem domains can be addressed through the implementation of a base set of Aspects for that domain. Thus, the modular Aspect mechanism also satisfies the *Flexibility* and the *Toy to Facsimile Scalability* requirement above.

Each Aspect Template provides a graphical editor for specifying its state. For simple aspects, the technical expert simply specifies a list of parameters and behavioral variables that the Aspect will access, and the interface is generated automatically. For more sophisticated Aspects, the technical expert may implement a custom interface.

Aspects allow the system’s physical actions to be controlled by the behavioral variables, and feed information from the physical to the behavioral domain. With one important exception, Aspects do not impose any dynamics on the Behavioral State. The exception is the *Behavioral Aspect* class. This class of Aspects provides dynamics for the behavioral state. Like all Aspects, it is a software object, albeit a rather sophisticated one. The interface to this Aspect Template is one of the principle means whereby the analyst may control the simulation. This is done by specifying *behaviors*, which are simply dynamics on the behavioral state. Like other Aspects, there can be multiple types of behavioral Aspects, using different formulations of behavioral dynamics. The Archimedes implementation has one behavioral Aspect, which uses a natural language-like *fuzzy logic* mechanism for specifying behaviors. As an example, Figure 2 is the editor for the behavioral (rule) Aspect Template. These rules enable a group of Agents to maintain their relative distance to one another. The natural-language-like quality of the rules shown in Figure 2 arises from the component nature of BASP. The domain expert has chosen “natural”

names for the agents, variables and variable categories. The fuzzy logic rule aspect provides a natural means for relating agents, variables and categories, and for interpolating between categories.

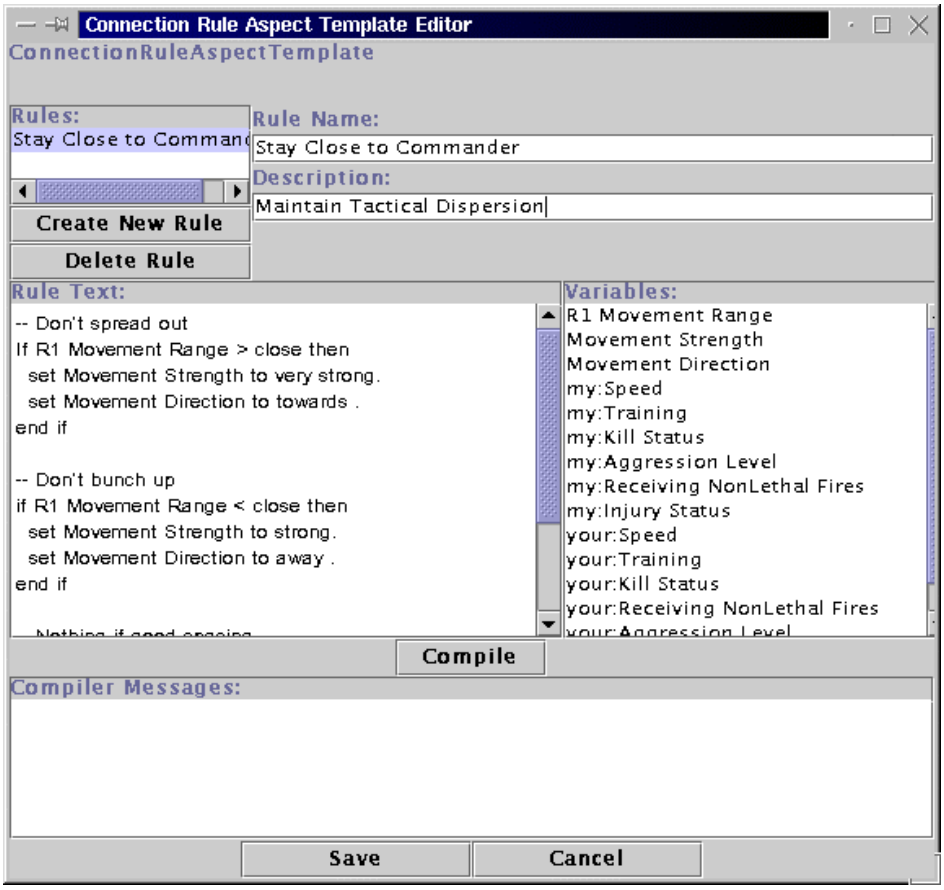

Figure 2: Editor for the Behavioral Aspect

The mechanism of the behavioral Aspect satisfies the rest of the *Accessibility* requirement above. The behavioral Aspect is designed to be the principle interface to the simulation for the end-user analyst. The analyst can quickly specify or modify the behaviors of the entities in the simulation. Using the aspect and its interface, the analyst can easily understand, specify and modify the system's behaviors, which are couched in familiar technical terminology.

While the fuzzy logic behavioral Aspect has proven to be very general and very useful by a wide class of analysts, it is not the final word. For problem domains where experts require a different formulation of behavioral dynamics - for example, stochastic dynamics, a new behavioral Aspect could be implemented on fairly short order. The only requirement is that the dynamics be defined on a set of behavioral variables. Hybrid simulations, which use different behavioral dynamics on different Agents and interactions, are then easily realized.

Archimedes consists of the BASP framework and a set of Aspects that endow Agents and their interactions with the capability to maneuver and fight on a computational battlefield. Figure 3 shows a snapshot of an Archimedes run.

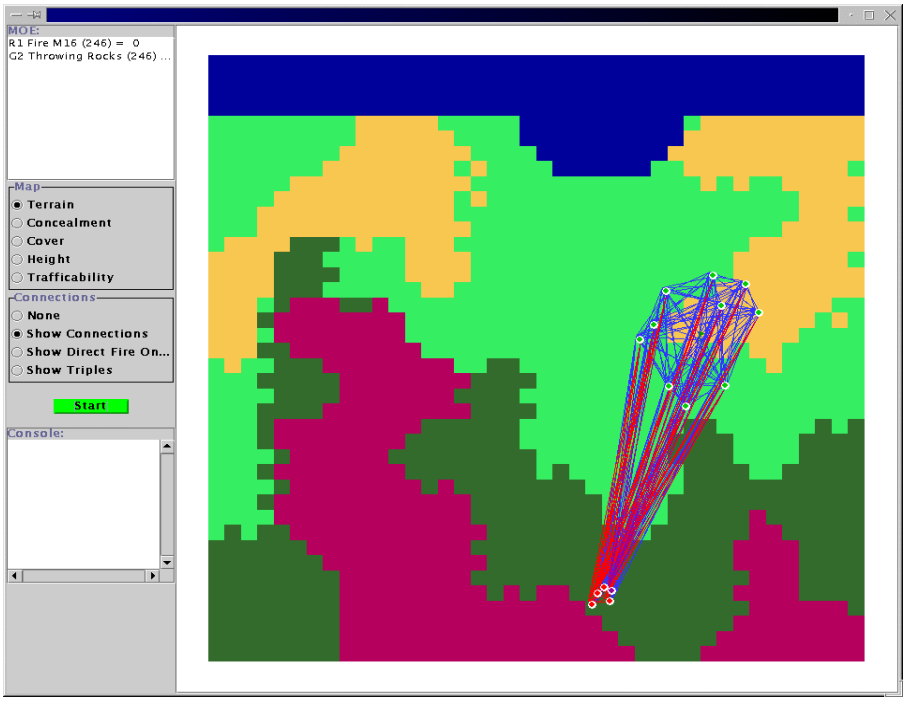

Figure 3. An instance of Archimedes, showing Agents and Connections.

4. Applications

The first simulation implemented with BASP, Archimedes, was developed for the Marine Corps to investigate the applications of ABM to combat simulation. The initial problem chosen for study was the effects of discipline on the outcomes of reconnaissance missions. To develop the simulation, several Marine reconnaissance experts were interviewed to determine what physical processes were most important in reconnaisance missions. As a result of these interviews, two custom Aspect Classes were developed, a *ReconnaisanceMovementAspect*, which allowed agents to adjust their posture and stealthiness while moving and a *DetectionAspect* which implemented a sophisticated detection algorithm (developed in a previous study by the U.S. Army). Both of these Aspect were built with a strong dependence on a *discipline* behavioral variable, which moderated the agents' abilities to move stealthily and detect. The Aspects were incorporated into the simulation and a study was executed.

These initial applications show the power of the BASP platform. Domain experts are intimately involved in the development of a given simulation and drive the development process. Rather than being hamstrung by the limitations of the platform, Domain Experts are able to dictate exactly what they would like to see in a given simulation, and well-defined, targeted software development is performed in support of these requirements. Extending the system to new domains is a straightforward and well-defined process that builds upon (rather than discarding) previous effort.

5. Conclusion

The problem of making a CAS simulation that is useful and accessible for both technical and domain experts is a difficult task. While some progress has been made to make the simulation process easy for the technical expert [2], little effort has been made towards making simulations accessible to the domain expert [4]. Further requiring that the framework be general enough to address different problem domains makes the problem harder still. An initial attempt to address these problems is proposed in the BASP modeling framework, although it is clearly not the final word on the subject. The success the approach has achieved with the Archimedes combat simulation suggests that, at the least, some of the questions asked here, about what a CAS simulation should be, are the right ones.

6. Acknowledgments

This work was supported under contract with the Marine Corps Combat Development Command's Project Albert, Contract No. M 00027-96-D-0012. The authors wish to thank Capt. M. Leonardi and Mr. A. Sawyers for useful conversations. The authors would also like to thank Project Albert, in particular Dr. A. Brandstein and Dr. G. Horne for the technical vision that has allowed this project to be realized.

7. References

1. http://work.ucsd.edu:5141/cgi-bin/http_webster

2. http://www.econ.iastate.edu/tesfatsi/acecode.htm provides a good survey of available ABM modeling technologies.

3. We are indebted to Nigel Gilbert for introducing us to the terms *Facsimile* and *Thinking Tool*.

4. The PIM/Paracell system by Flavors Technology espouses a similar viewpoint for distributed Agent based control systems. http://www.flavors.com/HTML%20Presentation%20folder/index.htm

5. http://www.econ.iastate.edu/tesfatsi/sylalife.htm

6. Hodges, James, and James Dewar. 1992. Is It You or Your Model Talking? A Framework for Model Validation, R-4114-AF/A/OSD, RAND, Santa Monica, Calif.