From Conceptual Models to Agent-based Simulations: Why and How

Swaroop Vattam[a], Ashok K. Goel[a], Spencer Rugaber[a], Cindy Hmelo-Silver[b], Rebecca Jordan[c]

[a]*School of Interactive Computing, Georgia Institute of Technology, Atlanta, USA*

[b]*Graduate School of Education, Rutgers University, New Brunswick, New Jersey, USA*

[c]*School of Environmental and Biological Sciences, Rutgers University, New Brunswick, New Jersey, USA*

Abstract. The core problem we address in this paper is how to take a declarative conceptual representation of a complex system and produce an agent-based simulation of that model. In particular, we describe a computational technique that takes Structure-Behavior-Function models of complex systems and simulates the behavior of the modeled system in NetLogo, an agent-based programming and simulation environment. This technique has been implemented in an interactive learning environment to promote complex systems learning among middle school students.

Keywords: Complex systems, interactive learning environments, modeling and simulation.

Introduction

A Structure-Behavior-Function (SBF) model ([1], [2]) of a complex system declaratively represents its functions (i.e., the observable inputs and outputs), its structure (i.e., the components and connections), and its behaviors (i.e., the causal processes that compose the functions of structural components into the functions of the system). We are exploring the utility of this knowledge representation for helping middle school students learn some of the core ideas related to complex systems (e.g., [3]). Our basic approach is to focus students' critical inquiry skills around building SBF models of systems, allowing them to express their understanding in a visual SBF knowledge representation language that not only makes explicit some of the targeted core ideas related to complex systems but also serves as a stimulus, scaffold and coordinator for various learning interactions within and among students.

Once a student develops a SBF model of a system, the model has the potential to become a shared construct, enabling teachers and peers to assess the student's understanding of the modeled system and for providing relevant feedback. However, having this social channel as the only form of evaluative support has its disadvantages. The students' tendency to wait until they have a relatively finished product before they are ready to share it, coupled with the greater turn-around time for a response to arrive from their teachers or peers prevent students from getting timely and intermediate feedback as they construct their models.

We have tried to address this problem by coupling the SBF representation language with a simulator capable of animating the SBF model. In particular, we have developed a computational technique that takes an SBF model and simulates the behavior of the modeled system in NetLogo ([4], [5]), an agent-based programming and simulation environment. The advantages of this approach are that students can (1) visualize the behavior of the system they modeled, (2) get feedback about their models by comparing the behavior of the system they modeled with either an expert's simulation or a real-life version of the system, and (3) get feedback at intermediate stages of their model development.

SBF models to NetLogo Simulations

Motivated by the larger educational issue of providing appropriate visualizations of and timely feedback on students' evolving understanding of complex systems, the core problem we address in this paper is how to take a SBF model of a system and produce a NetLogo simulation. HYPERION is a computational architecture for achieving this. HYPERION deals with three kinds of computational constructs (1) instance SBF models, (2) abstractions over instance SBF models called "behavior patterns", and (3) the SBF-NetLogo compiler. In order to simulate SBF models, they have to be transformed into NetLogo programs. This is done by the SBF-NetLogo compiler. In order to simulate a particular SBF model instance, the constructs of the SBF modeling language (e.g., states, transitions, components, substances, properties, etc.) have to be mapped to constructs of the NetLogo programming language (e.g., world, agents, breeds, topology, colors, shapes, movement, etc.). The compiler produces as output a program which can be loaded into NetLogo and executed.

The process of compiling is complicated by two factors. First, the mapping rules for the compilation process are context dependent and cannot be completely determined *a priori*. For instance, although we know beforehand that model elements of a certain category in SBF models (e.g., *components*) are mapped to one or the other programming primitives in NetLogo (e.g., *agents* or *breeds)*, the compiler cannot determine in a context independent way, whether an occurrence of a component, say *Fish*, should be declared as an *agent* or a *breed* in the NetLogo program. Second, the compilation process has to deal with an impedance mismatch arising from how the changes that a complex system undergoes are represented in the SBF and NetLogo frameworks. In SBF, a process that accounts for those changes are approximated (simplified) to a deterministic and discrete (event-based) representation. NetLogo, however, is designed to simulate continuous, stochastic processes. Therefore, in compiling SBF models, a NetLogo-suitable mapping function(s) has to be produced that accounts for the discrete event changes of the SBF model. To mitigate these challenges, we propose that the required mapping rules and function(s) can be fixed beforehand for a class of SBF models that share deep similarity. That is, if the mapping rules and function(s) for a particular class of models are known, any SBF model which is an instance of this class can be simulated by instantiating the class' mapping rules and function(s). To capture this notion of a class of SBF models, we introduce the notion of *behavior patterns*.

Behavior patterns are generic abstractions over many instances of SBF models that share a common deep structure. For instance, models of ecosystem containing the carbon-dioxide cycle and the nitrogen cycle are both instances of the *"production-*

consumption-cycle" behavior pattern. The role of a behavior pattern is to encode knowledge common to a set of similar SBF model instances and use that to knowledge to fill in the gaps during the SBF-NetLogo compilation process. In other words, behavior patterns set up expectations for SBF model instances that belong to that pattern. They suggest what information a model belonging to a certain pattern should contain. For example, the *"production-consumption-cycle"* pattern suggests that there must be one or more substances that are being produced and consumed (e.g., ammonia, nitrite, nitrate, and food in the ammonia cycle model). This pattern suggests that for every substance there has to be a producing agent and a consuming agent (e.g., fish consumes food and produce ammonia, nitrosomonas bacteria consume ammonia and produce nitrite, etc.). Since this pattern is related to a set of semantically related models, specifically ecological models, the domain semantics suggests the mapping rules to be included in the pattern (e.g., producing and consuming entities are collectives and hence the occurrence of *fish* or *plants* in SBF should be mapped to *breeds* in NetLogo). Similarly, the domain semantics also suggest the mapping function(s) to be included (e.g., mathematical relationships governing the rates of production and consumption).

This architecture raises a number of issues. First, how are particular instances of SBF models associated with behavior patterns. At present, we assume that this association is done manually by the students. Second, if the SBF model construction by the students are completely unconstrained, this may lead to models that may be internally inconsistent or too impoverished for the compilation process to succeed. To address this issue, we provide students with model templates rather than having them construct models from scratch. But this does not guarantee internal consistency or sufficient completeness, lacking which the compilation fails. This failure of compilation can itself serve as a feedback to the student that something is wrong with the model. These issues point to interesting avenues for further theoretical exploration.

This article presents a computational technique. Further studies are required to determine the soundness of the technique and its efficacy in addressing the educational need it was designed to address.

Acknowledgements: We thank Vivek Menon for his contributions. This research is supported by NSF ALT Grant #0632546 (Learning About Complex Systems by Constructing Structure-Behavior-Function Models in Middle Schools).

References

[1] Goel, A., Gomez, A., Grue, N., Murdock, W., Recker, M. & Govindaraj, T. (1996) Towards Design Learning Environments: Explaining How Devices Work. In *Proc. International Conference on Intelligent Tutoring Systems*, Montreal, Canada, June 1996.

[2] Goel, A., Vattam, S., & Rugaber, S. (2009) Structure, Behavior & Function of Complex Systems: The SBF Modeling Language. To appear in *International Journal of AI in Engineering Design, Analysis and Manufacturing*, Special Issue on Developing and Using Engineering Ontologies.

[3] Hmelo-Silver, C., Jordan, R., Demeter, M., Gray, S., Liu, L., Vattam, S., Rugaber, S., & Goel, A (2008). Focusing on Function: Thinking Below the Surface of Complex Natural Systems. *Science Scope,* Summer 2008.

[4] Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[5] Wilensky, U., & Resnick, M. (1999). Thinking in levels: A dynamic systems approach to making sense of the world. *Journal of Science Education and Technology*, 8, 3-19.