AN ONTOLOGY FOR AGENT-BASED MODELING AND SIMULATION

Scott Christley, Xiaorong Xiang, Greg Madey
Computer Science and Engineering, University of Notre Dame

ABSTRACT

Ontologies are a formal methodology for establishing a common vocabulary, for defining the concepts and relationships between those concepts of a particular domain, and for reasoning about the objects, behaviors, and knowledge that comprises the domain. In this paper, we present an ontology for agent-based modeling and simulation. Agent-based modeling and simulation has become an important and popular paradigm for the computational social and natural sciences; however, the application of this paradigm tends to be performed in an ad-hoc fashion leading to questions about underlying assumptions in an agent-based model, verification of the software implementation as a representation of that model, and validation of hypothesized conclusions inferred from data produced by computer simulation experiments. An ontology provides a formal, logical knowledge representation that supports automated reasoning. Such reasoning capability provides for consistency checking of the concepts and relationships in an agent-based model, can infer the assumptions inherent in a model, can infer the assumptions and the parameters inherent in a simulation or software representation of a model, and can enforce adherence to formal methods or best practices for verification and validation testing. These reasoning tasks direct, or at least inform, the modeler about relevant techniques and methods in the agent-based paradigm. The reasoning capability also provides a framework for automated generation of software code, automated design and execution of simulation experiments as well as automated generation and execution of validation tests for those experiments. Using the standard Ontology Web Language (OWL), we provide a complete, detailed ontology of agent-based modeling and simulation, and we show how the ontology is used as part of the modeling and simulation process.

Keywords: Agent-based Modeling, Agent-based Simulation, Automated Reasoning, Ontology, Artificial Intelligence, Discrete-Event Simulation

This work is supported in part by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829.

INTRODUCTION

Agent-based modeling and simulation has become an important and popular paradigm for the computational social and natural sciences; however, the application of this paradigm tends to be performed in an ad-hoc fashion based upon a subjective understanding of the agent-based concept. Different techniques for model construction and implementation of computer simulations are often accompanied by underlying assumptions that are unknown to the researcher or cannot be explicitly characterized for the particular model. In addition, artifacts in the computer simulation may manifest themselves which lead to legitimate questions about the verification of the implementation and validation of hypothesized conclusions. Model-to-model comparison, or docking, can expose these issues because it forces the researcher to confront some of these underlying assumptions while analyzing differences between the two models, yet this is no guarantee against situations where the two models may inadvertantly hide possibly relevant assumptions.

Ontologies are a formal methodology for establishing a common vocabulary, for defining the concepts and relationships between those concepts of a particular domain, and for reasoning about the objects, behaviors, and knowledge that comprises the domain. In this paper, we present an ontology for agent-based modeling and simulation. The ontology is general in that we define the terms, concepts, and relationships for the process and objects of agent-based modeling and simulation without reference to a specific application domain. Such an ontology provides a solution to the issues of ad-hoc construction and subjective interpretation in three ways. The establishment of a common vocabulary provides unambiquous interpretation of terms; while, the definition of concepts and their relationships make explicit the assumptions that accompany those concepts. Lastly, the reasoning capability of this ontology provides a framework for automatic generation of software programs, automatic composition of agent-based models to form new simulations, automatic design and execution of simulation experiments as well as the automatic generation of validation tests for those experiments. Using the standard Ontology Web Language (OWL), we provide a complete, detailed ontology of agent-based modeling and simulation. Our ontology is too large to include within this article, but the files can be obtained from our website (Christley 2004). However, to better associate the discussion with the ontology, we will use the convention of *Italics* when referring to specific classes or properties in the ontology.

AGENT-BASED MODELING AND SIMULATION

There are three basic reasons for using simulation; to design something that does not yet exist, to train people when the real task is costly or dangerous, and to understand some real world phenomena as part of scientific study. All three uses have different processes and techniques; though, the design task and scientific inquiry can be considered similar to each other, but we are going to concentrate exclusively on using simulation to understand real phenomena. Simulation is used because it provides a finer control over the complete system that is usually not possible with the real system, and it provides the ability for extensive "what-if" analysis through tweaking of parameter and altering assumptions in the underlying theory. However, simulation injects a new problem into the scientific method in that a model of the theory for the phenomena must be implemented in a concrete representation so that it can be manipulated and simulated, so the question becomes not just is the theory consistent with the real phenomena but is the concrete model representation an accurate description of the theory and is the execution of that model an accurate representation of the processes in the theory. It needs to be taken one step further and asked are the implications of the simulation consistent with the implications of the theory; because if not, the simulation does not provide the logical step required to say whether the theory correctly captures the real phenomena. Most of the work with modeling and simulation involves doing the proper checks to provide a high degree of confidence for taking that logical step.

Our focus is on agent-based simulation, yet much is shared with general discrete-event simulation, so we will highlight the differences where appropriate while relying upon core fundamentals that have made discrete-event simulation a successful field. Despite being called agent-based simulation, the methodological differences lie more in the constructed models versus the implementation of those models in a computer simulation. When viewing a phenomena through the agent-based paradigm, one sees agents interacting with other agents within an environment and within a spatial structure. An agent is the conceptual unit of interest; there may be multiple agents, and the concept serves to define a boundary between what is internal to the agent versus what is external. By agent, we are referring to a prototypical concept and not an individual. The environment and space can also be considered agents as they may have interaction mechanisms, but they are generally differentiated as their boundary is not well-defined. The environment, or possibly multiple environments, represents state information that is external to the agents. The environment could be global state for all agents, or it may be local state in conjunction with the spatial structure with its defined notions of locality. Space can be two or three dimensional physical space, or it may be a virtual construct like a network. Space is different from the environment in that it provides measures, like distance or connectivity, and typically only holds state specific to those measures. There can be multiple spaces each with its own set of measures. A cognitive agents maintains, among other things, internal state about what it perceives about the environment and space; so in a simulation, the environment and space represent actual truth versus what a cognitive agent might perceive as truth.

By modeling, we refer to the process of representing something with something else; it can be an abstract model whereby the representation simplifies or removes extraneous detail to capture the conceptual properties, or it can be a concrete model which, oppositely, specifies a more detailed representation. By simulation, we refer to the process of enacting the model to learn consequences and to compare against the real phenomena of interest. There are four key modeling concepts that represent different model types: *ConceptualModel*, *CommunicativeModel*, *ProgrammedModel*, and *ExperimentalModel*. The *ConceptualModel* is a verbal, abstract model that states the theory or hypotheses for the proposed agent-based representation and the goal and objectives of the corresponding agent-based simulation. A *ConceptualModel* also provides descriptive specification for the agent, the environment, the space, and the actions and properties for those constructs. A *ConceptualModel* is made more concrete by constructing a *CommunicativeModel*. In our process, the *CommunicativeModel* is a domain-specific ontology that fits within the general agent-based ontology. Objects in the model like agents, environment, and space are represented through subclasses of those concepts in the general ontology. Subclasses are also created for the properties of those objects as well as their actions. Through *SoftwareProgramming*, a *ProgrammedModel* is constructed from the *CommunicativeModel* by representing the ontological concepts with concrete implementation in software code. A *ProgrammedMode* is one that can be executed as a *ComputerSimulation*, and in a later section we will discuss how the *ProgrammedModel* can be automatically generated from the *CommunicativeModel* using a reasoner. Lastly, *DesignExperiment* involves taking a *ProgrammedModel* to produce an *ExperimentalModel* and *PerformExperiment* will cause the *ExperimentalModel* to produce *SimulationData*; *Validation* can use a *StatisticalTest* to compare the *SimulationData* against *EmpiricalData*. This is a simplified example of the modeling and simulation process as there are many more actions and concepts involved; Figure 1 shows a portion of the semantic network representing our formalized knowledge about agent-based modeling and simulation.

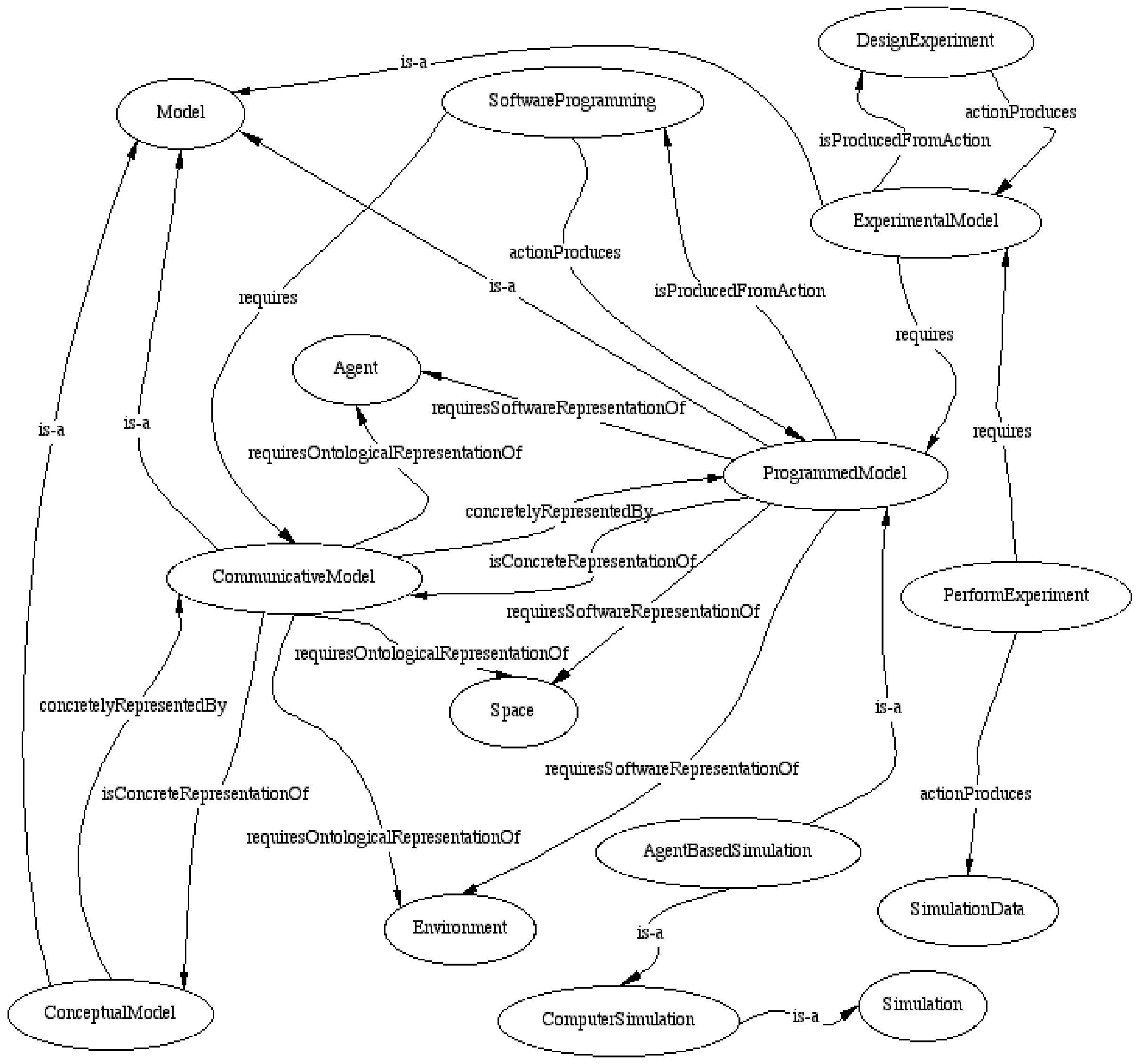

FIGURE 1 Relationships between Models and Agent, Space, and Environment

ONTOLOGY FOR AGENT-BASED MODELING AND SIMULATION

When developing an ontology for agent-based modeling and simulation, we have to be careful to clearly distinguish between the concepts and relationships that comprise the process of modeling and simulation versus the agents and behaviors in the domain of interest; yet these two are intimately related. The latter is called the domain-specific ontology, and the former is called the general ontology. The relationship between the two is simply that the domain-specific ontology provides more detail concepts and properties; for example, the general ontology has the concept of an agent which has some undefined properties and behaviors, but the domain-specific ontology will have the concept of a SoftwareProgrammingAgent which has defined properties like skill and resources and defined behaviors like writing code and fixing bugs.

Our ontology is implemented in the Ontology Web Language (OWL 2004) using Protege (Protege 2004). OWL represents knowledge as a semantic network with nodes as classes and directed edges as properties. We have 100+ classes in our ontology with a similar number of properties. The root classes include *Agent*, *Environment*, *Space*, *Action*, and *Property* for concepts in the agent-based paradigm. *Action* and *Property* along with *Model*, *Simulation*, *Representation*, *DataSource*, *Test*, and *Assumption* are root concepts for the process of modeling and simulation. Lastly, *Measure*, *Time*, and *Event* are concepts that appear in any general ontology. Notice that *Action* and *Property* are both concepts about the modeling process as well as in the model, so we have *AgentAction*, *EnvironmentAction*, *AgentProperty*, and *EnvironmentProperty* subclasses for those concepts in the agent-based model; while, *ModelerAction* and *ModelProperty* subclasses are concepts about the modeling process. The classes that refer to concepts in the agent-based model stop at a general description, so more specialized subclasses would be provided by the domain-specific ontology. Our ontology is focused on the process of modeling and simulation, so *ModelerAction* includes subclasses like *InputModeling*, *ParameterEstimation*, *DesignExperiment*, *Verification*, *Validation*, *ModelToModelComparison*, and others. The *Model* class encapsulates all types of models though we concentrate on *ConceptualModel*, *CommunicativeModel*, *ProgrammedModel*, and *ExperimentalModel* as described in the previous section. *Simulation* can be split between *ComputerSimulation* and *PhysicalSimulation* with *AgentBasedSimulation* as a subclass of the former. *Representation* deals with representational forms like *OntologyRepresentation* as given by a *CommunicativeModel* or *SoftwareRepresentation* as embodied in a *ProgrammedModel*. The *DataSource* class conceptualizes all sources of data like *EmpiricalData*, *RandomNumberGenerator*, and *SimulationData*. *Test* refers to all forms of testing especially specialized classes of *StatisticalTest* used in *InputModeling* and *Validation* actions. Lastly, we have the concept of *Assumption* which our reasoner will use to categorize the assumptions within the agent-based model. All of the concepts in the general ontology establish a common vocabulary that can be shared across domain-specific ontologies and provide unambiguous interpretation of conceptual terms.

Besides classes, OWL has properties that define the relationship between concepts; the properties themselves are concepts that can form an inheritance hierarchy. Many properties are found in most ontologies that represent general relationships such as composition with *isPartOf* and *isBunchOf*, dependencies like *requires*, ordering of events with *isBefore*, *isAfter*, *overlapsWith* among others, or actions like *has* and *produces*. We specialize many of these relationships for agent-based modeling so that we can perform more accurate reasoning tasks. For example, a *NormalDistribution hasParameter Mean* and *hasParameter Variance*; thus, we will be able to reason that a simulation using a *NormalRNG* to produce normally distributed random numbers will require two parameters to define the distribution. Likewise, *PerformExperiment requires* an *ExperimentalModel* that *isProducedFromAction DesignExperiment* and that *ExperimentalModel requires* a *ProgrammedModel* that *requiresSoftwareRepresentationOf Space*, *Environment*, and *Action*. As for classes, the properties establish a common vocabulary for relationships, and the properties and classes together form our complete knowledge base for agent-based modeling and simulation.

ONTOLOGICAL REASONING

An ontology formalizes our knowledge base, so it is possible to perform automated reasoning on the process of modeling and simulation as well as on the models and simulations themselves. Reasoning on a model and its corresponding simulations provide us with a set of inferred assumptions for the model, a set of inferred assumptions for the representation of the model as a simulation, and a set of inferred parameters for the simulation. Reasoning on the process of modeling and simulation gives the potential of automating many of the primary tasks in the process including software programming of the simulation, design and execution of computer simulation experiments, and validation of experimental results. In the following sections, we will describe each of these capabilities in more detail.

Inferred Assumptions

An *Assumption* can be further categorized into a *DataAssumption* or a *StructuralAssumption*. A *DataAssumption* refers to questions about how data is collected and how data is analyzed, so *InputModeling* of *EmpiricalData* to come up with an appropriate probability distribution introduces a *DataAssumption* that the distribution is an appropriate representation of the *EmpiricalData*. A *GoodnessOfFitTest* can be used to validate that assumption. A *StructuralAssumption* refers to questions about the composition of the model and the conceptual representations in the model; concepts in the model and the relationships between those concepts, as abstract constructions of reality, imply assumptions about how those constructs are represented and whether the relations are correct. Viewing a *CommunicativeModel* as a semantic network, a *StructuralAssumption* asks whether the nodes are appropriate concepts, whether the edges are appropriate properties, and whether concepts linked by an edge is an appropriate relationship. Assumptions can either be falsified or failed to be falsified (validated) much like a null hypothesis if an appropriate test can be performed. For the *InputModeling* example above, the *GoodnessOfFitTest* performs this function; while, experiments and tests would need to be performed to provide *Validation* of a *CommunicativeModel*. Not all assumptions can be tested like whether a *CommunicativeModel* accurately represents the concepts in a *ConceptualModel* because the *ConceptualModel* is a verbal model lacking a formal description; the best that can be performed is a *SubjectiveTest* like *FaceValidity*. The reasoner is able to infer all of the assumptions in an agent-based model from the *CommunicativeModel* through to the *ExperimentalModel*, and our goal is for the reasoner to determine whether these assumptions can be validated and what appropriate test should be used. The assumptions can be inferred by looking at the properties on the classes and questioning whether the relationship between the classes implied by the property is correct. With all of the assumptions clearly laid out, the modeler obtains a broader view of how the agent-based model can be validated and may offer insights into model changes to strengthen the overall theory.

Inferred Parameters

A *Parameter* is a *ModelProperty* which is considered as an input to the model. A *Parameter* may be given a value through the *ParameterEstimation* action, or the modeler may *AssignParameterValue* as part of *DesignExperiment*. The *Parameter* may have a constant value throughout the simulation, or it may be attached to a *DataSource* like *EmpiricalData* or sampled from a *Distribution* created by a *RandomNumberGenerator*. An *InitialCondition* is a *ModelProperty* similar to a *Parameter*, but an *InitialCondition* assigns values to state variables for just the start of the simulation; while, a *Parameter* is persistent through the whole simulation run. Like a *Parameter*, an *InitialCondition* may be assigned a specific value or sample values from a *DataSource*. The reasoner has the capability to determine all of the *Parameters* and

InitialConditions in an agent-based model. It can do this because the ontology encodes knowledge of the properties of agents, environment, and space, so a logical query on the properties provides the list. The results of such a logical query becomes part of the automatic design and execution of experiments whereby the query results are presented to the modeler for specification of input values.

Automated Software Programming

Complete automated software programming of the simulation requires more than a *CommunicativeModel* embedded within the agent-based ontology because it does not provide enough detail to generate source code for all agent and environment actions. Attempts to provide high-level specification of software can fail because too many assumptions must be made about the functionality and purpose of the software, or the specification process may be more cumbersome than directly writing the code (Rich et al. 1988, Flener et al. 1994). However, we believe an intermediate approach is both feasible and useful. The *CommunicativeModel* can be translated into the high-level structure of the *ProgrammedModel*. This includes generation of the object-oriented classes for the agent, environment, and spatial constructs in the model with instance variables for the properties of those constructs and accessor, constructor, and stub methods for the constructs' actions. Such an intermediate approach means the modeler can focus upon the software implementation for the fundamental behaviors in the model while much of the "glue code" required to make the simulation work is handled automatically.

Model Composition

Another fruitful area of automation is the composition of multiple, separate *CommunicativeModels* into a single *ProgrammedModel*. These *CommunicativeModels* can be created by the same modeling group or different groups. Composition of *CommunicativeModels* requires semantics of the interactions between the models. We separate the composition process into two situations according to whether these *CommunicationModels* consist of the same or different collection of entities.

1) Two *CommunicativeModels* represent the same collection of entities that interact together over time. We consider these two *CommunicativeModels* as representations of the same world phenomena. One of the research groups in the University of Notre Dame models the evolution of NOM (Natural Organic Matter, a complex mixture of molecules that is heterogeneous in structure and composition) using the agent-based modeling approach (Xiang et al. 2005). As NOM passes through an ecosystem, it is acted upon by variety of reactions. In order to satisfy different research interests, two communicative models are developed. One models the physical reaction behaviors of NOM, and the other models the chemical reaction between NOM and its environment. A new, third model can be generated that includes both of these two behaviors by composing the two *CommunicativeModels*.

2) Two *CommunicativeModels* represent a different collection of entities that interact together over time. They are considered as representations of different world phenomena. In the example we describe above, the microbes, fungi, and bacteria exist in the natural environment and interact with NOM; in the current *CommunicativeModels*, they are represented as a set of environment state variables. It is more realistic that these micro-organisms be represented as agents

in the NOM world and their interaction with molecules are explicitly modeled. When there is an existing model that models the micro-organisms life cycle (micro-organisms can reproduce themselves and die) with agent-based modeling approach, creation of the new model can be benefited from the composition of these two existing models.

The composition of *CommunicativeModels* requires merging different domain-specific *CommunicativeModels* together. This merging process can be automated with ontological reasoning. Three possibilities for semantics arise for these two situations: the semantics may either be already described in both *CommunicativeModels*, in only one model, or in neither of the models. In the first situation, most likely, both *CommunicativeModels* have the same semantics. In the second situation, the semantic are most likely existing in one *CommunicativeModel* but not in another (partially overlapped).

One important task residing in the merging process is determining whether two domain specific concepts are the same in two *CommunicativeModels*. Determining the "structural equivalence" of two concepts by comparing the incoming edges and outgoing edges of these two concepts is one way to complete the task. Much research has addressed on matching the concepts using sophisticated algorithm and AI techniques, such as machine learning (Doan et al. 2003, Noy et al. 2000). The merging process may involve integration of new knowledge, such as specifying the new interaction among agents, which requires the input from model developers. With complete knowledge representation, the composition process can be done automatically.

Automated Design and Execution of Experiments

Automated design of simulation experiments can be implemented through manipulations of the *ProgrammedModel*. Such manipulations include basic assignments of values to *Parameters* and *InitialConditions*, enabling or disabling of *Actions* for the *Agents*, *Environment*, and *Space*, or even completely different implementations for those constructs. Here we take the viewpoint that an experiment works within the framework of a *CommunicativeModel* and that manipulations to that model fall outside the domain of the *ExperimentalModel*. However, most model manipulations can be supported so long as the possible changes are encapsulated through ontological concepts in the *CommunicativeModel*. An example can illustrate; suppose you have designed a model and corresponding simulation whereby the agents interact in a two-dimensional continuous space using a Euclidean distance neighborhood measure, and you decide you want to replace the space with a random network structure connecting the agents. Changing the spatial structure will create a logical inconsistency because a network does not have a Euclidean-distance neighborhood measure; the inconsistency is resolved by manual alteration of the *ProgrammedModel* to utilize a different neighborhood measure. In contrast, if the original *CommunicativeModel* had both spaces, then a general concept would have been required to encapsulate the neighborhood measure; the result being the *ProgrammedModel* allows for automatic manipulation, via a *Parameter*, of the spatial structure through use of a generalized neighborhood measure. This is not to say that one is more capable than the other, but because we have taken an intermediate approach to software code generation, inconsistencies due to model changes outside of the *ExperimentalModel* may not be automatically resolved within the *ProgrammedModel*.

Once an *ExperimentalModel* has been designed, it can be executed and *SimulationData* is produced which can then be validated. One execution of a simulation is not sufficient; numerous executions, or replications, of the *ExperimentalModel* must be performed with different seed values for any *RandomNumberGenerator* in the simulation, and the reasoner can automate these replications because knowledge of the seed values is part of the ontology. Likewise, a modeler does not generally design a single experiment; experimentation is often an iterative process whereby experimental results are analyzed, changes are made to the *CommunicativeModel*, those changes flow through to the *ProgrammedModel*, and a new *ExperimentalModel* is designed. This iterative process continues until the modeler feels the *CommunicativeModel* has been sufficiently validated. At this point, the next step depends upon the purpose of the simulation. Presuming the simulation is for scientific discovery, *SensitivityAnalysis* is an example action that can be performed to better understand the role of the model parameters, or experiments with different model parameters or design may be performed to generate hypotheses that can be tested against the real world phenomena.

Validation of Simulation Experimental Results

Validation is the process of comparing a model against the real world phenoma it represents. All *Validation* is based upon a *Test* that decides whether two things are same or not. There are weak tests and strong tests; a weak test is a *SubjectiveTest* that does not have a well-defined decision procedure. A *SubjectiveTest* includes things like a *VisualTest* where you make a visual comparison of two graphs or *FaceValidity* where a knowledgeable user makes a determination if the model appears reasonable. A strong test is generally associated with a *StatisticalTest* where a formal mathematical decision procedure exists to objectively make a determination. Computers have difficulty performing *SubjectiveTests* bu they excel at *StatisticalTests*, so the reasoner can perform automatic validation provided it has sufficient knowledge about what type of *StatisticalTest* is appropriate for the *SimulationData* provided by an experiment. Many statistical tests exists and formalizing all of them in our ontology is a large task; however, we incorporated many of the standard techniques like *GoodnessOfFitTest*, *ConfidenceInterval*, *AnalysisOfVariance*, *TestOfMeans*, and *TimeSeriesAnalysis*.

One particular form of *Validation* introduced by Axtell et al. (1996) is *ModelToModelComparison* where two simulations are compared. The original definition has the same *CommunicativeModel* but different *ProgrammedModel*, possibly written in different programming languages or using different simulation toolkits, and correlated *ExperimentalModels* are designed and their *SimulationData* is compared. *ModelToModelComparison* provides a good test to validate that the *ProgrammedModel* is an accurate representation of the *CommunicativeModel*, so differences indicate that artifacts exist in the *ProgrammedModel*. Takadama et al. (2004) proposes the notion of cross-element validation that makes small changes, one element at a time, in the *CommunicativeModel* and compare the experimental results. For such an experiment, the Bonferroni approach can be used, if the *SimulationData* is a fixed sample size, to produce a confidence level of whether the two models are statistically similar or different. We consider both the original definition and cross-element validation to be forms of *ModelToModelComparison*; likewise, our iterative description of the experimental process allows for the possibility of *ModelToModelComparison* between *CommunicativeModels* as they evolve from one iteration to the next. With knowledge of multiple programming languages and multiple simulations toolkits, the reasoner can automatically generate multiple *ProgrammedModels* from a single *CommunicativeModel* allowing for greater experimentation.

FUTURE WORK

Our coverage of simulation in general has been brief for this short article; we have described key areas that we consider relevant to agent-based modeling and simulation while leaving out some areas completely, and in the areas we do cover, our discussion is not as encompassing as we would like. However, we have presented a high standard for automation of many simulation tasks. Going forward we intend to implement tools specific to agent-based modeling that can perform these tasks and put them in practice on a couple of actual agent-based simulations; this should help elicit more issues that are not apparent just from the theory. One of our key assumptions is the completeness of our ontology which makes many of the automated tasks possible. A more realistic scenario is to assume incomplete knowledge as well as uncertainty, then we use a probabilistic reasoner for making decisions and a learning algorithm to accumulate additional knowledge. This is very much what a modeler does as part of the scientific inquiry into a phenomenon; a useful tool will work alongside the modeler helping to increase the knowledge base while automating many of the mundane tasks.

REFERENCES

Axtell, R., R. Axelrod, J. Epstein, and M. Cohen, 1996, "Aligning Simulation Models: A Case Study and Results," *Computational and Mathematical Organization Theory*, 1:123-141.

Balci, O., 1998, "Verification, Validation, and Accreditation", in *Proceedings of the 1998 Winter Simulation Conference*, pp. 41-48.

Banks, J., 1998, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, Wiley co-published by Engineering & Management Press, New York, NY.

Banks, J., J.S. Carson II, B.L. Nelson, and D.M. Nicol, 2001, *Discrete-Event System Simulation*, Prentice Hall, New Jersey.

Christley, S., 2004, OWL for Agent-based Modeling and Simulation, available at http://www.nd.edu/ ~schristl/research/ontology.

Doan, A., J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy, 2003, "Learning to Match Ontologies on the Semantic Web," in *The International Journal on Very Large Databases*, 12.

Flener, P., and L. Popelmnsky, 1994, "On the Use of Inductive Reasoning in Program Synthesis: Prejudice and Prospects," in *Logic Programming Synthesis and Transformation, Meta-Programming in Logic: Fourth International Workshops, LOBSTR'94 and META'94, Pisa, Italy*, pp. 69-87, Springer-Verlag, Berlin.

Grimm, V., 1999, "Ten Years of Individual-based Modeling in Ecology: What have we learned and what could we learn in the future?", *Ecological Modeling,* 115:129-148.

Miller, J.A., G.T. Baramidze, A.P. Sheth, and P.A. Fishwick, 2004, "Investigating Ontologies for Simulation Modeling," in Proceedings of the 37th Annual Simulation Symposium, pp. 54-63, Arlington, Virgina.

Noy, N.F., and M.A. Musen, 2000, "Prompt: Algorithm and Tool for Automated Ontology Merging and Alignment," in *The Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

O'Sullivan, D. and M. Haklay, 2000, "Agent-based models and individualism: is the world agent-based?", *Environment and Planning A* **32**(8):1409-1425.

OWL, 2004, Web Ontology Language, available at http://www.w3.org/tr/owl-features.

Protege, 2004, Protege Ontology Editor, available at http://protege.stanford.edu.

Rich, C., and R.C. Waters, 1988, "Automatic Programming: Myths and Prospects," *IEEE Computer* **21**(8):40-51.

Ropella, G.E., S.F. Railsback, and S.K. Jackson, 2002, "Software Engineering Considerations for Individual-based Models", *Natural Resource Modeling* **15**(1):5-22.

Russell, S., and P. Norvig, 1995, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey.

Sarjoughian, H.S. And F.E. Cellier (editors), 2001, *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler*, Springer-Verlag, New York, NY.

Takadama, K., and H. Fujita, 2004, "Lessons Learned from Comparison between Q-Learning and Sarsa Agents in Bargaining Game," in *North American Association for Computational Social and Organizational Science (NAACSOS 2004)*, Pittsburgh, PA.

Xiang, X., Y. Huang, G. Madey, and S. Cabaniss, 2005, "Modeling the Evolution of Natural Organic Matter in the Environment with an Agent-based Stochastic Approach", *Journal of Natural Resource Modeling* (to appear).