Impact of the Semantic Web on Modeling and Simulation

John A. Miller
Congzhou He
Julia I. Couto

Department of Computer Science
University of Georgia
Athens, GA 30602

1 Introduction

During the middle 1990s, the World-Wide Web began to substantially impact the use of computer technology. This sparked the development of the field of Web-based Simulation, which is still advancing today. This chapter will examine how an ongoing major initiative involving the Web, the Semantic Web, may further impact Modeling and Simulation.

More specifically, this chapter considers the issue of using semantics in Modeling and Simulation (M&S). The impetus for this is the large initiative to develop the next generation Web, the Semantic Web being developed by the Artificial Intelligence, Database and Information Retrieval communities. A complimentary parallel track is represented by the Model Driven Architecture (MDA) approach being developed by OMG and the Software Engineering Community. The goal of this initiate is for all software development to be model driven.

Semantics (and the Semantic Web) will likely impact the M&S Community in two ways. First, the community should develop ontology to delineate, define and relate the concepts in the field. Ontology for modeling and simulation should be logically connected to more general (or higher-level) ontology e.g., one for mathematics such as Monet (Caprotti, Dewar, and Turi 2004) or one for general knowledge upper ontology such as the Suggested Upper Merged Ontology (SUMO) (Niles and Pease 2001). Second, simulation models, model components and other artifacts should be provided with richer semantic descriptions. The least disruptive way to do this is through annotation in which the artifacts refer to semantic models (e.g., a concept in an ontology).

The fact that the Semantic Web is being developed and simulation artifacts can be semantically annotated, begs the question of why do it. This question relates to the basic motivation for having the Semantic Web and Semantic Web Services. For the M&S community, semantics represented in ontology provides standard terminology to the community and beyond, so that common understanding of concepts and relationships can be achieved, which, in turn, increases the potential for application interoperability and reuse of simulation artifacts. Semantic Web technology can also be used for discovery of simulation components, composition of simulation components, implementation assistance, verification and automated testing.

In order to make the discussions in this chapter more directed, we will develop as we go a small ontology for Discrete-Event Simulation (DeSO). The purpose of this ontology is to provide a general conceptual foundation for modeling and simulation. Every effort was made to keep the ontology from becoming convoluted. If concepts were too complex to be defined in a straightforward way, they were left out. At this point, the DeSO ontology is a toy example. Later, we plan to expand and merge it with the more developed Discrete-event Modeling Ontology (DeMO). DeMO is oriented toward discrete-event modeling techniques such as Markov Chains, Finite State Machines, Petri Nets and Event Graphs. We are making DeSO more general in the following ways:

1. Include concepts related to common methodologies for creating simulation models, e.g., those built using simulation languages (or programming languages augmented with simulation libraries). DeMO is more oriented toward formal modeling techniques (or course that was a sensible place to start since these are well defined, at least mathematically).

2. Take a first step to extend DeMO with concepts from combined continuous and discrete simulation, without obscuring the discrete event concepts.

3. Include enough concepts to allow, say, a simulation engine to interact with an animation engine. The animation engine would permit realistic (or at least interesting) rendering using 2D or 3D graphics. Either engine (simulation or animation) could include software such as a physics engine to enhance the realism of animation. This is part of the motivation for item 2, allowing, for example, smooth continuous motion governed by Newton's Laws of Motion.

Building such a large ontology is a daunting task which needs guidance from well-established foundational knowledge. In this work, we use the following foundational sources: Modeling, Simulation, Systems Theory, Physics, Mathematics and Philosophy.

We endeavored to make our definitions as compatible as we could with existing definitions within these fields. Many sources were used for this including Wikipedia (Wikipedia 006c), WordNet (Miller, Beckwith, Fellbaum, Gross, and K. 1990), OpenMath (OpenMath 2006), SUMO (Niles and Pease 2001), Stanford Encyclopedia of Philosophy (of Philosophy 2006), astroonto (Shaya, Thomas, Huang, and Teuben 2006), Simulation Reference Markup Language (SRML) (Reichenthal 2002), eXtensible Modeling and Simulation Framework (XMSF) (Brutzman 2004) and DEVS (DEVS 2005) as well as textbooks and papers in a variety of fields (see the bibliography).

Finding and defining the concepts is hard enough, but the subsequent step of determining a minimal set of useful properties is even more difficult. More important to get right are the relationships between the concepts. This is where much of the formal semantics comes in, since after all many of the concepts are defined in natural (not formal) languages. Indeed, certain semantically primitive concepts are not formally definable.

The rest of this chapter is organized as follows. In section 2, we overview developments in the Semantic Web relevant to creating and using ontology for modeling and simulation. Section 3 provides a conceptual framework suitable for defining the top concepts for such ontology. This is followed, in section 4, by high-level classifications based on these main concepts. Techniques for adding semantics to simulation models is given in section 5. A summary of the DeSO ontology is presented in section 6. An overview of the DeMO Ontology is given section 7. Lastly, section 8 summarizes the chapter.

2 Semantic Web: Relevant Issues

Ever since the Scientific American article by Berners-Lee, Hendler and Lassila (Berners-Lee, Hendler, and Lassila 2001), there has been a great deal of research and development on the Semantic Web. Indeed, much of it is rooted in prior research in knowledge representation, distributed artificial intelligence, database systems and information retrieval. A large portion of the current Web consists of HTML pages (either static or dynamic) aimed for humans to read. In order to make the Web more accessible by programs (or agents), the Web content needs to be organized better, linking meaning with content. An obvious first step is to replace the formatting tags of the HyperText Markup Language (HTML), with ones that are related to content. This is the purpose of the eXtensible Markup Language (XML) and its schema languages: Data Type Definition (DTD) and XML Schema Definition (XSD). XML is good for representing nested structures in documents, but is weak regarding named relationships.

The Resource Description Framework (RDF) is useful for indicating that certain entities of interest are discussed in a document and that these entities are related to other entities in this and other documents. In this way, it permits logical connections within and between documents. Although, one might think that hyperlinks in HTML or XLinks in XML documents play a similar role, from a program's perspective these are akin to untyped pointers. RDF provides a richer modelling language, and although RDF syntax can be represented using XML, the underlying abstract models for the two languages are fundamentally different. The abstract model for XML is tree based, while the model for RDF is graph based (Berners-Lee 1998; Johnston 2005).

So far, the above additions to the Web mainly provide it with better organization, which is key to making the Web more useful to programs. The real goal of the Semantic Web is to make the Web content more understandable to programs. One approach is to use Natural Language Processing and Text Understanding. These long term research efforts are beginning to bear fruit, and various algorithms have been designed to process text at morphological, syntactic, semantic and discoursal levels with reasonable accuracy (Mitkov 2003). However, they are not the principal focus of current Semantic Web research. As already mentioned, the tags used by XML are more meaningful than the tags used by HTML (e.g., <h3> ... </h3> versus <address> ... </address>. While certainly true, this meaningfulness is mainly attributed to human understanding. What does it mean to a program? A first step is to reduce the program's requirements for understanding the document to a schema which applies to several documents of the same kind. If the program knows the XSD for a group of documents, then it can more readily process the document. Furthermore, if the program knows the RDF Schema (RDFS) for this group, it can process relationships between entities in this group of documents. This capability is particularly useful for semantic search (Sheth, Ramakrishnan, and Thomas 2005). Whereas, Web search engines such as Yahoo and Google use keyword search and page ranking schemes, semantic search follows meaningful links, and has the potential, in specific domains, to enhance precision and recall [1] of documents as well as direct one to relevant portions of documents (Noronha and Silva 2004). Still, the depth of program understanding is rather shallow (useful, but shallow).

Deep understanding approaching human levels is such a long term goal that something more intermediate is needed. For one thing, it would be nice to give the tags used in XML documents more precise definitions. A key aspect of the Semantic Web is to provide standard (i.e., agreed upon) definitions of terms or concepts in a variety of domains. A terminology defines a set of related terms, which may be classified to form a taxonomy. When named relationships are added, it may be referred to as ontology. Specifically, ontology concerns the classification of concepts (or classes) as well as their subclasses, properties and relationships to other concepts. These defined concepts can also be used to annotate the content of documents. Finally, instances of these concepts can be created by extracting content from Web pages. Together the classes, properties and instances form a knowledge base. The Web Ontology Language (OWL) provide this capability for the Semantic Web (OWL comes in three flavors: OWL-Lite, OWL-DL where DL stands for Description Logic, and OWL-Full). Other possible languages for modeling ontology include the Entity-Relationship model (Chen 1976), Unified Modeling Language (Rumbaugh, Jacobson, and Booch 1998), Knowledge Interchange Format (Genesereth and Fikes 1992), and Resource Description Framework (Klyne and Carroll 2004).

Having introduced the term knowledge base, we should mention that typically they may also include rules (or something equivalent). Indeed, the latest part of the Semantic Web undergoing standardization is the Semantic Web Rule Language (SWRL). Rules allow new facts to be generated from existing facts and relevant rules, thus greatly increasing the expressivity of the knowledge base. Unfortunately, as the expressivity goes up, so does its complexity. Table 1 shows the current set of languages used in the Semantic Web, and includes the complexity class of basic inferencing

[1]Precision means the fraction of retrieved documents that are relevant; recall means the fraction of relevant documents that are retrieved.

operations such as subsumption [2] (Tractable Fragments of the OWL 1.1 Web Ontology Language).

[Table 1 about here.]

A more general and deeper discussion of semantics and ontology as well as their relationship to the Semantic Web is given in the Appendix. Although all the Semantic Web languages are important, the Web Ontology Language is the most relevant to this chapter. We may use it to define and relate terms or concepts in the fields of modeling and simulation. In the next section, we develop a conceptual foundation for modeling and simulation. From this conceptualization, we create an OWL ontology. This ontology is broad, but currently shallow. This ontology also includes a few SWRL rules. Later in this chapter, we overview the DeMO ontology which is narrower and deeper.

3 Conceptual Foundation for Discrete-event Simulation

In this section, we develop a conceptual framework that is needed to clearly capture the foundational concepts of discrete event modeling and simulation. A secondary goal is to provide a very general framework for discrete event modeling including combined discrete-continuous modeling. A tertiary goal is to keep this framework as simple as possible. This last goal, may allow naivety to creep in, especially in regards to continuous modeling. For example, with continuous modeling, energy-based modeling may work better than state-based modeling (Cellier 1991). Keep in mind that the main purpose of this framework is to create basic ontological concepts for understanding the field of discrete-event simulation and modeling.

The model world begins with an empty void with space and time coordinates (Wikipedia 006b).

- **Time**. Let $t \in T$ indicate a point in time. Typically, T would be a subset of the field of real numbers $\mathbb{R}$ or the ring of integers $\mathbb{Z}$.

- **Space**. Let $x \in X$ indicate a point or location in a vector space X. For example, X could be the three dimensional vectors space $\mathbb{R}^3$ or something more abstract. Together, space and time form a space-time continuum as in relativity theory. We, however, continue to treat time as a special dimension in space-time. The void is then filled with objects, which are principally entities. Entities are the things that exist in the model world. If entities do not interact, the entities that exist at the start of simulation would simply move at a constant velocity (or remain at rest) forever. In order to allow entities to interact, additional entities need to be introduced into the model world. These agents may cause changes to entities such as entity creation, destruction, property updates and acceleration. Events are used to model

[2]The Ontology Definition Metamodel ODM supporting UML which is under development by the Object Management Group (OMG) has a proposal to have a description logic core.

changes that occur instantaneously (or nearly so). Forces are used to model changes that occur smoothly over time.

- **Entity**. An entity k is an object that exists in space-time. It is also uniquely identifiable. Examples of entities include customers in banks, golf balls flying through the air and even molecules in boiling water. As the number of entities becomes very large, modeling techniques that deal with aggregations of entities offer advantages. In many cases, the models will deal with properties of aggregations such as pressure, temperature or weight rather than the entities (or aggregate entities) themselves.

- **Event**. An event e is an object that does not exist in space-time, rather it exists only in the time dimension. It has a creation time, but the important time is its occurrence time. When the event occurs, it may affect other entities, trigger other events or modify forces. For examples, it may create (or destroy/cancel) other events, increase (or decrease) forces, move entities or change entity properties. An event is considered to occur instantaneously and therefore can produce discontinuities in the trajectories of entities within space-time (see below and Figure 1). In order to relate the event to space-time, we assume that it is associated with a particular main entity or agent. Finally, the event must specify what action is to be performed. The action may be specified as algebraic equations, difference equations or in general using action logic (all of which may be implemented using a programming or simulation language). The type of action determines the type of the event (for example an arrival event or a departure event). The complete set of event types is denoted by the set E.

- **Force**. Complementary to events which have immediate effects, forces make changes over time. This corresponds to the world view provided by classical physics, e.g., as exemplified by Newton's Laws of Motion. Force laws are typically expressed as differential equations. A common force to use is gravity, which in simulations/animations makes the motion of entities look more realistic.

- **State**. Let $\{w(t) | t \geq 0\}$ be the process (e.g., a stochastic process) representing the evolution of the model world over time. We would like to be able to stop the process at some t_i, and save the minimal amount of information from the initial conditions $w(t_0)$ and the current world $w(t_i)$, so that the process can be resumed without affecting any future results. How this is done and how the dynamics are expressed in terms of this information largely defines the type of modeling technique that is applied. We assume that the following are defined at the beginning of the world $w(t_0)$: the range of time for the model world to exist, the space for entities in the model world to exist in, the event action logic (for discrete changes), and the forces (for continuous changes) and an initial event (or events) to initiate the simulation.

– **No Force Case**. At first, let us keep things simple and suppose there are no forces. The dynamic state of the model world $s(t)$ is simply the aggregation of the configuration (location and, if need be, property values) of all the currently existing entities.

$$s(t) = (x_1(t), ..., x_k(t), ...) \quad where \quad x_k(t) \in X \tag{1}$$

Besides the current state, we must know the future events that are ordered in the time dimension.

$$r(t) = ((e_1, t_1), ..., (e_j, t_j), ...) \quad where \quad e_j \in E \quad and \quad t_j \geq t \tag{2}$$

Let $w(t)$ be the model world at time t.

$$w(t) = f(w(t_0), s(t), r(t)) \tag{3}$$

That is, the model world can be reconstructed from the initial configuration of the world, the current state of the entities and the list of future events.

– **Force Case**. Adding forces to the simulation can be done in many ways. Pritsker (Pritsker 1986) talks about three types of interactions: type 1: a discrete event makes a discrete change to a continuous variable (a discontinuity); type 2: a discrete event affects the physical laws governing the behavior of entities (equations governing continuous variable); type 3: a "state" event triggers a "time" event. A state event is said to occur when an entity or variable reaches a certain threshold, say $x_k \geq c$.

4 Types of Mathematical Models

Now that we have defined the fundamental concepts of time, space, entity, state, event and force, we may classify mathematical models by differentiating them based on these fundamental concepts. Recall that the first two concepts, time T and space X, form the time-space continuum in which entities exist. Finally, the agents of change are events E and forces. We may consider changes in time t and state $s(t)$ as specified by a clock function c and a transition function f.

$$h = c(s(t), t) \tag{4}$$

$$s(t + h) - s(t) = f(s(t), t)g(h) \tag{5}$$

The clock function c determines the time increment h, while the transition function f determines the next state. For continuous-time models, we let $g(h) = h$; otherwise we let $g(h) = 1$.

4.1 Classification Based on State

.

The state of the model world, which is a snapshot at a particular time, is based on the more primitive notion of space as well as the notion of entities that populate the space. If there is only one entity and it has no varying properties, then the two concepts space and state may be unified (i.e., $s(t) = x(t)$). Since the concepts of space and state go together, we will classify them together, in regards to whether they are discrete or continuous. The distinction between discrete and continuous simply depends on the cardinality of the state space S. The state space is discrete if its cardinality is less than or equal to the cardinality of the Natural numbers, $\mathbb{N}$, denoted by $\aleph_0$; otherwise, we consider it to be continuous, i.e.,

$$|S| <= \aleph_0 \quad or \quad |S| > \aleph_0 \tag{6}$$

Similarly, one could say that discrete means that the state space is finite or countably infinite (like the integers, $\mathbb{Z}$), whereas, continuous means the state space is uncountable (like the reals, $\mathbb{R}$).

4.2 Time Based Classification

Our second classification is based upon time, particularly the clock function (i.e., how time is advanced: continuous-time, discrete-event, discrete-time or static).

4.2.1 Continuous-time Models

In continuous-time models, the clock moves smoothly and continuously. The next instant of time $t + h$ is infinitesimally beyond the current time t. If we let $g(h) = h$ and consider the limit as h tends to 0, we obtain the following:

$$\lim_{h \to 0} s(t+h) - s(t)/h = f(s(t), t) \tag{7}$$

$$\frac{d}{dt}\, s(t) = f(s(t), t) \tag{8}$$

This equation is a first-order Ordinary Differential Equation (ODE). Rather than having a function to describe the state trajectory (i.e., the values of $s(t)$ over time), the function in equation (8) describes the rate of state change. The trajectory can then be determined using some solution technique (e.g., integrating factors, Runga-Kutta, etc.). Writing the equation in terms of the derivative allows one to concisely express commonly occurring phenomena such as systems with constant growth rates and many laws of classical physics such as Newton's Second Law of Motion. For one entity (e.g., a ball) whose coordinates in space-time are given by ($x(t)$, t) and acted upon by a constant force $-g$, Newton's Second Law becomes the following:

$$-g = m \, \frac{d^2}{dt^2} \, x(t) \tag{9}$$

Note that this is a second-order ODE. However, by introducing another variable, velocity v, into the state, this second-order ODE can be converted into two coupled first-order ODE's.

$$-g = m \, \frac{d}{dt} \, v(t) \tag{10}$$

$$v(t) = \frac{d}{dt} \, x(t) \tag{11}$$

In this and many other cases, enlarging the state from $x(t)$ to $(x(t), v(t))$ can allow one to model phenomena using first-order ODE's. This is similar to the technique of enlarging the state space to make stochastic systems Markovian. Still, much of physics requires more than time derivates, e.g., partial derivates, leading to a Partial Differential Equation (PDE) such as the Heat Equation or Shrodinger's Equation. We do not include PDE's, because the goal of this work is simply to generalize the DeMO papers so as to address hybrid discrete-continuous simulations as well as basic physics engines used in animations.

4.2.2 Discrete-event Models

The major division of the field of modeling and simulation is between continuous-time and discrete-event models. Both have very large and long established communities. Discrete event models are very general and include discrete-time models. They can handle anything except an infinite number of infinitesimal changes which requires calculus. Discrete-event models are the focus of the Discrete-Event Modeling Ontology (DeMO) presented in (Miller, Baramidze, Fishwick, and Sheth 2004; Fishwick and Miller 2004; Miller and Baramidze 2005; Silver, Lacy, and Miller 2006).

In this case, model dynamics are simplified in that state changes can only occur at a countable number of points and hence a simulation may focus on these time points (also known as event occurrences). Therefore, the evolution of the model world is driven by events. Events represent things that can happen which may cause state changes (i.e., nothing else can cause the state $s(t)$ to change). Besides making state changes, events may also trigger other events to happen at the current time or in the future. The clock and transitions are given as follows: Letting $g(h) = 1$, we have

$$h = c(s(t), t) \tag{12}$$

$$s(t + h) - s(t) = f(s(t), t) \tag{13}$$

The causality due to events, in their most general form, is embedded in the clock c and transition functions f. However, it is often useful to think of the clock and transition functions as working together to advance the model forward in time to the next state, based on the event $e \in E$ that occurs. This is indicated by making the c and f functions parametrically dependent on e. Processing the event e advances the clock to the event's occurrence time $t + h$ and transitions the state $s(t)$ to the next state $s(t + h)$.

$$t + h = c(s(t), t; e) + t \tag{14}$$

$$s(t + h) = f(s(t), t; e) + s(t) \tag{15}$$

The event e is a point in a finite set E of event types such as {arrival, departure}. This now begs the question of how e is chosen. In general, determination of e can be complex, since events are created by other events and can indeed be canceled by other events. A future event may be created and put in, for example, a Future Event List (FEL). If the future event is not canceled, it will eventually come to the front of the time-ordered FEL, and become the imminent event to be processed next (i.e., used in the evaluation of c and f). Abstractly, this may be denoted by introducing activation and cancellation functions.

4.2.3 Discrete-time Models

For discrete-time models, the state $s(t)$ may change only at event occurrence times which happen with regularity, i.e., every h, a fixed constant number of time units. Although, h can be any fixed constant, it can also be rescaled to one (i.e., let $g(h) = h = 1$). Then the clock function simply returns 1 every time, while the state change is as follows.

$$s(t + 1) - s(t) = f(s(t), t) \tag{16}$$

This equation is a Difference Equation, which is the discrete analog to a differential equation. In this equation, the state s may be a discrete random variable. If we add the following restrictions:

1. Let the time be discrete, for example, let $T = \mathbb{Z}$.
2. Let the clock function be the successor function.
3. Let the transition be time homogeneous, that is, invariant over time.
4. Let E be a singleton set or equivalently the event is embedded into transition probabilities.

Then, the difference equation becomes the balance equation for Discrete-time Markov Chains

$$P(s(t+1) = s_j) = \sum P(s(t+1) = s_j | s(t) = s_i) P(s(t) = s_i) \tag{17}$$

where $P(s(t+1) = s_j)$ is the probability that the next state is s_j. DeMO gives a step-by-step development of more and more restrictive Markov models (e.g., from Generalized Semi-Markov Processes to Semi-Markov Process to Markov Chains).

4.2.4 Static Models

So far we have been simplifying how we deal with time. Obviously, the simplest thing to do is freeze time (or equivalently eliminate it all together). This moves us from a dynamic world view, to a static one. Although simulation is mainly concerned with dynamic models, static models (often called Monte Carlo models) are also useful.

4.3 Causality Based Classification

Causality is a well established principle in philosophy as well as classical physics. Some modern theories such as general relativity, quantum mechanics and string theory challenge the simple notion of causality. Yet for the simulations we are considering, we assume causality (or cause and effect). Causes or agents of change may cause changes in the current state $s(t)$ or even defer their effects to the future. The effects may introduce a gradual or sudden change. Sudden changes are captured as events that theoretically happen instantaneously. In reality they may not, but it is reasonable to represent them this way in the model. Gradual changes happen over time by an ongoing application of force. In physics, the primary causes of changes are forces such as the four fundamental forces: gravity, electro-magnetic, weak nuclear and strong nuclear forces.

We may classify models based upon a characterization of the causes of change. Change may be modeled discretely or continuously. In the discrete case, changes happen discretely at specific event times. Between these times, what is happening in the simulation may be ignored. Thus, discrete event simulations, for efficiency sake, jump discretely through time, processing event after event.

This may present a problem for animation in the following sense: After leaving one service center, a customer goes to the next. From a simulation point of view this time may be ignored. From a animation point of view the customer should smoothly go from one center to another. This can be handled by adding animation friendly events to the simulation, or by doing event interpolation and adjustment in an animation engine.

The bottom line is that between events, the system being modeled need not be static (e.g., entities or particles may be moving), it is just that these changes are not judged to be important for the purposes of the model. Typically, in higher fidelity models which more faithfully represent the system, more events will be represented and animations should look more realistic.

In summary, changes to the model may occur discretely or continuously. Discrete changes due to events make discontinuous changes (or jumps) to entities in the model world. Continuous changes due to forces make infinitesimal changes in an infinitesimal amount of time to produce typically smooth changes.

Note that the character of space-time does not determine whether changes occur discretely or continuously, in general, but clearly, continuous changes require space (or state) and time to be continuous. Since there are three concepts (time, state, change) where the discrete versus continuous dichotomy applies, there are eight distinct possibilities. Of these eight possibilities only five make sense, as shown in Table 2.

[Table 2 about here.]

4.4 Classification Based on Determinism

Determinism has been a hotly debated subject in philosophy and physics for a long time. In classical physics, mathematical models were basically deterministic. Probability was introduced to deal with lack of knowledge or just to simplify the model. Modern physics, however, postulates that probability is a fundamental part of reality. Therefore, we classify models as either deterministic or stochastic. For a deterministic model, given the input, the output is uniquely determined. However, for a stochastic model this is not necessarily the case since randomness is included in the model.

5 Adding Semantics to Simulation Models

In this section, we claim that adding semantics to simulation models is going to become more and more important in the future. A skeptic might counter-claim that modeling and simulation are general purpose techniques that achieve their usefulness through abstraction. In this way, a bank and drive through restaurant can be modeled in similar ways using abstract queues with different parameter values for inter-arrival and service times. It is great to be able to abstract out the essential features and discover fundamental similarities. The modeling and simulation community has been doing this successfully for decades. In our opinion, this paradigm has three weaknesses:

1. The mapping from the real world to the abstract model is largely in the mind of the simulation analyst.

2. High fidelity, multifaceted modeling is difficult to achieve.

3. Building models out of model components is limited, in part due to lack of semantics.

As the Semantic Web progresses, one might consider how it could positively impact the development and use of mathematical models in general and simulation models in particular. The purpose of a mathematical model is to create an abstract representation of a system or mini-world (be it real or artificial). As an abstract rather than concrete representation, the model could be mapped to multiple systems or mini-worlds. Being abstract, the model can be more easily analyzed and manipulated than an actual system. In this sense, abstraction is good. However, too much abstraction can result in loss of realism and meaningfulness. One way to lessen the loss is by increasing the amount of explicit semantics given.

Previously, simulation was concerned with getting the numbers right. Have we created a simulation model (validation) and a program implementation (verification) that produce accurate estimates or predictions? Animation puts an additional constraint on this. The time evolution of the model should "look right". Still, what is the relationship between the model and the system? What are the things moving around in the model? How do they compare to similar thing found in other models? How do they relate to existing knowledge such as the laws of classical physics?

A small step in this direction, is to document the model (or even the program implementing the model). However, this is likely to be minimal and certainly informal. An alternative would be annotate the model as well as the model elements so that, for example, one would know what an entity looks like and what it means. Defining the meaning of something is only feasible if related terms are already defined. Then clearly these related terms only make sense if terms related to them are defined. These terms should be logically organized into a well-defined conceptualization and made readily available using the Web. This is one of the central thrusts of the Semantic Web and Web based ontology.

Since simulation is used for modeling of a vast array of fields, the above prescription is really quite challenging. First ontology needs to be developed for simulation and modeling methodology. Then ontology from application domains (e.g., health care, transportation, etc.) needs to be utilized. Fortunately, many domains are developing ontology as shown for scientific domains in the Table 3.

[Table 3 about here.]

There are many more ontology listings on the Open Biomedical Ontologies (OBO) site ($obo.sourceforge.net/cgi-bin/table.cgi$) with 52 at last count.

The focus of this chapter, however, is more on ontology for simulation and modeling methodology. One way to start is to try to understand what a model is and what it is used for (its purpose). The word itself has many definitions (e.g., in Merriam-Webster's dictionary and in WordNet). We are interested in its usage for abstract, conceptual or mathematical models. From Wikipedia (Wikipedia 006b), we have the following definition:

- "An abstract model (or conceptual model) is a theoretical construct that represents physical, biological or social processes, with a set of variables and a set of logical and quantitative relationships between them. Models in this sense are constructed to enable reasoning within an idealized logical framework about these processes."

The purpose of a model or modeling in general, is even harder to capture. In an idealized sense, a model is the essence of science. Since the real world or real systems are so complex, models are constructed that can be manipulated logically or mathematically. The models help us dissect, understand and make predictions about the real world. For science to be self-correcting, the models (or hypothesis or theories) must be falsible. In other words, tests and experiments must be developed to show that the model has deficiencies that need to be corrected either by improving the model or throwing it out completely. Besides empirical validation, models need to be consistent with other models or theories.

Let us now examine in greater detail, the problem of defining or describing a model in terms of (i) statics and (ii) dynamics. The statics of an entity define its type (types of properties) and immutable state. The statics can be described at a high-level using, for example, a UML class diagram or and OWL ontology. The dynamics of an entity define its behavior. There are several ways to describe behavior in UML (e.g., Sequence Diagrams, Collaboration Diagrams, Statechart Diagrams or Activity Diagrams). In addition, other formalisms such as Process Algebras, Petri Nets, Bond Graphs, Activity Cycle Diagrams and Event Graphs may be used. The current state of affairs is that there are several competing approaches and none are as successful as the approaches used for statics. Clearly, the problem is much more difficult.

Ontology is ideal for describing things, so statics can be well handled. Dynamics or behavioral specifications are more challenging. Although knowledge representation languages in AI, such as Frames (Minsky 1974), support the use of procedural attachments, current Semantic Web initiatives are avoiding this complexity because of the need to effectively support querying and inferencing on a Web scale. Still there is, however, ongoing research work on behavioral specifications for Semantic Web Services. Behavioral units (such as operations) can be annotated with functional semantics, such as functional category, inputs, outputs, preconditions and postconditions (Sivashanmugam, Verma, Sheth, and Miller 2003). For simple cases, preconditions and postconditions (or effects) may be expressed using an ontology language like OWL, while for more complex conditions a rule language like SWRL is more suitable. One could apply such an annotation approach to simulation in several ways. For example, in the Event-Scheduling paradigm, the behavior can be captured in the logic of an occur operation (event routine). Similarly, in the Process-Interaction paradigm, the behavior can be captured in the logic of the entity's script (a network of operations).

The complexity of modeling dynamics is testified to by the plethora of modeling techniques used: Message Charts, Collaboration Diagrams, State Charts, Activity Diagrams in UML, the three Simulation World-Views, Event Scheduling, Process-Interaction and Activity Scanning in Simulation, as well as, Event Graphs, State Machines, Petri Nets, Process Models and Process

Algebras.

The goal is to capture what an entity does short of providing code to implement the behavior. (The specification should provide the basis for verification of the code, and hence, cannot be the code. Yet to allow automatic verification, the specification must be machine interpretable.) This is the essence of providing a semantic description of behavior. Entities can be coupled by (i) shared state, (ii) invocations or (iii) events, with each being more loosely coupled than the former. The complexity of verification goes up dramatically if the shared state space is large, and so it should be keep to a minimum. In the Software Engineering as well as the Agent and Semantic Web Services communities, the semantics of invocations is often modeled using Inputs, Outputs, Preconditions and Postconditions. Since Inputs and Outputs are objects, they may be described ontologically. For example, in the proposed Semantic Annotations for Web Services Description Language (SAWSDL) standard (based on WSDL-S) (Akkiraju, Farrell, Miller, Nagarajan, Schmidt, Sheth, and Verma 2005; Farrell and Lausen 2006) they may be annotated with model references to OWL or UML. Preconditions and Postconditions (alternatively Effects) may be modeled with a constraint language such as SWRL or UML's Object Constraint Language (OCL). Although, this approach can be used to describe an invocation, it says nothing about the sequencing or ordering of invocations, leading the need to describe interactions via a protocol specification. In order to more fully capture behavior, richer languages are necessary. Unfortunately, use of Turing-complete languages makes inferencing fundamentally challenging, leading to a tradeoff between the ability to capture detailed behavior versus the ability to analyze it.

We close this section by listing ontology in Table 4 used for modeling and simulation as well as ontology that could provide foundations for modeling and simulation.

<INSERT TABLE 4 HERE>

[Table 4 about here.]

6 Overview of the DeSO Ontology

The Discrete-Event Simulation Ontology (DeSO) is an initial attempt at providing a concise, but adequately precise ontology for the most fundamental concepts often referred to in modeling and simulation. Such an effort, however, is by no means easy, as to precisely define the basic concepts would mean to define many of the relevant concepts in mathematics, philosophy and physics. We have no intention of making DeSO a huge, self-contained ontology, yet we have taken significant measures to make it work far beyond its size.

The current DeSO includes the six elementary concepts in modeling and simulation, namely, time, space, physical entity, state, effector and model. We present an overall picture of how models are classified based on certain properties of these basic concepts as described in section 4. These concepts are also complemented in DeSO by some other related concepts to provide more accurate definitions and to reflect the complicated relations between them.

The first measure we have taken to compact DeSO is to start off the ontology by importing the Suggested Upper Merged Ontology (SUMO) rather than starting from scratch. The SUMO was developed at TeKnowledge to cover around 1000 of the most general concepts intended for use by middle-level and domain ontology like DeSO. We summarize a few of the advantages of importing an upper ontology below:

- First, an obvious advantage is the reuse of established knowledge system. DeSO, for example, uses directly many definitions and relations already in SUMO, such as TimePoint, Set, and FiniteSet.

- Second, an important use of ontology is to facilitate information sharing through the use of common vocabulary, as common vocabulary effectively reduces ambiguity in communication and facilitates machine understanding. Since OWL does not enforce the unique name assumption, the same class name may be used to refer to different concepts in different ontological specifications. [3] The following two approaches will guarantee that the same name in DeSO and SUMO refers to the same concept.

  The first approach is to use the needed SUMO class directly without reproducing the same concept in DeSO. This sometimes requires that we create some new DeSO classes whose existence is dependent on the SUMO. For example, if we need to define two new classes

[3] Part of developing ontology is to handle synonyms and homonyms. Different names for the same concept are synonyms, while different concepts with the same name are homonyms.

"DeterministicFunction" and "StochasticFunction" in DeSO and we can choose to use the superclass "Function" in the SUMO directly, then the two DeSO classes would be created as the subclasses of SUMO:Function. While this approach is favorable theoretically, current ontology editors like Protégé, do not support the notion of a package view as one would see in Javadoc, so the classes in the SUMO and DeSO are mixed up structurally, and the classes of DeSO may be buried deep inside a SUMO hierarchy. Such mixture often deprives us of the ability to freely create new class relationships and to generate visualization, and, thus, the freedom to express what we want to say in the ontology.

  Because of the limitations of the current ontology editors, common vocabulary between the SUMO and DeSO is realized through the second approach: using the equivalentClass restriction in OWL. Our way of using the SUMO classes is to generate new classes in DeSO and restrict these classes as equivalent to the classes in the SUMO where appropriate. We choose to use the same class names as in the SUMO for easy understanding, although identical classes names are not required for the equivalentClass restriction. For example, to conform to the naming system of the SUMO, the "Entity" class that we talked about in section 3 is called "PhysicalEntity" in DeSO. This latter approach provides us with desirable autonomy of DeSO, the flexibility of generating class relations as we want, as well as common vocabulary between the two ontological specifications. In DeSO, for example, a new TimePoint class is defined as equivalent to the SUMO TimePoint class and is used in all the relationships involving TimePoint in DeSO.

- Third, the use of an upper ontology makes inferences across different domains easier, as relations are established across OWL files by sharing the generic concepts in the upper ontology.

The second characteristic of DeSO is that it utilizes SWRL on top of OWL so that the expressiveness of the ontology increases considerably without substantial addition in size. For instance, we would have needed 24 additional classes for all the combinations of model types based on the four classification criteria we mentioned in Section 4; instead, we use 9 rules that express the same ideas and more. A very simple example of these SWRL rules is

$$isStochastic(?m, false) \rightarrow isDeterministic(?m, true) \tag{18}$$

which means "if a Model m is not stochastic, then m is deterministic." This short rule allows us to reuse the definition of Stochastic Model and saves us the trouble of defining Deterministic Model. However, rules are not always so short. More often than not, we need to write long rules to define some concepts. In DeSO, Stochastic Model is defined with the following rule:

$$\begin{aligned} &existsIn(?m, ?st) \wedge populatedBy(?st, ?pe) \wedge effectedBy(?pe, ?ef) \wedge computes(?ef, ?sf) \\ &\qquad \wedge StochasticFunction(?sf) \rightarrow isStochastic(?m, true) \end{aligned} \tag{19}$$

To put it in plain English, the rule says "if a Model m exists in some SpaceTime st, and st is populated by some PhysicalEntity pe, and pe is effected by some Effector ef which computes a StochasticFunction sf, then m is Stochastic." SWRL rules are more than simple definitions of concepts, in that they reflect the relations between the concepts in addition to the definitions of the concepts themselves.

DeSO has been developed using Protégé with its OWL plugin. The SWRL editor (an extension to the Protégé OWL plugin) has been used to edit the SWRL rules in DeSO. Figure 1 is a visualization of the classes in DeSO and their relationships created by OntoViz (one of the several popular visualization plugins for Protégé).

In short, DeSO is a middle-level ontology built upon the SUMO that includes the most fundamental concepts in the domain of modeling and simulation. As one of the first attempts at using SWRL in an ontology, DeSO achieves maximal expressiveness within minimal volume.

[Figure 1 about here.]

7 Overview of the DeMO Ontology

Work on the Discrete-Event Modeling Ontology (DeMO) began in 2003 (Miller, Baramidze, Fishwick, and Sheth 2004; Fishwick and Miller 2004) to explore issues and challenges in developing ontology for simulation and modeling. As its name suggests, it is focused on discrete events models, in which state changes discretely over time due to the occurrence of events. It used the OWL language to define over 60 classes and many properties. Figures 2-5 are some visualizations created by OntoViz showing the DeMO classes and their relationships. The ontology consists of four main parts: `ModelConcept`, `DeModel`, `ModelComponent` and `ModelMechanism`. `DeModel` is itself divided into four parts based on the three simulation world views plus a fourth representing state models, namely, `StateOrientedModel`, `ActivityOrientedModel`, `EventOrientedModel` and `ProcessOrientedModel`. [4]

As illustrated by the OBO site, it is better to have several (but not too many) interrelated ontology, rather than one huge monolithic ontology. Along these lines, DeMO as it is extended, could be divided into more than one ontology. In addition, DeMO ignores much of the simulation domain such as continuous models, statistical modeling, output analysis, random varieties, etc. Also, DeMO at present has few instances. One could attempt to populate the ontology (or

[4]Note, the images show DeMO version 1.8, which is missing the `ProcessOrientedModel` subtree, which is going into version 1.9.

knowledge-base) with information about simulation engines, available simulation models, model components, etc. This could be done by writing extractors for scanning the Web for information or by providing a mechanism for publication. Alternatively, one could simply use the ontology for annotation of simulation artifacts (as is done in the proposed WSDL-S (Akkiraju, Farrell, Miller, Nagarajan, Schmidt, Sheth, and Verma 2005) standard). Then special semantic search engines could precisely retrieve the information requested.

There can be several ways to approach the descriptions of different modeling formalisms (formalism specification) for the purpose of ontology engineering. One way is to consider each model separately and define them from scratch. This may be called a "problem in hand" approach - given a problem, define a modeling formalism that "fits" the problem well. This is a natural approach from a practical point of view: different modeling formalisms "fit" differently into different problems; some are more fitting for one purpose, some for another. Another way is to define some very general formalism and consider all other models to be some sort of sub-formalisms - restrictions on a general framework such as DEVS. This view is logical and natural as well, because many of the existing modeling approaches have a formal description and it is only a question of finding a general enough framework that encompasses all the existing formalisms and from which new sub-formalisms can be derived. However, if this philosophy is taken to the extreme, it can lead to unnecessary complexity and awkward notions.

DeMO utilizes a middle ground approach, where several general (upper-level) formalisms are defined independently of each other. (Of course they do not have to be completely independent of each other and may themselves be derived from some even more general formal framework.) These upper-level formalisms can be viewed as root classes for a taxonomic tree for the discrete-event modeling and simulation domain. All other modeling formalisms are defined as restrictions on one of the root classes.

[Figure 2 about here.]

[Figure 3 about here.]

[Figure 4 about here.]

[Figure 5 about here.]

Importantly, DeMO uses a uniform approach to a description process of a modeling formalism. Each DeModel is considered as having Model Components and Model Mechanisms (syntax and semantics of the model), which in turn are defined using fundamental Model Concepts. This approach allows for great flexibility and straightforwardness in constructing an ontology and defining new formalisms.

We close by giving a simple application for the DeMO ontology involving Petri nets. Figure 6 shows a screenshot from the Protégé ontology editor of the PetriNet class in DeMO. One thing to notice in this diagram is where the class fits in the class hierarchy. Another important aspect is the properties section. The former indicates how PetriNets relate to other modeling formalisms, while the latter indicates what is in a PetriNet (for example, the following properties, has-ActivitySet, has-ArcSet, has-Component, has-Mechanism, has-PlaceSet, has-TimeSet, time-Specified-by, define the structure and mechanics of a PetriNet). Since PetriNets are a very popular formalism, there are several simulators that run PetriNets. For the purposes of standardization and interoperability, the Petri Net Markup Language (PNML, (Jngel, Kindler, and Weber 2000)) has been created. Several of the simulators accept input in this format. One existing application of DeMO is the automatic generation of PNML specifications from instances stored in the DeMO ontology. Note that DeMO maintains topological information on the PetriNet, while PNML requires geometrical coordinates. Rules could be developed to select layout algorithms that will take the topological information and convert it into geometrical coordinates. This could lead to visually appealing animations of Petri net executions.

[Figure 6 about here.]

8 Summary

We have developed a general conceptual framework for modeling and simulation as represented by DeSO and DeMO shown in figures 1-5. The potential impact of Semantic Web research on the Modeling and Simulation communities has been discussed. In particular, the use of OWL and SWRL has been demonstrated in the DeSO ontology. Issues in the construction and use of ontology for modeling and simulation have also been addressed.

References

Akkiraju, R., J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma (2005). Web service semantics - wsdl-s. `http://www.w3.org/Submission/WSDL-S/`.

Berners-Lee, T. (1998). Why rdf model is different from the xml model. `http://www.w3.org/DesignIssues/RDF-XML.html`.

Berners-Lee, T., J. Hendler, and O. Lassila (2001). The semantic web. *Scientific American 284*(5), 34–43.

Brutzman, D. (2004). Extensible modeling and simulation framework (xmsf). `http://www.movesinstitute.org/xmsf`.

Caprotti, O., M. Dewar, and D. Turi (2004). Mathematical service matching using description logic and owl. http://monet.nag.co.uk/cocoon/monet/publicdocs/monet_onts.pdf.

Cellier, F. E. (1991). *Continuous System Modeling.* New York: Springer-Verlag.

Chen, P. P. (1976). The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems 1*(1), 9–36.

DEVS (2005). Devs. http://www.sce.carleton.ca/faculty/wainer/standard/.

Farrell, J. and H. Lausen (2006). Semantic annotations for wsdl. http://www.w3.org/2002/ws/sawsdl/spec/SAWSDL.html.

Fishwick, P. A. and J. A. Miller (2004). Ontologies for modeling and simulation: Issues and approaches. In *Proceedings of the 2004 Winter Simulation Conference (WSC'04)*, Washington, DC, pp. 259–264.

Genesereth, M. and R. Fikes (1992). *Knowledge Interchange Format, Version 3.0 Reference Manual.* Stanford, CA: Computer Science Department, Stanford University.

Gentzen, G. (1969). *Investigations into Logical Deduction.* North-Holland.

Hoare, C. (1969). An axiomatic basis for computer programming. *Communications of the ACM 12*(10), 576–585.

Horrocks, I., P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean (2003). Swrl: A semantic web rule language combining owl and ruleml. http://www.daml.org/2003/11/swrl/.

Johnston, P. (2005). Xml, rdf, and dcaps. http://www.ukoln.ac.uk/metadata/dcmi/dc-elem-prop/.

Jngel, M., E. Kindler, and M. Weber (2000). The petri net markup language. The Workshop AWPN.

Klyne, G. and J. J. Carroll (2004). Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

McGuinness, D. L. and F. Van Harmelen (2004). Xml, rdf, and dcaps. http://www.w3.org/TR/owl-features/.

Miller, G., R. Beckwith, C. Fellbaum, D. Gross, and M. K. (1990). Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography 3*(4), 235–244.

Miller, J. A. and G. Baramidze (2005). Simulation and the semantic web. In *Proceedings of the 2005 Winter Simulation Conference (WSC'05)*, Orlando, FL, pp. 2371–2377.

Miller, J. A., G. Baramidze, P. A. Fishwick, and A. P. Sheth (2004). Simulation and the semantic web. In *Proceedings of the 37th Annual Simulation Symposium (ANSS'04)*, Arlington, Virginia, pp. 55–71.

Minsky, M. (1974). A framework for representing knowledge. *MIT-AI Laboratory Memo 306*.

Mitkov, R. (2003). *The Oxford Handbook Of Computational Linguistics*. Oxford University Press.

Niles, I. and A. Pease (2001). Towards a standard upper ontology. In C. Welty and B. Smith (Eds.), *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Ogunquit, Maine.

Noronha, N. and M. J. Silva (2004). Using the semantic web for web searches. `http://xldb.di.fc.ul.pt/data/Publications_attach/NormanPaperInteraccao2004.pdf`.

of Philosophy, S. E. (2006). Stanford encyclopedia of philosophy. `http://plato.stanford.edu/contents.html`.

OpenMath (2006). Openmath. `http://www.openmath.org/cocoon/openmath/index.html`.

Palmer, M. (2004). Verbnet. `http://www.cis.upenn.edu/~mpalmer/project_pages/VerbNet.htm`.

Plotkin, G. D. (1981). A structural approach to operational semantics. *Tech. Rep. DAIMI FN-19*.

Pritsker, A. A. (1986). *Introduction to Simulation and SLAM II (3rd ed.)*. New York, NY: John Wiley & Sons, Inc.

Reichenthal, S. (2002). Srml: A foundation for representing boms and supporting reuse. In *Proceedings of the 2002 Fall Simulation Interoperability Workship*, Orlando, Florida.

Rumbaugh, J., I. Jacobson, and G. Booch (1998). *The Unified Modeling Language Reference Manual (Addison-Wesley Object Technology Series)*. Essex, UK: Addison-Wesley Longman Ltd.

Scott, D. and C. Strachey (1971). Toward a mathematical semantics for computer languages. In *Proceedings of the Symposium on Computers and Automata*, New York, NY, pp. 19–46.

Shaya, E., B. Thomas, P. Huang, and P. Teuben (2006). Astroonto. `http://archive.astro.umd.edu/`.

Sheth, A., C. Ramakrishnan, and C. Thomas (2005). Semantics for the semantic web: the implicit, the formal and the powerful. *International Journal on Semantic Web and Information Systems 1*(1), 1–18.

Silver, G., L. Lacy, and J. A. Miller (2006). Ontology based representations of simulation models following the process interaction world view. In *Proceedings of the 2006 Winter Simulation Conference*, Monterey, CA, pp. 1168–1176.

Sivashanmugam, K., K. Verma, A. P. Sheth, and J. A. Miller (2003). Adding semantics to web services standards. In *Proceedings of the 1st International Conference on Web Services*

(ICWS'03), Las Vegas, Nevada, pp. 395–401.

Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Pacific Grove, CA: Brooks/Cole Publishing Co.

Tarski, A. (1983). *Logic, Semantics, Metamathematics (2nd edition).* Indianapolis, IN: Hackett.

Wikipedia (2006b). Wikipedia. `http://en.wikipedia.org/wiki/Model_\%28abstract\%29`.

Wikipedia (2006c). Wikipedia. `http://www.wikipedia.org/`.

Appendix: Semantics: Some Perspectives

Semantics has been a major topic of inquiry for long time. As a traditional branch of linguistics, it refers to the study of meaning of language. Deeply rooted in philosophy, semantics was first formalized in logic in the 19th century and was later expanded to deal with programming-language semantics. Now, the Semantic Web Initiative is bringing new life to this research and is attempting to make it practical and scalable at the Web level. In this appendix, we look at semantics from all these viewpoints: Philosophy, Linguistics, Logic, Programming Languages and the Semantic Web.

Linguistics divides the world distinctly into "language" and "meaning". Words, go in a lexicon, axioms encoding meaning go in an ontology, and semantic lexicographers create bi-directional mappings between the two. The lexicon maps words to concepts, listing multiple concept types for words that have more than one meaning. With many variations of notation and terminology, the basis for most systems in computational linguistics consists of (Sowa 2000): (i) lexicon - set of symbols, (ii) grammar - rules governing the ordering of symbols, and (iii) ontology - a topology of concepts. A mapping that involves all three serves as a foundation for establishing meaning.

Given a description in a formal language, a difficult question is "what does it mean". Such a description, as with natural language, will contain objects/nouns and actions/verbs. The meaning of the nouns and verbs may be refined using adjectives and adverbs, respectively. The elements of the description can therefore be naturally decomposed into the two parts: objects and actions.

The study of the nature of objects has long been studied in the field of ontology. First defined by Aristotle as "the science of being *qua* being," ontology studies the existence of things. It concerns the nature and meaning of existence as well as the basic categories of entities. Ontology is considered fundamental because it tells people what words refer to entities, and it provides the classification (including classes and subclasses) of entities as well as their properties and relationship to other entities. Languages for modeling ontology (or the related notion of schema) include the Entity-Relationship model (Chen 1976), Unified Modeling Language (Rumbaugh, Jacobson, and Booch 1998), Knowledge Interchange Format (Genesereth and Fikes 1992), Resource Description Framework (Klyne and Carroll 2004), Web Ontology Language (McGuinness and Van Harmelen 2004) and Semantic Web Rule Language (Horrocks, Patel-Schneider, Boley, Tabet, Grosof, and

Dean 2003). The last four of these languages are based on logic, primarily, description logic and first order predicate logic.

Now that we have a way of describing entities, statically, we need to describe their dynamics. Issues of behavior and interaction come to the forefront. One might look for an analog to ontology used to describe nouns, objects or entities, that would work for verbs or actions. Unfortunately, dynamics is much more challenging that statics. The first phase of science is to describe the entities (e.g., genes and proteins), while the second phase is to describe (better yet predict) how the will behave or interact (e.g., biochemical pathway models). Dynamic models involve entities that change (appear, disappear, move, change properties and affect others) over time.

Verbs are most naturally captured in ontology as relationships such as "student A *enrolls in* course B". However, this begs the question, what does *enrolls in* mean. The verb is not so much modeled as it is used in the model of student. Still, one could in OWL define the *enrolls in* property to be a subproperty of *takes* to claim some semantics is provided. Some attempts at verb classification have been done, e.g., see VerbNet (Palmer 2004).

A more complete treatment of dynamics calls for space-time models which have a collection of interacting entities that change over time. (Note, for generality, space is often represented abstractly as state which may include coordinates as well as other types of information).

In formal logic, semantics provides a way to show that a statement (logical expression) is true. The most prevalent approach is model-theoretic semantics (Tarski 1983). A logical expression consists of constants, variables, logical connectives, functions and predicates. In first-order logic, variables can be quantified, while in second order logic, functions and predicates may be quantified. Unless, the expression is a tautology, a model is required to determine its truth value. The model will indicate the domain that variables can range over, as well as, how to evaluate the functions and predicates. If the model relates to something meaningful (e.g., a part of the real-world) then the expression can be meaningfully interpreted. There are also other alternative approaches such as proof-theoretic semantics (Gentzen 1969).

The semantics of programming languages formally or mathematically deals with the meaning of programming languages. The symbols and the allowable orderings of these symbols are defined using the language's lexicon (what symbols) and grammar (what order). The lexicon is often described using a regular language, while the grammar is often defined using a context-free language. Together these constitute the syntax of the language. Capturing what a sequence of symbols means is not so easy. For example, what does x + y mean? Does the addition operator mean integer addition, floating point addition or string concatenation? There are three approaches to defining the meaning of programs: denotational semantics, operational semantics and axiomatic semantics (Hoare 1969; Scott and Strachey 1971; Plotkin 1981).

There is an ongoing debate about whether the Semantic Web is really semantic (i.e., will it explicate the meaning of resources on the Web). This debate involves open issues in philosophy and science, which are not likely to resolved any time soon. Hence, we simply claim the approach makes things “more” meaningful, in the sense of being easier to find, use and understand. Whether the machine truly understands it, is an issue for others to tackle.

List of Figures

Figure 1: DeSO Visualization

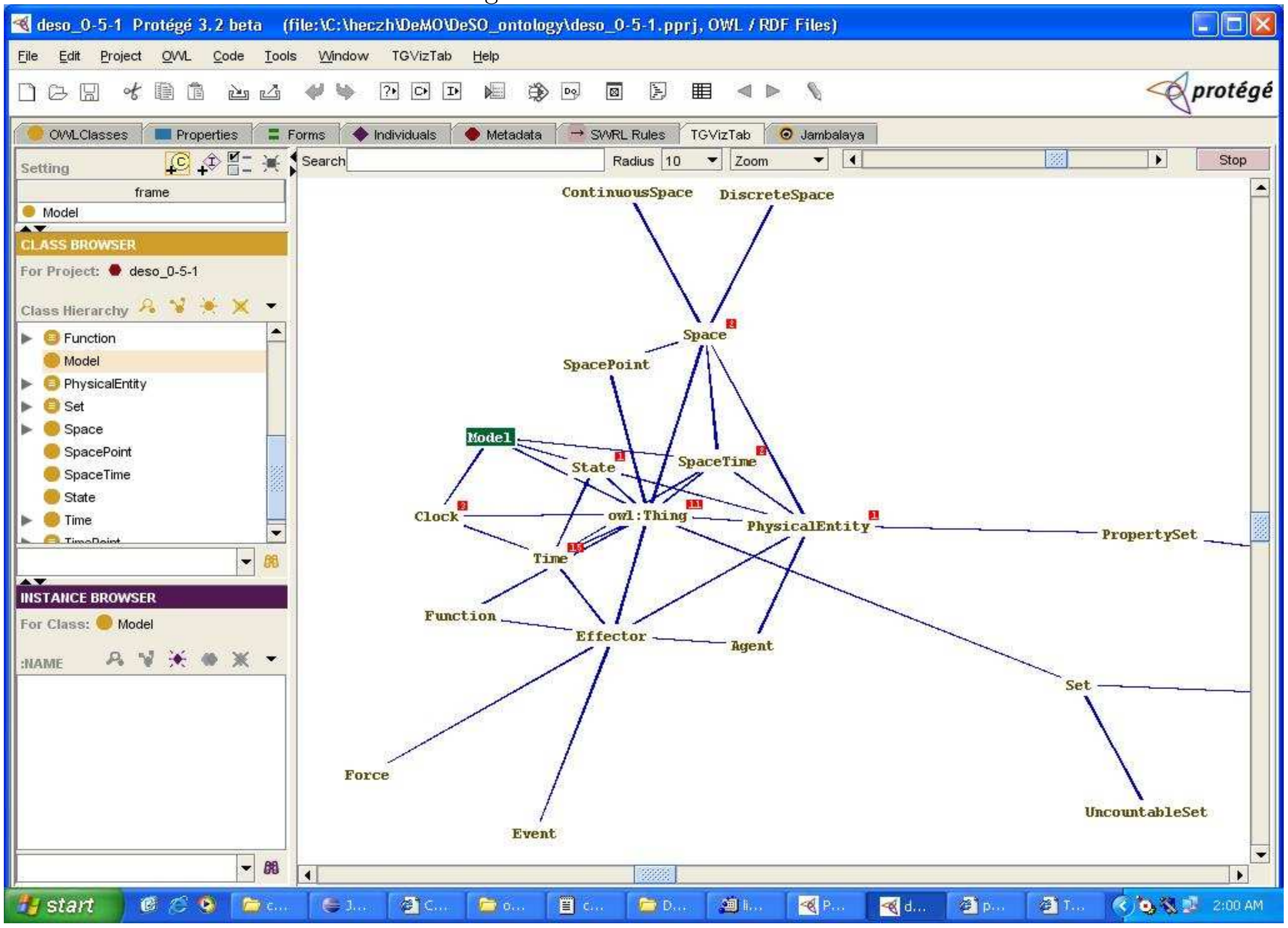


Figure 2: DeMO Model Concept

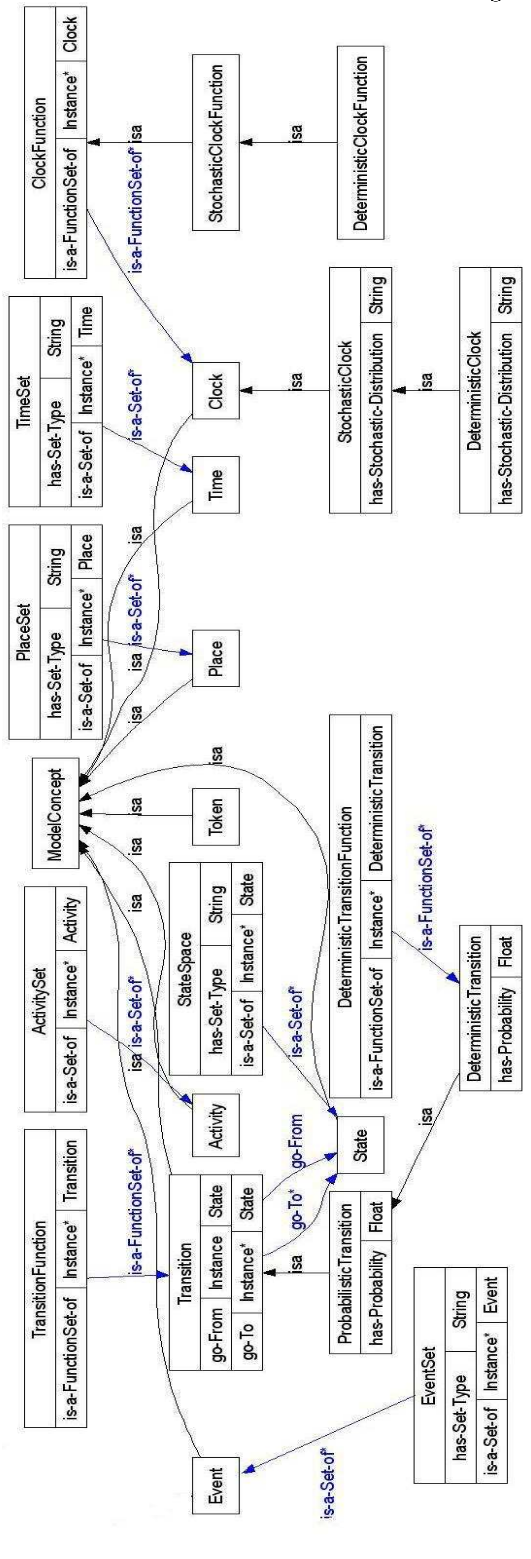


Figure 3: DeMO Model Component Class Hierarchy

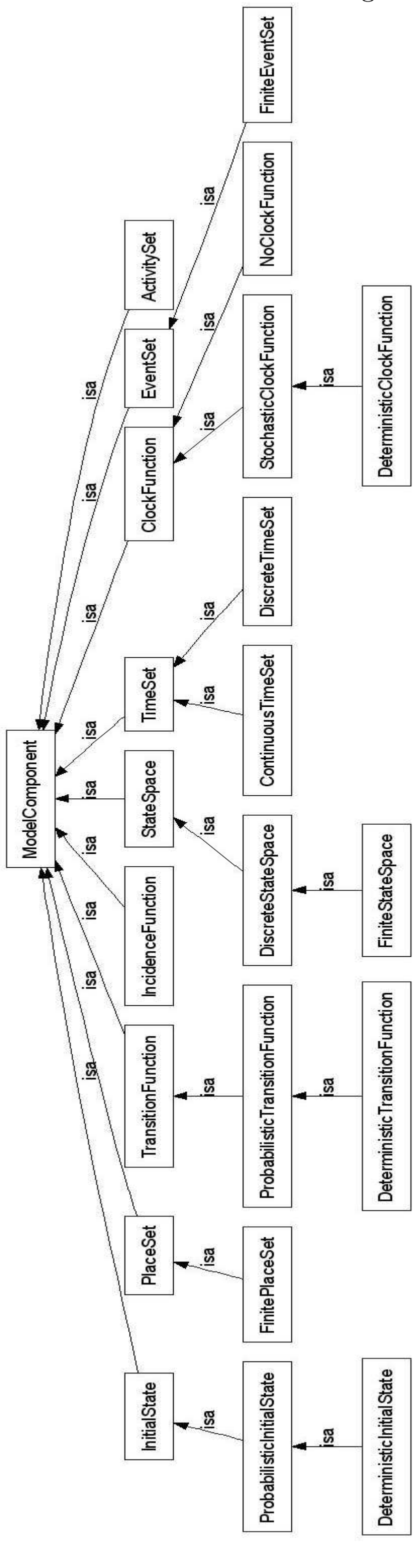


Figure 4: DeMO Model Mechanism

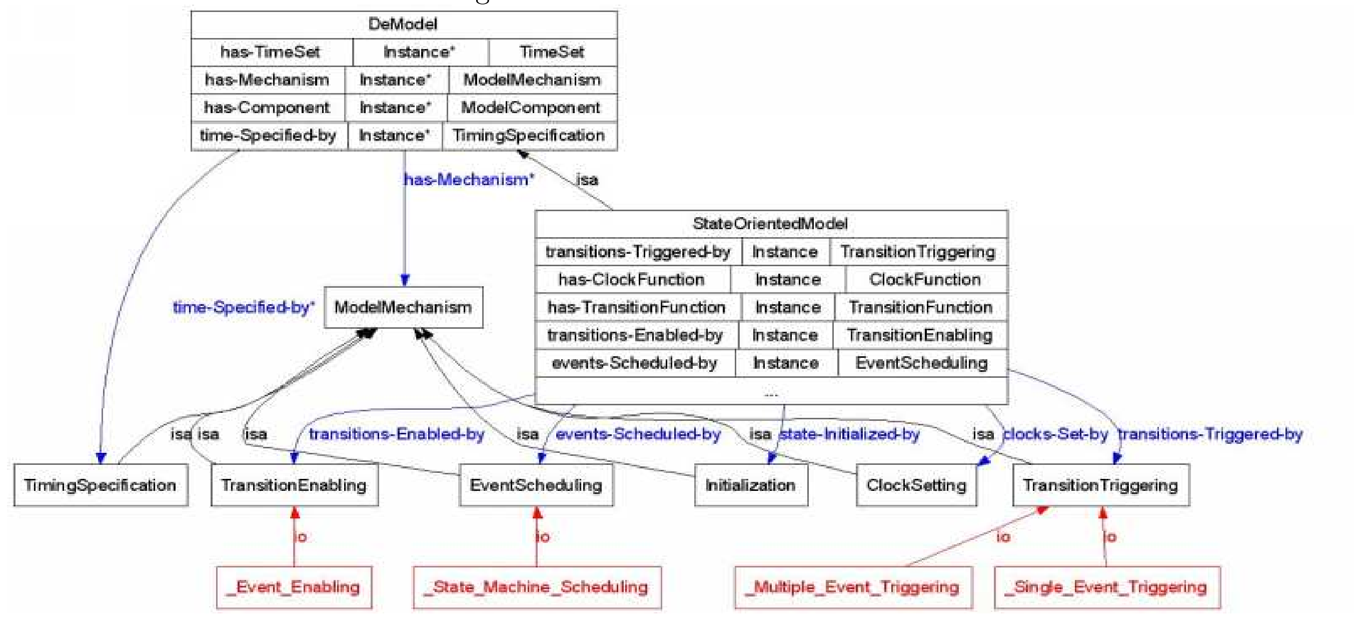
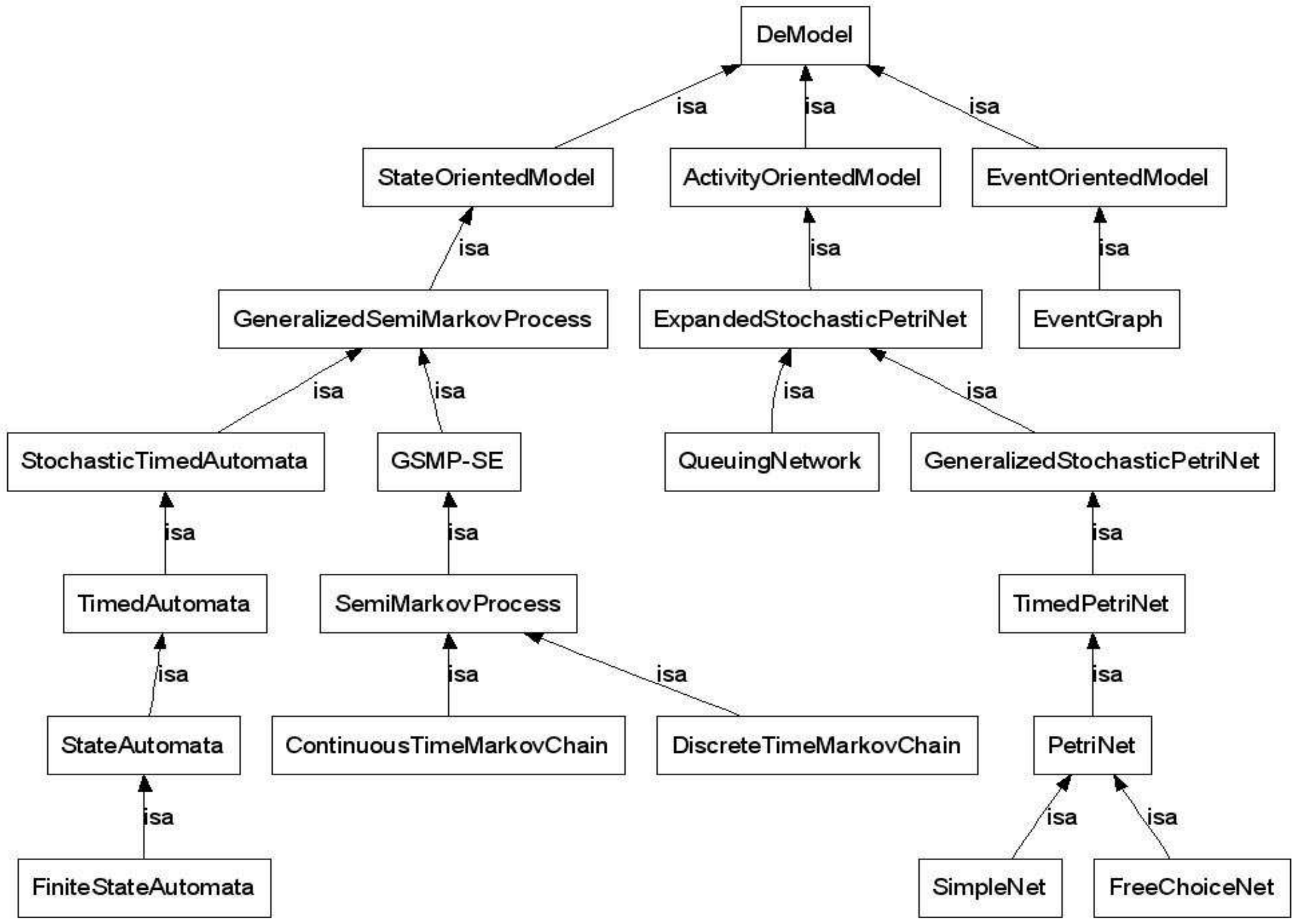

DeModel
has-TimeSet | Instance* | TimeSet
has-Mechanism | Instance* | ModelMechanism
has-Component | Instance* | ModelComponent
time-Specified-by | Instance* | TimingSpecification
has-Mechanism*
isa
StateOrientedModel
transitions-Triggered-by | Instance | TransitionTriggering
has-ClockFunction | Instance | ClockFunction
has-TransitionFunction | Instance | TransitionFunction
transitions-Enabled-by | Instance | TransitionEnabling
events-Scheduled-by | Instance | EventScheduling
...
time-Specified-by*
ModelMechanism
isa
transitions-Enabled-by
events-Scheduled-by
state-Initialized-by
clocks-Set-by
transitions-Triggered-by
TimingSpecification
TransitionEnabling
EventScheduling
Initialization
ClockSetting
TransitionTriggering
io
_Event_Enabling
_State_Machine_Scheduling
_Multiple_Event_Triggering
_Single_Event_Triggering

Figure 5: DeMO DeModel Class Hierarchy



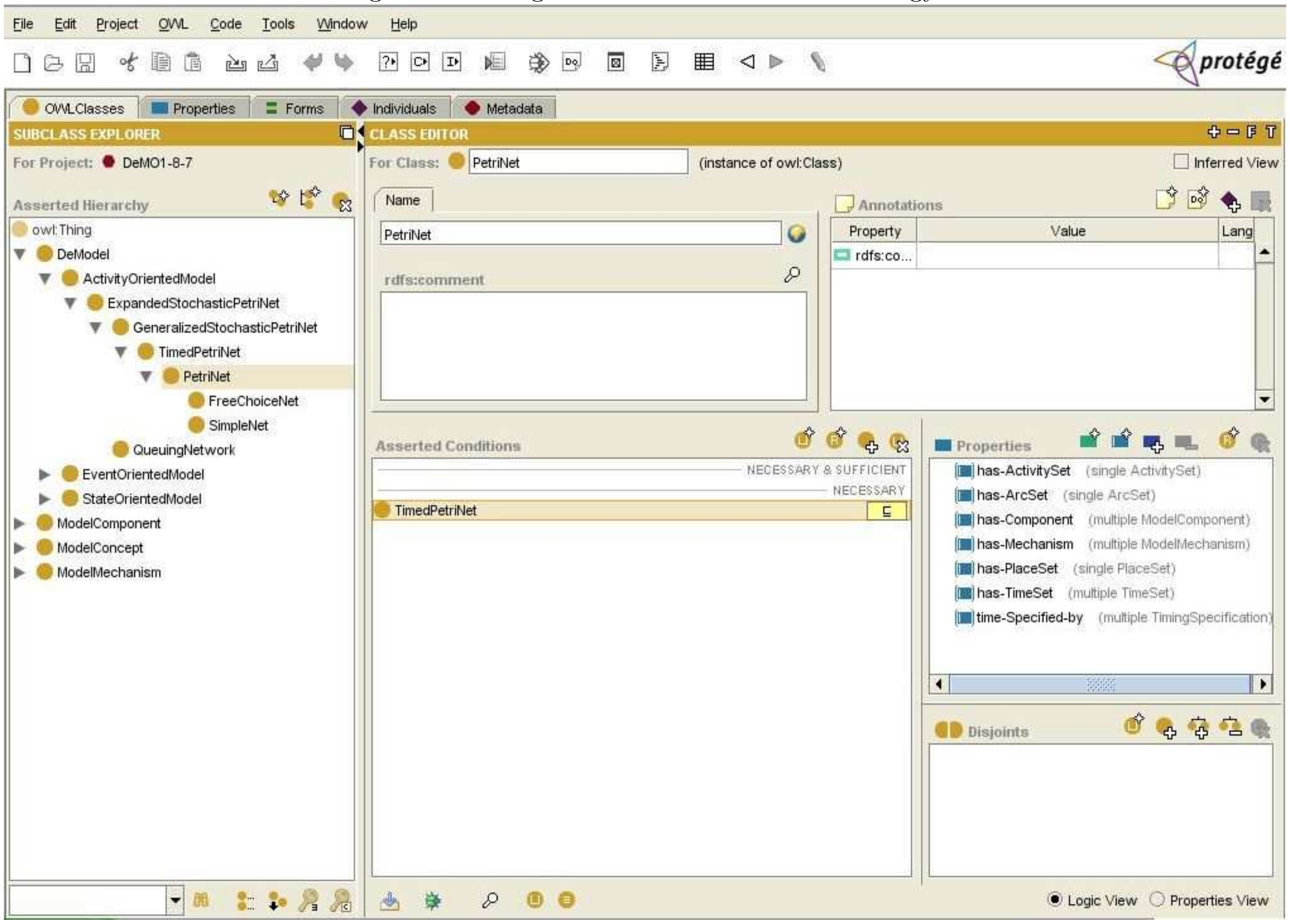


Figure 6: Protégé Screenshot of DeMO Ontology

For Project: DeMO1-8-7

- owl:Thing
  - DeModel
    - ActivityOrientedModel
      - ExpandedStochasticPetriNet
        - GeneralizedStochasticPetriNet
          - TimedPetriNet
            - PetriNet
              - FreeChoiceNet
              - SimpleNet
        - QueuingNetwork
    - EventOrientedModel
    - StateOrientedModel
  - ModelComponent
  - ModelConcept
  - ModelMechanism

For Class: PetriNet (instance of owl:Class)

Name: PetriNet

rdfs:comment

Property	Value	Lang
rdfs:co...		

NECESSARY & SUFFICIENT

NECESSARY

TimedPetriNet ⊑

- has-ActivitySet (single ActivitySet)
- has-ArcSet (single ArcSet)
- has-Component (multiple ModelComponent)
- has-Mechanism (multiple ModelMechanism)
- has-PlaceSet (single PlaceSet)
- has-TimeSet (multiple TimeSet)
- time-Specified-by (multiple TimingSpecification)

List of Tables

Acronym	**Language**	**Schema**	**Complexity of $\sqsupseteq$**
XML	eXtensible Markup Language	DTD, XSD, Relax NG	–
RDF	Resource Description Language	RDFS	PTIME
OWL-Lite	Web Ontology Language	OWL Schema Portion	EXPTIME-Complete
OWL-DL	Web Ontology Language	OWL Schema Portion	NEXPTIME-Complete
OWL-Full	Web Ontology Language	OWL Schema Portion	Semi-decidable
SWRL	Semantic Web Rule Language	–	Semi-decidable

Table 1: Semantic Web Languages.

State	**Time**	**Change**	**Model Type**
C	C	C	Continuous System Simulation
C	C	D	Discrete Event Simulation
D	C	D	e.g., Generalized Semi-Markov Processes
C	D	D	e.g., Time Series
D	D	D	e.g., Discrete-time Markov Chains

Table 2: Discrete (D) vs. Continuous (C).

Name	**Title**	**Domain**
PhysicsOnto	Ontology of Physics	Physics
	archive.astro.umd.edu/ont/Physics.owl	
AstroOnto	Ontology of Astronomy	Astronomy
	archive.astro.umd.edu/ont/Astronomy.owl	
ChemOnto	Chemical Ontology	Chemistry
	www.personal.leeds.ac.uk $\sim$ *scs1tvp/onto/chemonto.owl*	
GO	Gene Ontology	Biology
	archive.godatabase.org/latest $-$ *termdb/go_daily* $-$ *termdb.owl.gz*	
SO	Sequence Ontology	Biology
	cvs.sourceforge.net/viewcvs.py/song/ontology/so.obo	
MDEG	Microarray Gene Expression Data	Biology
	mged.sourceforge.net/ontologies/MGEDOntology.owl	

Table 3: Ontology for Scientific Domains

Name	**Title**	**Domain**
Monet	Mathematics on the Web	Mathematics
	www.cs.man.ac.uk/ ~ dturi/ontologies/monet/allmonet.owl	
GeomOnto	Ontology of Geometry	Mathematics
	archive.astro.umd.edu/ont/Geometry.owl	
StatOnto	Ontology of Statistics	Statistics
	archive.astro.umd.edu/ont/Statistics.owl	
DeMO	Discrete-event Modeling Ontology	Simulation
	chief.cs.uga.edu/ ~ jam/jsim/DeMO/	
DeSO	Discrete-event Simulation Ontology	Simulation
	chief.cs.uga.edu/ ~ jam/jsim/DeSO/	
MSOnto	Agent Ontology for Modeling and Simulation	Simulation
	www.nd.edu/ ~ schristl/research/ontology/agents.owl	

Table 4: Ontology for Modeling and Simulation