

Investigating Ontologies for Simulation Modeling

John A. Miller
Gregory T. Baramidze
Amit P. Sheth
Computer Science Department
University of Georgia
Athens, GA 30602, U.S.A.

Paul A. Fishwick
CISE
University of Florida
Gainesville, FL 32611, U.S.A.

Abstract

Many fields have or are developing ontologies for their subdomains. The Gene Ontology (GO) is now considered to be a great success in biology, a field that has already developed several extensive ontologies. Similar advantages could accrue to the simulation and modeling community. Ontologies provide a way to establish common vocabularies and capture domain knowledge for organizing the domain with a community wide agreement or with the context of agreement between leading domain experts. They can be used to deliver significantly improved (semantic) search and browsing, integration of heterogeneous information sources, and improved analytics and knowledge discovery capabilities. Such knowledge can be used to establish common vocabularies, nomenclatures and taxonomies with links to detailed information sources. This paper investigates the use, the benefits and the development requirements of Web-accessible ontologies for discrete-event simulation and modeling. As a case study, the development of a prototype OWL-based ontology for modeling and simulation called the Discrete-event Modeling Ontology (DeMO) is also discussed. Prototype ontologies such as DeMO can serve as a basis for achieving broader community agreement and adoption of ontologies for this field.

1 Introduction

One of the ways in which a field matures is through the generation of taxonomies and more recently ontologies. These reflect how words, phrases and concepts representing domain semantics can be grouped and interrelated. Simulation is not that different from other fields in computing, where taxonomies exist on paper, but it is still rather difficult to connect what one research group is doing in Activity Diagrams (Birtwistle, 1979), for instance, with what another group is doing in Event Graphs (Schruben, 1983). How do the components of

the models connect? Up until recently, this was somewhat of an academic question since one might relate terms and phrases on paper, but this seems less than complete. What is needed is an efficient way of effecting links between concepts so that these linkages can be subsequently employed in databases, query engines, and human-computer interaction models. In addition, such organization of knowledge within a field helps increase the interoperability, integration and reuse of simulation artifacts. In the simulation and modeling domain, such artifacts include libraries, components, simulators, animators, simulation tools, and models. Associated with these artifacts are documents such as manuals, tutorials, papers and result repositories.

Important benefits to the modeling and simulation community can accrue through the use of new emerging Web technology, so that work may be done in more coordinated or cooperative ways. Prior such efforts included considerable work on Web-Based Simulation (Fishwick, 1996; Nair et al., 1996; Page et al., 2000; Miller et al., 2001). Web-based simulation began with the concept that the Web would have a profound influence on the way in which we do simulation. Much of this work was focused on infrastructure to support Web-based simulation. More recently, the eXtensible Markup Language (XML) has resulted in researchers developing simulations using XML applications and schemas (Fishwick, 2002; Kim et al., 2002). Although XML documents with their use of more meaningful tags carry some semantics (at least to humans), XML alone will not get the job done, as it does not provide enough semantics to achieve the goals of discovery, interoperability, integration and reuse.

A useful step forward is to provide a way to find and organize modeling and simulation information, knowledge and artifacts. One could of course use a Web Search Engine (e.g., Google www.google.com). If one is looking for a discrete-event simulator implementing the process-interaction world view, one could give Google a set of keywords. For example, the search string

”discrete-event simulation” + ”process-interaction” gave 803 hits. Manually, sifting through the 803 documents can be quite arduous. It is also frustrating to be looking for a software tool and get mainly papers from the search engine. Semantics allows the use of context to reduce such problems.

Current research and development into what some call the next generation of the Web, referred to as the Semantic Web (Berners-Lee et al., 2001), promises to transform the Web by providing machine-processable and meaningful descriptions of Web resources. This can improve discovery, integration and use/reuse of Web resources, and we believe it also holds significant promise for the Simulation and Modeling community (Lacy, 2001).

This grand effort to transform the Web into something more meaningful involves the creation of several Web languages and their supporting tools (see www.w3c.org). The use of XML in place of HTML improves structure (valid XML documents must comply with a DTD) and uses tags with more application-oriented meaning. Another way to improve search is to create descriptions of the content of Web pages. The Resource Description Framework (RDF) is used to make statements about resources which indicate their properties and relationships. As RDF collections get large, they themselves need to be organized. This is typically done with another Web language: RDF-Schema (RDF-S), DAML+OIL or nowadays principally Ontology Based Language (OWL). Traditionally, one could view these as a schema, much like a database schema. However, remember the Web is the target, so these schemas need to be more universal (they are not for a particular application or simply to represent the structure of a database). They should be sharable, independent of particular resources (e.g., documents or applications), understandable by humans, processable in meaningful ways by computers, and more complete for a domain. Essentially, they should describe the way things are for a particular domain. Meeting all these goals takes one beyond database schemas into the field of knowledge representation.

Much of this work involves the use of ontologies (Gruber, 1995) to define terms or concepts in certain domains (narrow and deep ontologies). For a particular domain, types of things are defined as classes, which have properties and relationships. The meaning of a concept is typically given in natural language. Meaning is also captured via the class’s position within a taxonomy (subclass-of/is-a hierarchy) as well as its properties, relationships and restrictions. Unlike schemas or data models which are application or data access oriented (i.e., the point is to get the job at hand done), ontologies are meant to define a domain and

to be shared and used by many (Denny, 2002). In a way, it provides semantics by agreement (we agree that this term should mean this). Most useful ontologies are therefore created by expert groups (e.g, GO at www.geneontology.org). Other examples of ontologies used in scientific domains include *EngMath* in Engineering Math, *EHEP* in Physics, *OntoNova* in Chemistry.

There are also efforts to define broad and shallow ontologies, called upper ontologies, such as the Suggested Upper Merged Ontology (SUMO) at ontology.teknowledge.com or the Standard Upper Ontology at suo.ieee.org. Proponents indicate that this facilitates the use of multiple ontologies and greater integration. Another school of thought views monolithic upper ontologies as too ambitious and inflexible. Fixing an upper ontology that numerous domain ontologies (and therefore applications and data models) are dependent upon may lead to a maintenance nightmare.

There has been much prior work on Web-Based Simulation, which takes advantage of Web technologies and availability to enhance simulation and modeling. We have discussed the general problem of linking simulation terminology, and how Web-Based Simulation has led us into XML-based approaches. A new wave of technology is emerging that should have an even greater impact on modeling and simulation, namely, the Semantic Web and Web Services (Silver et al., 2002). One of the most important contributions of the Semantic Web is the creation of ontologies for specific domains, which is the focus of this paper. In section 2, we overview the Ontology Web Language. We then briefly review prior taxonomies for modeling and simulation in section 3. Section 4 discusses the use and development of ontologies for modeling and simulation. We present strategies and issues that came up as we developed a prototype ontology called DeMO, the Discrete-event Modeling Ontology. DeMO is offered as an initial case study and impetus for the formation of expert groups to create standard ontologies for discrete-event modeling. The paper is wrapped up with conclusions and future work in section 5.

2 Ontology Web Language

The Ontology Web Language (OWL) has been approved by the World Wide Web Consortium (W3C) to be the standard language for expressing Web accessible ontologies. It incorporates features from and supplants its predecessors (RDF, RDF-S, DAML, DAML+OIL).

Compared to RDF-S, OWL adds more feature for describing properties and classes such as relations between classes (e.g., disjointness), richer typing of properties, cardinality of properties, and characteristics of

properties (e.g., symmetry, transitivity). As a revision of the DAML+OIL ontology language, OWL is built upon the lessons learned from the design and application of DAML+OIL (McGuinness and van Harmelen, 2003).

OWL comes in three flavors: OWL Lite which has expressiveness similar to RDF-S, OWL DL based on description logics which are computationally complete and decidable. OWL Full which provides additional feature and sacrifices decidability (McGuinness and van Harmelen, 2003).

Our ontology is represented in OWL DL and was developed using Protégé 2000 (*protege.stanford.edu*) with the OWL and EzOWL plugins. Protégé 2000 is currently one of the most popular ontology editing tools, see (Denny, 2002) for a survey. OWL DL allows the definition of a class to represent entities of a certain kind. Classes may (i) be divided into subclasses (ii) have properties/relationships, and (iii) have instances. Properties/relationships may have certain characteristics (e.g., transitive, symmetric, functional, inverse, cardinality). As such it shares much in common with Entity-Relationship Modeling (ERM), and the Unified Modeling Language (UML). ERM is intended for data modeling, UML is intended for software modeling, and OWL is intended for ontology/knowledge modeling. Compared to UML Class Diagrams, OWL DL permits the creation of subproperties as well as more ways of restricting properties/relationships.

3 Taxonomies for Discrete-Event Simulation and Modeling

Surveying existing taxonomies is a useful way to begin the development of an ontology, hence we give a very brief review.

Given a system $S(t)$ evolving over time, a model can be viewed simply as an approximation $M(t)$ of $S(t)$. Commonly, models are classified according to how they deal with time (*Static vs. Dynamic*), state (*Discrete vs. Continuous*) and randomness (*Deterministic vs. Stochastic*).

Models can be classified based on various characteristics: static vs. dynamic, time-varying vs. time invariant, continuous state vs. discrete state, time-driven vs. event-driven, descriptive vs. prescriptive, analytic vs. numeric (Zeigler, 1976; Page, 1994). In particular, discrete event simulation models can be defined as abstract, dynamic, descriptive, numerical models (Page, 1994). (Cassandras and Lafortune, 1999) define discrete-event system models as discrete-state, event-driven, time-invariant, dynamic models. Within the class of discrete event models further subclassifi-

cation may be achieved by the means of the following characteristics: stable vs. unstable models, steady-state vs. transient model, probabilistic vs. stochastic, and autonomous (no input) vs. nonautonomous models (Page, 1994). (Nance, 1993) divides simulation models into three main classes: Monte-Carlo, continuous, and discrete event. The International Council on Systems Engineering maintains and updates a broad taxonomy for software tools including simulation tools (INCOSE, 2002). (Schruben and Roeder, 2003) propose a new way of organizing the highest levels a modeling taxonomy for discrete event simulation based on a dichotomy between resident entity cycle modeling and transient entity flow modeling.

A natural way to create a discrete-event modeling taxonomy is to take an implementation viewpoint and start with the main simulation world-views (Fishman, 1973). Such a taxonomy, in a more general context, which includes simulation, database and workflow modeling, was discussed in (Miller et al., 1997) and a portion of it is summarized below.

For discrete-event simulation modeling three main world views (simulation modeling paradigms) are traditionally considered: event-scheduling, activity-scanning and process-interaction.

Event-Scheduling (ES). This approach focuses on the events that can occur in a system. An event instantaneously transforms the state of the system and/or schedules future events. Future events are scheduled by placing them on a future event-list. Time is advanced by jumping to the timestamp of the earliest event (imminent event) on the list and processing that event. An example of a modeling formalism that fits nicely in this world view is an *Event Graphs (EG)* formalism. In an event graph, nodes represent events, while directed edges represent causality (Schruben, 1983). Edges can be annotated with time delays and conditions.

Activity-Scanning (AS). In this approach, the models focus on activities and their preconditions (triggers). An activity consists of an event-pair (a start event and an end event). Activities are scheduled when their preconditions become true. The **Three-Phase (TP)** approach may be considered to be a more efficient variant of Activity-Scanning or a hybrid of Activity-Scanning and Event-Scheduling. Efficiency is gained by putting unconditional events on a future-event list as in ES and still triggering conditional events as in AS. *Activity Cycle Diagrams (ACD)* and *Petri Nets (PN)* are modeling formalisms following this world view. Activity Cycle Diagrams (ACD) are graphs with two types of nodes (bipartite graphs), activities and wait states, where the arcs connect either activities to wait states or wait states to activities (Tocker, 1963; Pidd, 1992). These diagrams depict the life-cycles of interacting en-

entities flowing through a system. Similarly, Petri Nets (PN) are graphs with two types of nodes (bipartite graphs), transitions and places, where the arcs connect either transitions to places or places to transitions (Petri, 1962; Peterson, 1977). A transition will fire if there is a (or are enough) token at each of its input places. In Timed Petri Nets, transitions have delays associated with them.

Both Activity Cycle Diagram and Petri Net models may fit in either the activity-scanning or process-interaction world views.

Process-Interaction (PI). This approach focuses on processes and their interaction with resources. A process captures the behavior of an entity as it flows through the system, step by step. Examples include *Activity Diagrams (AD)* and *Network Diagrams (ND)*. Activity diagrams are graphs consisting of a well defined set of functional nodes such as start, terminate, delay, engage resource and release resource (Birtwistle, 1979). The graph shows the flow of entities as well as resources through the system. Network (or block) diagrams are used by many popular commercial simulation packages (e.g., General Purpose Simulation System (GPSS) (Schriber, 1974), Simulation Language for Alternative Modeling (SLAM) (Pritsker, 1979) and Simulation ANalysis (SIMAN) (Pegden, 1990)). These network diagrams are similar to activity diagrams, but have more types of nodes corresponding to the underlying primitives supported in their associated simulation languages.

Besides these three classical world-views, there are other popular approaches to discrete-event modeling: state transition models, for example, focus on the states of the system and transitions between them (e.g., Markov Chains).

Most of the simulation taxonomies are based on execution characteristics of models – scheduling events, scanning for activities, interacting processes. The simulation taxonomies, in turn, determine the taxonomies of the models themselves. Fishwick (Fishwick, 1996b) suggests a different approach based on model “syntax”. In particular, he creates a model hierarchy that conforms closely to the taxonomy of programming languages: declarative, functional, and constraint. By closely coupling simulation models with programming paradigms, this taxonomic exposition highlights the connections between models and programs. For example, declarative finite state automata and Petri nets bear similar features to declarative programming languages based on rules and logic. Functional methods of simulation modeling are related to functional languages formalized with lambda calculus, such as Lisp. Other modeling categories to consider are conceptual and spatial models (Fishwick, 1995).

Another useful way to build a taxonomy is based on subsumption relationships between the formalisms used. In other words, one can define top-level modeling formalisms that subsume (in some sense) lower formalisms (e.g., General Semi-Markov Processes (GMSP) subsume Semi-Markov Processes (SMP), which subsume Markov Chains (MC), etc.).

As mentioned before, an ontology may be more than just a taxonomy with a simple hierarchic structure, but can capture more complex knowledge. It may indeed implicitly capture multiple taxonomies. A taxonomy, however, is a good mental starting point for building an ontology. Defining a backbone taxonomy and “growing” the related concepts on top of it is the basic approach used to develop the DeMO ontology.

4 Discrete-Event Modeling Ontology

In this section, we will highlight and discuss some of the details of the Discrete-event Modeling Ontology (DeMO). DeMO is a Web-accessible ontology for discrete-event modeling (DEM). It was envisioned to be a prototype designed to investigate the principles of construction of an ontology for discrete-event modeling and to flush out the problems and promises of this approach, as well as, directions of future research.

It is hoped that a future fully developed DeMO ontology (or other such an ontology/ies) will be useful to researchers and practitioners of modeling and simulation. Applications include locating modeling and simulation software, particular modeling applications and modeling components as well as facilitating meta-modeling and multi-modeling. It can also become a focal target of standardization efforts in this area.

As of this moment, we do not attempt to cover all existing discrete-event modeling formalisms in DeMO, rather the goal is to establish the main building principles that will allow them to be included in future ontologies. We illustrate this using many of the established discrete-event modeling techniques.

When building this ontology, we try to capture as much knowledge about the DE modeling domain as seems plausible. That means identifying all relevant concepts and providing (if possible) machine-interpretable formal specifications for these concepts.

Let us look closer at the DeMO ontology. It has four main abstract classes representing main concepts in this knowledge domain: *DeModel*, *ModelConcepts*, *ModelComponents* and *ModelMechanisms*. The rationale here was the following: any DeModel is built from model components and is “put in motion” by model mechanisms. As shown in figure 1, ModelConcept represents the most basic concepts used in DeMO from which the

other concepts are built.

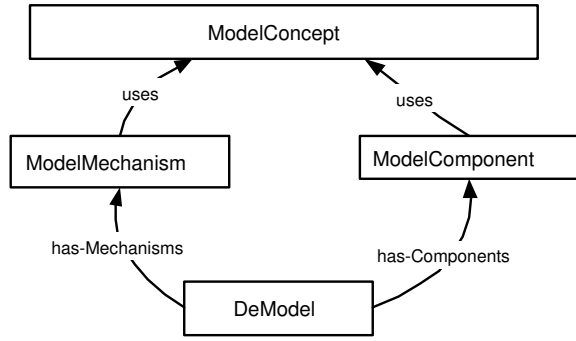


Figure 1: Graphical Representation of the rationale behind DeMO design

4.1 DeModel Class and the Backbone Taxonomy

The DeModel class represents a root of our backbone taxonomy which, as it was mentioned earlier, is our mental starting point for building an ontology. This taxonomy of models is defined based on (i) world-view of these models and (ii) their theoretical modeling power (subsumption relationships).

Following closely the classical implementation-based world-views, the *first-level classification* splits the root of the taxonomy in four different types of discrete event models based on their structural characterization (i.e., the formal components used to define the model structure): *State-Oriented*, *Event-Oriented*, *Activity-Oriented*, and *Process-Oriented Models*.¹

Each of these first-level formalisms defines a set of components (concepts) used in building lower subclasses. Their subclasses are defined by placing restrictions on properties of the higher formalisms. This *second-level classification* illustrates a subsumption approach in building a taxonomy.

The top subclass of State-Oriented Model is Generalized Semi-Markov Process (GSMP). All other models in this subtree are derived from it by placing restrictions on its components (e.g., Timed Automata, as defined in (Cassandras and Lafortune, 1999), have deterministic ClockFunction; Discrete-Time Markov Chains have discrete TimeSet together with Poisson ClockFunction, etc.).

We chose the Extended Stochastic Petri Net (ESPN) to be the top subclass of Activity-Oriented Mod-

¹An alternative would be to use a completely unifying framework/formalism such as Discrete-Event Simulation (DEVS) (Zeigler, 1976). For simplicity and to maintain the natural flavor of existing models, we chose at this time not to aim for complete unification.

els. Various restrictions on these components including the restrictions on the topology of the underlying Place-Transition Nets allow us to derive many popular activity-oriented modeling formalisms (e.g., Generalized Stochastic Petri Nets have Poisson ClockFunction; Free Choice Nets and Simple Nets are Petri Nets with specific restrictions on graph topology, etc.).

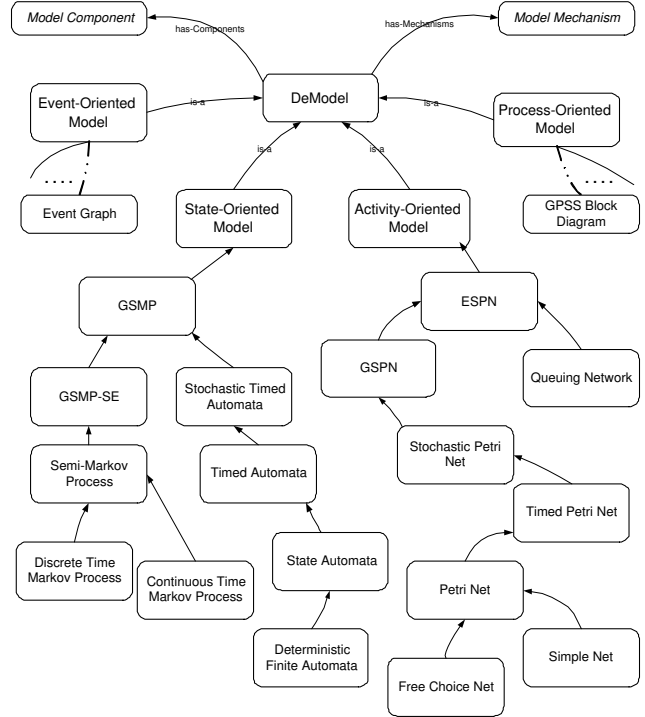


Figure 2: Portion of the backbone taxonomy

The result is a hierarchical structure (partially presented in figure 2) where classes are related through *is-a-subclass-of* (parent/child) relationships. This kind of taxonomy with corresponding links on the Web is easily browsed by humans and may serve as a good reference. However, we can go further with ontologies. We can put more complex and meaningful machine-understandable knowledge in the ontology by using stronger semantic connections in the DES models. Using formal definitions traditionally used in the literature for different DE modeling formalisms we identify the key concepts of the domain and categorize them.

4.2 ModelComponent and ModelConcept Classes

The ModelComponent class describes the model elements used as the building blocks of DeModel (each model is linked through *has-a* relationships with its component classes). These generally correspond to the elements of *n-tuples* traditionally used in the literature

to formally define the models. It is convenient to split this class into four subclasses in correspondence with the first-level classification.

The State-Oriented and Event-Oriented models need the following formal components to be defined: *StateSpace*, *EventSet*, *TimeSet*, *TransitionFunction*, *ClockFunction* and *InitialState*. Analogously, Activity-Oriented Models need the following formal components *PlaceSet*, *ActivitySet*, *TimeSet*, *IncidenceFunction*, *InitialMarking* and *ClockFunction*. Here *ClockFunction* has a slightly different meaning than in State-Oriented modeling formalism. Process-Oriented models have *ProcessSet*, *ProcessStateSet*, *TimeSet*, *TransitionFunction*, and *ClockFunction* as formal components.

Many of the *ModelComponents* characterizing different first-level formalisms are either identical in meaning or translatable into each other. These relationships may be captured using description logic tools provided by OWL, thus placing stronger semantic connections between the model components.

To capture even more knowledge, *ModelComponents* themselves are defined in terms of other concepts (e.g., *StateSpace* is defined as a set-of *State*). The most basic, fundamental concepts are placed in a *ModelConcept* class. This class has the following subclasses: *State*, *Event*, *Time*, *Transition*, *Activity*, *Place*, *Token*, *Entity*, *Process*, *Resource*, *Color*, and *Clock*. Representing fundamental concepts, some of these classes may not be formally defined in terms of other concepts (classes) and often only informal descriptions in English are provided. In a sense, they serve as a set of basic terms upon which the ontology is built, and in that respect are similar to such geometric terms as point, line, between, and congruent.

To provide greater modularity, the *ModelConcept* class together with its subclasses may be placed in a separate meta-ontology which will serve as some type of a discrete-events glossary used by DeMO or other ontologies.

4.3 ModelMechanism Class

The class *ModelMechanism* contains subclasses that specify different mechanisms (rules of engagement) adopted by particular modeling techniques: *EventSchedulingMechanism*, *TransitionTriggeringMechanism*, *ClockSettingMechanism*, etc.

In essence, the *ModelMechanisms* “explain how to run the model”, at least abstractly. Event scheduling procedures, for example, are quite different for different models, and they are captured in the *EventSchedulingMechanism* Class. The same is true with transition firing. Consider, for example, Atomic vs. Non-Atomic Firing semantics in Petri Nets, which are han-

dled by *TransitionTriggeringMechanism*. The *ClockSettingMechanism* describes how the event clocks that determine a *ClockFunction* are set and itself has two subclasses, *StochasticClockSetting* and *DeterministicClockSetting*.

The *ModelMechanisms*, at this point, are treated as basic concepts in that they are not defined in terms of other concepts, and only a description in English is provided. More knowledge concepts can certainly be defined related to *ModelMechanism* possibly giving rise to a new ontology based on a few “axioms”. Until then each model class in DeMO has to use a specific instance (rather than a class) of each relevant *ModelMechanism* in its definition. Note that *ModelComponents* are treated differently – a specific class (not an instance) of each *ModelComponent* is used in the definition of each concrete *DeModel*.

The specific division between *ModelMechanisms* and *ModelComponents* is somewhat arbitrary – some models, in principle, may be reformulated in such a way that a model mechanism can become a component and vice-versa.

4.4 Models in DeMO

As already mentioned, we did not attempt at this point to include all known DE modeling formalisms in the DeMO ontology. However, one of the goals was to design DeMO in such a way that to allow relatively simple “update” procedures.

To insert a new model (modeling formalism) in DeMO, one has to determine the first-level class which this formalism belongs to (State-Oriented, Event-Oriented, Activity-Oriented, or Process-Oriented Model). Then the model is defined by placing restrictions on the properties of the top formalism and specifying the instances of *ModelMechanisms* used.

Figure 3 depicts the GSMP model class, which is the highest concrete class in the State-Oriented model formalism. It is defined by placing the following restrictions on the properties of the *StateOrientedModel* presented above: *ClockFunction* is specified/restricted to be *StochasticClockFunction*, *TransitionFunction* is specified/restricted to be *ProbabilisticTransitionFunction*. In addition, it uses the following instances of subclasses of *ModelMechanism*: *TransitionTriggering* has value *_MultipleEventTriggering mechanism*, *TransitionEnabling* has value *_EventEnabling mechanism*, and *EventScheduling* has value *_StateMachineScheduling mechanism* (note, an underscore in front is used to denote instances of classes). The *_MultipleEventTriggering mechanism*, for example, is an instance of *TransitionTriggering* that allows simultaneous event occurrence to trigger a

transition. The `_EventEnabling` mechanism means that the transitions are enabled together with events, and the `_StateMachineScheduling` mechanism explicates how the set of imminent events is determined. Note that unrestricted properties are inherited from a parent class (or classes).²

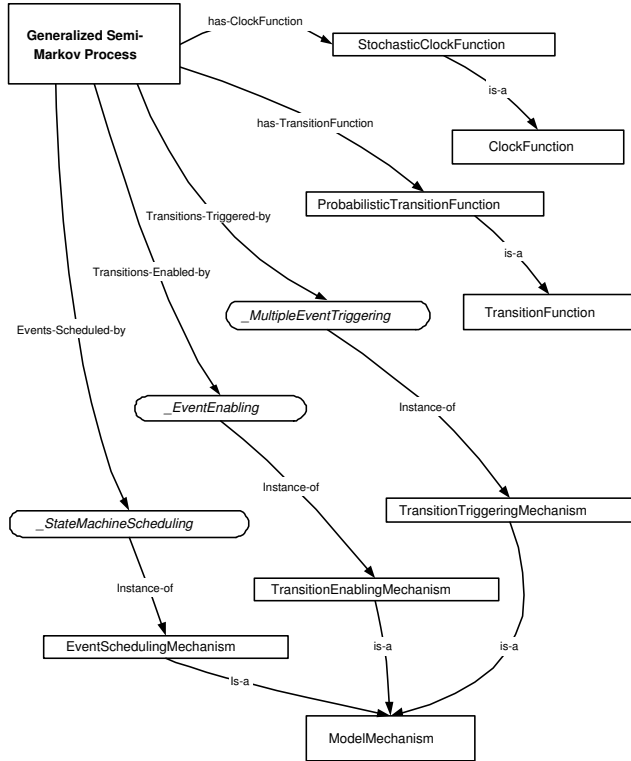


Figure 3: Graphical Representation of Generalized Semi-Markov Process model class.

An issue concerning multiple ways of defining a class may arise. A class, `SemiMarkovProcess` for example, can be produced from `GSMP` by changing the method of scheduling events. In `DeMO` that may correspond to forcing the `EventScheduling` mechanism to have an appropriate instance, say `_MarkovianScheduling` mechanism. On the other hand, effectively the same model will be produced by putting restrictions on the cardinality of the `EventSet` – setting it equal to one. Ideally, both ways of defining the class need to be captured in the ontology. However, it is not clear how this should be done, and for now we pick one based on simplicity and intuitive appeal.

At this point, `DeMO` is not “populated”, i.e., instance data has largely been ignored. However, in the future the ontology will be populated with descriptions of ar-

²The complete ontology is too large to be shown in this paper, but is available at chief.cs.uga.edu/~jam/jsim/DeMO.

tifacts available on the Web (models, components, libraries, papers, etc.).

4.5 Potential Benefits

We envisage several possible benefits or usages for `DeMO` (or other such ontologies).

Browsing. One could look at the concepts in the ontology and navigate to related concepts. This can be done with `Protégé` by expanding/contracting the class hierarchy tree. `EzOWL` provides a GUI-oriented visualization of the same hierarchy. Finally, there is some initial work on making `OWL` tools interoperate `UML` tools.

Querying. Query languages are being developed to provide powerful querying capabilities for one or more ontologies. There are many such languages under development (e.g., `RQL`, `DQL`, `OWL-QL`) and more work is required as the `Semantic Web` stack gets fleshed out. Current work will soon establish the next layer, the logic layer (e.g., `SWRL` and `FORUM`). Given such query capabilities, queries on `DeMO` would be very useful.

Service Discovery. One could look for a `Web` service to perform a certain modeling task. For the use of ontologies in `Web` service discovery, please see (Verma et al., 2003). For using `Web` services in simulation modeling, see (Chandrasekaran et al., 2002).

Components. By providing an ontology, we create a `Web`-based infrastructure for storing and retrieving executable simulation model components. These components can facilitate reuse. For example, consider `ProbabilisticTransitionFunction` in Figure 3. Code implementations of specific probability density functions can be attached directly to this link, and they are made available to those searching for them.

Hypothesis Testing. The `LSDIS` Lab is currently carrying out funded research to allow hypothesis testing to be carried out using the `Semantic Web` (Sheth et al., 2003). In the future, this capability could be used to pose challenging questions such as which adaptive routing algorithm will work best on the evolving `Internet`.

Research Support. Papers in the field of modeling and simulation may be linked into the ontology to help researchers find more relevant research papers more rapidly. These links can be added manually or through automatic extraction/classifications tools such as those provided by `Semagix` (www.semagix.com).

Mark-up Language Anchor. It is popular today to develop domain-specific `XML`-based mark-up languages. This allows interfaces to software or descriptions of software to be presented in platform and machine-independent ways. In the simulation community such work has begun (e.g., `XML` for *rube* (Fishwick 2002b)). This enhances the interoperability of simulation mod-

els. The tags used in the markup language should ideally be anchored in a domain ontology.

Facilitate Collaboration. By establishing a shared conceptual framework, opportunities for increased collaboration between researchers are increased. This includes interoperability of simulation tools, model reuse and data sharing.

5 Conclusions and Future Work

This paper represents an early attempt to create an ontology for modeling and simulation. Such an ontology could contribute to increasing collaborative work in these areas as well as enhance the reusability of artifacts. DeMO is intended to be a starting point for an eventual well-accepted ontology for Modeling and Simulation. One way for this to happen quickly is to start with a core set of papers on the subject and then form groups to study the issue and come to an agreement on an ontology. Experience of groups such as the one that created the Gene Ontology (GO) should be considered (The Gene Ontology Consortium, 2000) in this endeavor.

Important issues to consider for future development include:

- Use of an Upper Ontology. To create an ontology with a high-level of precision requires the definition of many base concepts. One way to do this is to use an upper ontology such as SUMO or SUO. Since SUMO is still being refined, we decided to do this at a later time, so at this point our ontology defines mathematical concepts such as set and function. When SUMO (or SUO) becomes a stable, useful and well-accepted upper ontology, it should probably be used.
- Compatibility with Other Ontologies (e.g., a future Math ontology, or an ontology for Graph Topology).
- Breadth of the Ontology. If the domain ontology is too broad it may become too complex and disjointed. Ambiguities may be quite difficult to resolve. On the other hand, if it is too narrow, it is of limited use.
- Further categorization in knowledge subdomains is required (e.g., for ModelMechanisms), as well as the representation of deeper relationships between the concepts (such as mappings between different models, for example).
- Handling of Multiple Taxonomies. What is the best way to embed multiple taxonomies in the ontology? Should a principal taxonomy be picked as

the backbone (subsumption of modeling techniques was chosen in DeMO). The other taxonomies then became secondary (e.g., determinacy, application area, etc.).

- Link to (or incorporate) Discrete-Event Simulation (DEVS) models which could be viewed as a taxonomy or ontology of their own right (Sarjoughian et al., 2001).

References

- (Berners-Lee et al., 2001) T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web," *Scientific American*, Vol. 284, No. 5, pp. 34-43.
- (Birtwistle, 1979) G.M. Birtwistle, *Discrete Event Modeling in SIMULA*. MacMillan, London, England.
- (Cassandras and Lafortune, 1999) C.G. Cassandras and S. Lafortune, S., *Introduction to Discrete Event Systems*, Kluwer.
- (Chandrasekaran et al., 2002) S. Chandrasekaran, G. Silver, J.A. Miller, J. Cardoso and A.P. Sheth, "Web Service Technologies and their Synergy with Simulation," *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 606-615.
- (Denny, 2002) M. Denny, "Ontology Building: A Survey of Editing Tools," www.xml.com/pub/a/2002/11/06/ontologies.html.
- (Fishman, 1973) G.S. Fishman, *Concepts and Methods in Discrete Event Digital Simulation*, Wiley, New York, NY.
- (Fishwick, 1995) P.A. Fishwick, *Simulation Model Design and Execution: Building Digital Worlds*, Prentice-Hall, Inc., Englewood Cliffs, NJ.
- (Fishwick, 1996) P.A. Fishwick, "Web-Based Simulation: Some Personal Observations," *Proceedings of the 1996 Winter Simulation Conference*, San Diego, CA, pp. 772-779.
- (Fishwick, 1996b) P.A. Fishwick, "A Taxonomy for Simulation Modeling Based on Programming Language Principles," *IIE Transactions on IE Research*, Vol. 30, pp. 811-820.
- (Fishwick, 2002) P.A. Fishwick, "Using XML for Simulation Modeling," *Proceedings of the 2002 Winter Simulation Conference* San Diego, CA, pp. 616-622.
- (Fishwick, 2002b) "The rube Framework for Personalized 3-D Software Visualization," *Software Visualization*, Springer Verlag.
- (The Gene Ontology Consortium, 2000) The Gene Ontology Consortium, "Gene Ontology: Tool for the Unification of Biology," *Natural Genetics*, Vol. 25 pp. 25-29.
- (Gruber, 1995) T.R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing,"

- Int. Journal of Human-Computer Studies*, Vol. 43, pp. 907-928.
- (INCOSE, 2002) INCOSE, "SE Tools Taxonomy - Simulation Tools," www.incose.org/tools/tooltax/simulation_tools.html.
- (Kim et al., 2002) T. Kim, J. Lee, and P.A. Fishwick. "A Two-Stage Modeling and Simulation Process for Web-Based Modeling and Simulation", *ACM Transactions on Modeling and Computer Simulation*, Vol. 12 no. 3, pp. 230-248.
- (Lacy, 2001) L. Lacy, "Semantic Web Applications for Modeling and Simulation," www.daml.org/2001/07/dmsso-applications/semantic-web-071101.ppt.
- (McGuinness and van Harmelen, 2003) D.L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," www.w3.org/TR/owl-features.
- (Miller et al., 1997) J.A. Miller, A.P. Sheth and K.J. Kochut, "Perspectives in Modeling: Simulation, Database and Workflow," *Proceedings of the 16th International Conference on Conceptual Modeling: Preconference Symposium*, Los Angeles, CA.
- (Miller et al., 2001) J.A. Miller, P.A. Fishwick, S.J.E. Taylor, P. Benjamin and B. Szymanski, "Research and Commercial Opportunities in Web-Based Simulation," *Simulation Practice and Theory*, Special Issue on Web-Based Simulation, Vol. 9, No. 1-2, October 2001, pp. 55-72
- (Nair et al., 1996) R. Nair, J.A. Miller and Z. Zhang, "A Java-Based Query Driven Simulation Environment," *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, December 1996, pp. 786-793.
- (Nance, 1993) R.E. Nance, "A History of Discrete Event Simulation Programming Languages," *Proceedings of the Second ACM SIGPLAN History of Programming Languages Conference*, Cambridge, MA, Reprinted in ACM SIGPLAN Notices, Vol. 28, No. 3, pp. 149-175.
- (Page, 1994) E.H. Page, Jr., *Simulation Modeling Methodology: Principles and Etiology of Decision Support* Ph.D. Dissertation, Virginia Tech. www.thesimguy.com/ernie/papers/unref/dissert/dw.html
- (Page et al., 2000) E.H. Page, Jr., P.A. Fishwick, K.J. Healy, R.E. Nance, and R.J. Paul, "Web-Based Simulations: Revolution or Evolution," *ACM Transactions on Modeling and Computer Simulation*, Vol. 10, No. 1, pp. 3-17.
- (Pegden et al., 1990) C.D. Pegden, R.E. Shannon and R.P. Sadowski, *Introduction to Simulation Using SIMAN* McGraw-Hill, NY.
- (Peterson, 1977) J.L. Peterson, "Petri Nets," *ACM Computing Surveys*, Vol. 9, pp. 223-252.
- (Petri, 1962) C.A. Petri, "Fundamentals of a Theory of Asynchronous Information Flow," *Proceedings of IFIP Congress 62*, pp. 386-390.
- (Pidd, 1992) M. Pidd, *Computer Simulation in Management Science*, Wiley, Chichester, England, 3rd edition.
- (Pritsker, 1979) A.A.B. Pritsker, *Introduction to Simulation with SLAM*, Wiley, New York, NY.
- (Sarjoughian et al., 2001) H.S. Sarjoughian, F.E. Cellier and B.P. Zeigler (Editors), "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, Berlin-Heidelberg.
- (Schriber, 1974) T.J. Schriber, *Simulation Using GPSS*, Wiley, NY.
- (Schruben, 1983) L. Schruben, "Simulation Modeling with Event Graphs," *Communications of the ACM*, Vol. 26, pp. 957-963.
- (Schruben and Roeder, 2003) L. Schruben and T. Roeder, "Fast Simulations of Large-Scale Highly-Congested Systems," *Simulation: Transactions of the Society for Modeling and Simulation International*.
- (Sheth et al., 2003) A.P. Sheth, S. Thacker and S. Patel, "Complex Relationship and Knowledge Discovery Support in the InfoQuilt System," *VLDB Journal*, Vol. 12, No. 1, pp. 2-27.
- (Verma et al., 2003) K. Verma, K. Sivashanmugam, A.P. Sheth, A. Patil, S. Oundhakar and J.A. Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," To appear: *Information Technology and Management*.
- (Zeigler, 1976) B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley and Sons, New York, NY.