

Ontologies for Modeling and Simulation: An Initial Framework

JOHN A. MILLER , GREGORY T. BARAMIDZE , AMIT P. SHETH and GREGORY A. SILVER

University of Georgia

and

PAUL A. FISHWICK

University of Florida

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 Modeling and Simulation community. Ontologies provide a way to establish common vocabularies and capture domain knowledge for organizing the domain with a community-wide agreement. They can be used to provide significantly improved (semantic) search and browsing, integration of heterogeneous information sources, and improved analytics and knowledge discovery capabilities. In this paper, the design and development of a draft ontology for Modeling and Simulation called the Discrete-event Modeling Ontology (DeMO) are discussed, which can form a basis for achieving a broader community agreement and adoption. Relevant taxonomies and formal frameworks are reviewed and the design choices for the DeMO ontology are made explicit. Prototype applications that demonstrate various uses and benefits from such ontologies for the Modeling and Simulation community are also presented.

Categories and Subject Descriptors: I.6.1 [**Simulation and Modeling**]: Simulation Theory—*Model classification*; I.6.8 [**Types of Simulation**]: Discrete event; I.6.5 [**Model Development**]: Modeling Methodologies; I.6.7 [**Simulation Support Systems**]: Environments; I.6.m [**Miscellaneous**]: Discrete-event Modeling Ontology

General Terms: Theory, Standardization, Languages, Design

Additional Key Words and Phrases: Discrete-event modeling, ontology design, knowledge domains, ontology-driven modeling

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 repre-

Author's address: J.A. Miller, Computer Science Department, University of Georgia, Athens, GA 30602, U.S.A.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

senting domain semantics can be grouped and interrelated. Modeling and Simulation (M&S) 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 different types of models relate? 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 connections 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 Modeling and Simulation 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 1996b; 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 2002b; 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. XML is meant for documents and data exchange. The structure of a group of related XML documents is specified using a schema language, such as XML Schema, which can provide some level of standardization in semantics.

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], promise 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 XML-based Web languages and their supporting tools (see www.w3.org). They should be sharable, independent of particular resources (e.g., documents or applications), understandable by humans, processable in meaningful ways by computers, and as descriptive as possible for a domain. Essentially, they should describe the way things are for a particular domain. Meeting all these goals takes one beyond database-like schemas into the field of knowledge representation.

Much of this work involves the use of ontologies [Gruber 1995] that 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 expressed 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 2004]. 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).¹

A new wave of technology associated with the Semantic Web is emerging that could have an even greater impact than Web-based simulation [Miller et al. 2004; Fishwick and Miller 2004]. For the modeling and simulation community, one of the most important contributions of the Semantic Web is the creation of ontologies for specific domains. These can be viewed as Web-based knowledge repositories of explicit concepts pertaining to specific domains and relationships between them.

This paper develops an example of an ontology for Discrete-Event Modeling domain and sketches the steps involved in creating such ontologies. In particular, we worked to identify the concepts that are most relevant for the Discrete-Event Modeling domain, the relationships between them, the overall architecture of, and some of the technical steps involved in creating, deploying and using such an ontology.

The paper also illustrates different ways in which such an ontology may be used by the Modeling and Simulation community. It also argues that the creation of ontologies for Modeling and Simulation domain is a useful step in further development of the field and it will open new perspectives in the way Modeling and Simulation is being viewed. All steps of simulation model development cycle, as well as, such long-standing goals of M&S community as model reuse, composition, modularity can benefit from such ontologies. We also argue that combining the modeling and simulation ontologies with different domain ontologies emerging in all branches of science and technology will facilitate streamlining of model development for specific scientific domains, interdisciplinary knowledge exchange, creation of model repositories, etc.

The rest of this paper is structured as follows. In Section 2, we consider languages suitable for defining ontologies meant for the Semantic Web. We then briefly review prior taxonomies for modeling and simulation in Section 3.1. Section 3.2 discusses

¹There 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 (SUO) at suo.ieee.org, which can arguably facilitate the use and integration of multiple ontologies.

existing discrete-event modeling frameworks and builds a backbone taxonomy for our proposed ontology expressed semi-formally in a combination of mathematics and natural language. Section 4 discusses the steps in the development of ontologies for discrete-event modeling. We present strategies and issues that came up as we developed a prototype ontology called DeMO [Miller et al. 2004], 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. Section 5 discusses applications and benefits of the discrete-event modeling ontology. The paper is wrapped up with conclusions and future work in Section 6.

2. ONTOLOGIES FOR THE SEMANTIC WEB

Although ontology languages such the Knowledge Interchange Format were developed in the 1990's, the Semantic Web initiative has reinvigorated efforts to create a new ontology language which can have a wider appeal. The Knowledge Interchange Format has the expressive power of First Order Logic (FOL) which enables complex concepts to be precisely described. The truth value of any FOL expression can be effectively proved (FOL is sound), but it is not possible to generate all true statements (FOL is incomplete) – FOL is said to be semi-decidable [Gödel 1930; 1931]. For the Semantic Web, some researchers feel that a decidable language would be more appropriate, since limiting the language expressivity will improve its computability/tractability. The Web Ontology Language (OWL) was designed with this expressivity/complexity trade-off in mind and comes in three flavors: OWL Lite, OWL DL (Description Logic) and OWL Full, with the first two being decidable. Still some simple expressions such $x < y$ or $z = x + 1$ as well as recursive definitions are beyond the capabilities of OWL. To fill the gap, the Semantic Web Rule Language (SWRL) (www.w3.org/Submission/SWRL/) is being proposed which essentially combines OWL DL with a subset of RuleML (Horn-like rules)(www.ruleml.org). The current proposal will lead to a semi-decidable language, although there is active research ongoing to find decidable subsets that are useful [Bechhofer et al. 2004]. From our experience, complex ontologies suitable for modeling and simulation will need the capabilities of SWRL or at least a significant subset thereof. (In fact, a whole family of languages may be needed to represent a “spectra” of related ontologies that will vary in their expressiveness and usage.)

2.1 Structure of the Semantic Web

Before going into more detail about ontologies, we will give a brief overview of the architecture of the emerging Semantic Web. This will provide a better perspective on the work presented in this paper.

At the XML 2000 Conference, Berners-Lee presented his plan for the Semantic Web Architecture [Berners-Lee 2000]. The XML language forms the base of the architecture and is proposed to be the basic syntax for documenting structured information. Next in the architecture, the Resource Description Framework (RDF) and RDF Schema languages add semantics to the data model with machine predictable tags. The ontology languages represented by OWL are more expressive than RDF in that they are able to express more class relationships and property constraints. This is the layer that serves as the basis for later logical inference.

Based on the information from the ontology layer, the logic languages may use rules to make deductions about the facts and relations not explicitly stated in the ontology. In the proof layer, automatic agents will be able to send and accept proofs necessary for interchange. With proofs, trust can be built up among a community of Web users [Berners-Lee et al. 2001].

Each layer of the Semantic Web architecture (or stack) has an associated Web Language, as indicated in the Table I. In the subsections below, we overview two of the first four layers/languages – ontology layer and rule/logic layer – since they are directly related to the topic of the paper. The other two layers are discussed in the Appendix A. The upper layers are still very much future work, so we do not discuss them further.

Table I. Layers of the Semantic Web architecture.

Layer	Principal Language	Name (URL)
Resource/Data	XML, XSD	Extensible Markup Language, XML Schema Definition (www.w3.org/XML , www.w3.org/XML/Schema)
Meta-Data	RDF, RDF Schema	Resource Description Framework, RDF Schema (www.w3.org/RDF)
Ontology	OWL (Lite, DL, Full)	Web Ontology Language (www.w3.org/2004/OWL/)
Rule/Logic	SWRL	Semantic Web Rule Language (www.w3.org/Submission/SWRL/)
Proof/Trust	<i>Future Work</i>	

2.2 Ontology Layer - Web Ontology Language (OWL)

The Web Ontology Language 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 extends upon RDF and RDF Schema. Compared to RDF Schema, OWL has more features 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).²

OWL comes in three flavors: OWL Lite which has expressiveness similar to RDF Schema, OWL DL based on description logics which are computationally complete and decidable, and OWL Full which provides additional features and sacrifices decidability [McGuinness and van Harmelen 2003].

Our ontology is represented in OWL DL and was developed using Protégé 3.1 (protege.stanford.edu) with the OWL plug-in. Protégé is currently one of the most popular ontology editing tools (see [Denny 2004] for a survey). OWL DL allows the definition of a class to represent entities of a certain kind. Classes may

²As a revision of the DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) ontology language, OWL is built upon the lessons learned from the design and application of DAML+OIL [McGuinness and van Harmelen 2003].

(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 and the Unified Modeling Language (UML). Entity-Relationship Modeling is intended for data modeling, UML is intended for software modeling, and OWL is intended for ontology modeling and knowledge representation that is consistent with the Web infrastructure. Compared to UML Class Diagrams, OWL DL permits the creation of subproperties as well as more ways of restricting properties/relationships.

2.3 Rule/Logic Layer - Semantic Web Rule Language (SWRL)

There are some recent proposals for languages at the Rule/Logic Layer including the Semantic Web Rule Language (SWRL), which extends OWL DL with the ability to write rules using a subset of RuleML. It also permits Horn logic rules to be added to OWL descriptions. This allows more complex predicates to be created and used for more precise definitions of concepts. It also has built-ins for basic arithmetic (e.g., $x + y$) and comparisons (e.g., $x < y$). Since the rule/logic layer is in its preliminary stages of development, we plan to enhance DeMO using this layer in our future work. An alternative to SWRL is RuleML which has a longer maturation horizon.

3. TAXONOMIES FOR DISCRETE EVENT MODELING

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]. Surveying existing taxonomies is a useful way to begin the development of an ontology, hence we first give a brief review of existing simulation modeling taxonomies and then show how the existing discrete-event modeling formalisms can be grouped together to serve as a backbone taxonomy for DeMO.

3.1 A review of existing taxonomies

Given a system evolving over time, a model of it can be viewed simply as an approximation that captures or mimics the essential features of the system. Commonly, models are classified according to how they deal with time (*Static vs. Dynamic*), state (*Discrete vs. Continuous*) and randomness (*Deterministic vs. Stochastic*).

There are two broad types of simulation modeling: continuous simulation and discrete-event simulation. The distinction is based on whether the state can change continuously (water level in a reservoir) or at discrete points in time (number of customers in a bank). Continuous changes are often modeled as differential equations in which both state and time are continuous. Discrete-event simulation models are very popular for modeling many types of real-world systems such as banks, hospitals, transportation systems, and computer networks so we will focus our attention on them.

Within discrete-event modeling there are multiple ways to construct a taxonomy. The development/use of simulation programming languages led to the creation of taxonomies for discrete-event simulation. These languages allowed developers to define models using categories identified with a particular view of reality. Developers

used the views implicit with the simulation languages when contemplating the system [Lackner 1962]. Kiviat's [1969] discussion of simulation programming languages lead to the basic taxonomy for discrete-event simulation which includes the three world-views: event-scheduling, activity-scanning and process-interaction. This type of 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 discussed below. Most of the diagrammatic simulation modeling techniques implicitly or explicitly depict entities (e.g., bank customers) flowing through a system. The classification/taxonomy below mentions several popular general simulation modeling techniques partitioned according to their principal world view.

Event-Scheduling (ES). These models focus 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 in a future event-list. Time is advanced by jumping to the time-stamp of the earliest event (imminent event) on the list and processing that event.

—*Event Graphs (EG).* In an event graph, nodes (or vertices) represent events, while directed edges represent causality [Schruben 1983]. Edges can be annotated with time delays and conditions.

Activity-Scanning (AS). These 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).* Activity Cycle Diagrams 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 [Tocher 1963; Pidd 1992]. These diagrams depict the life-cycles of interacting entities flowing through a system.

—*Petri Nets (PN).* Petri Nets 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 place is a storage area for tokens (entities), while a transition takes input token(s) to produce output token(s). A transition will fire if there is a (or are enough) tokens at each of its input places. In Timed Petri Nets, transitions have delays associated with them. In Colored Petri Nets, tokens can have attributes. Both Activity Cycle Diagrams and Petri Nets models can be used to generate simulations following either the activity-scanning or process-interaction world views.

Process-Interaction (PI). These models focus on processes and their interaction with resources. A process captures the behavior of an entity as it flows through the system, step by step.

—*Activity Diagrams (AD).* 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 Diagrams (ND)*. 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 et al. 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: (i) State Transition Models (e.g., Markov models), (ii) Discrete Event System Specification (DEVS) models [Zeigler 1976; Zeigler et al. 2000] and (iii) Object-Orientated Simulation (see Appendix D).

Most of the simulation taxonomies are based on execution characteristics of models – scheduling events, scanning for activities, interacting processes. Fishwick [1995; 1996a] suggests a different approach based on model “syntax.” In particular, he creates a taxonomy 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.

Another useful way to build a taxonomy is based on subsumption relationships between the formalisms used (e.g., General Semi-Markov Processes (GSMPs) subsume Semi-Markov Processes (SMP), which subsume Markov Chains (MCs)). Formal structures can also be compared using homomorphisms to show one modeling technique can do anything another technique can, or isomorphisms to show two techniques are equivalent. This is the basic approach used to develop a backbone taxonomy for the DeMO ontology.

Discrete event simulation models can be defined as abstract, dynamic, descriptive, numerical models [Page 1994]. Cassandras and Lafortune [1999] define discrete-event systems as discrete-state, event-driven, time-invariant, dynamic models. Within the class of discrete event models, further subclassification may be achieved by the means of the following characteristics: stable vs. unstable models, steady-state vs. transient models, deterministic 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 as opposed to the classical three worldviews. An ontology is more than a taxonomy, and indeed may implicitly capture multiple taxonomies.

In Section 4 we will show and discuss some of the details of the Discrete-event

Modeling Ontology (DeMO). DeMO is an ontology for discrete-event modeling (system dynamics for discrete-event systems). In their essence, the models in the ontology capture how state evolves over time, although they may focus on other concepts such as entity, event or place. The state space may be either discrete (finite or countably infinite) or continuous, while time may be continuous (uncountable), although the number of state changes (via events) must be discrete. We consider both stochastic models and deterministic models. It is hoped that the DeMO ontology will be useful to researchers and practitioners of modeling and simulation. Such an ontology could find application in locating modeling and simulation software, particular modeling applications and modeling components. It also may help in integrating different aspects of simulation and modeling in a robust and flexible way, interdisciplinary model exchange, as well as facilitating meta-modeling and multi-modeling [Fishwick and Zeigler 1992; Vangheluwe et al. 2002].

At present our ontology development has focused on the classical world views. Newer modeling paradigms such as object oriented and agent based while important are more focused on large scale model development which includes issues of hierarchical modeling and model reuse. After developing the core ontology it would be natural to pursue these in our future work.

3.2 Grouping Discrete-Event Modeling Formalisms — a Backbone Taxonomy for the Ontology

As a first step in the development of the DeMO ontology, we start by composing a taxonomy of discrete-event modeling formalisms. We do this by combining existing formalisms into a backbone taxonomy on which the rest of the ontology is built. This is done so that we can collect the existing knowledge in an organized fashion. Although slight modifications to formalisms were made, our purpose was not to create a new unifying framework, but rather to create a useful stepping stone for producing the ontology. A good review and evaluation of existing formal approaches in the field of discrete-event simulation is given in [Page 1994].

An alternative to creating a taxonomy for several existing formalisms is to create a unified general framework such as DEVS [Zeigler 1976; Zeigler et al. 2000]. Classical DEVS is given by a seven tuple that includes three sets (input, output, state) and four functions (time advance, internal transition, external transition, output). It also implicitly includes a set for time. Zeigler shows how to translate event-scheduling and activity-scanning models into classical DEVS. Process interaction is also formalized with an extended structured model. These well defined formalisms could be used for ontological development (see [Silver et al. 2006] for an example related to process interaction).

We use a series of top formalisms that differ from each other in their structural principles motivated by their world-views and modeling power. The assumption is that each of the top formalisms is general enough to serve as a formal basis for discrete-event systems modeling. Other modeling formalisms are added by placing restrictions on higher formalisms (e.g., timed automata is a stochastic timed automata with a deterministic clock structure, etc.).

The result is a hierarchical structure where objects are related through *is-a* or *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, more complex and meaningful knowledge can be captured in an ontology by using stronger semantic connections in the discrete-event simulation models. Links to other Web resources can be established as well.

To help develop the conceptual framework, rather than getting bogged down in details related to simulation implementations, we will focus on modeling concepts independently of simulation/implementation details as well as observation frames. In order to have an appropriate level of precision with these concepts, we have examined many of the common modeling formalisms used in simulation.

Since we are initially aiming to create an ontology for discrete-event models decoupled from model simulation and model observation paradigms, we want to construct taxonomy based on characteristics of models. We pick structural characteristics of the modeling formalisms as the basis for this classification.

Model formalisms are typically defined as n -tuples where the elements of a tuple are usually *sets* or *functions*. Given a certain type of model (e.g., Markov Chain, Timed Automata, Event Graph, Timed Petri Net, Activity Cycle Diagram, etc.) these elements are defined and a collection of them (an n -tuple) serves as a formal definition of the model. In lieu of having a well accepted formalism, standard terms and definitions [Schriber and Brunner 1996] may also be relied upon. This particularly comes into play with process-oriented models, where there is a great deal of literature on terms and definitions, but comparatively little on formalisms [Zeigler 1976; Cota and Sargent 1992; Silver et al. 2006].

In building the taxonomy, we generally use the following strategy: the elements are identified as independently as possible of any particular type of model and then by grouping, restricting, and/or specializing these elements, the model types in the taxonomy are produced. Care must be taken, since if too much generalization and unification are done, the models lose their individual flavor and possibly their usefulness.

In modeling and simulation, a model can be thought of as representing a mini-world that has its own structure (what does it look like) and behavior (what does it do). In order to define a model's mini-world, its structure and behavior need to be defined. Sets and functions on these sets can be used to define the structure of the mini-world, while additional explanation is usually provided to define the behavior. Since the behavior depends on the structure, we consider structure to be more fundamental. Hence, we use structure for the first-level classification in our taxonomy.

In general, most useful dynamic models will have notions of time and space. In space and time, things happen to change the mini-world. In continuous models, change may be constantly occurring, while for discrete-event models, measurable change occurs only at discrete points in time. Dynamic discrete-event models may therefore be characterized by three *primitive sets* based on time, space and events. Although time may be abstract, it intuitively matches our notions of actual time. We refer to the corresponding set as the time set (T). On the other hand, space may correspond to our notion of the three-dimensional world or it may be highly abstract, essentially giving the state (complete current configuration of the mini-world). Thus, we refer to the corresponding set as the state space (S). Changes to the model's mini-world are caused by events, which in discrete-event models occur

instantaneously. The types of changes that may occur (i.e., what can happen in the model's mini-world) define the event set (E).

Many of the models in this taxonomy will explicitly use these three primitive sets in their definitions, while others will use related sets (e.g., activity, place and entity) from which the primitive sets may be deduced.

Based on their structural characterization (i.e., the sets used to define the model structure), we use the following first-level classification.

- Discrete-Event Models - Time.
- State-Oriented Models - State and Event.
- Activity-Oriented Models - Activity and Place.
- Event-Oriented Models - Event and State.
- Process-Oriented Models - Process and State

Since a great deal of formal work has already been done for State-Oriented and Activity-Oriented models we first concentrate on these two model types and then cover Event-Oriented and Process-Oriented models more briefly.

3.2.1 State-Oriented Models. In the DeMO ontology, State-Oriented models may be characterized by the three sets, S , E and T .

- State space (S) - A model changes over time from state to state. Its current state at a particular time is quantified by the values of a collection of indicators. Indicators record various properties of the model. The set of all possible combinations of indicators for this model is called the state space S .
- Event set (E) - An event characterizes a type of event instance that may occur. An event instance occurs instantaneously at a particular time and may cause a state change and/or future events. Event $e \in E$ is enabled at time t_1 and fires at time t_2 . The difference between these times is referred to as the event lifetime. The combination of the event and its firing time may be viewed as an event occurrence (e, t).
- Time set (T) - A nonnegative number represents the passage of time. Time $t \in T$ is integer-valued (e.g., $t \in \mathbb{Z}^+$) for discrete-time models and real-valued (e.g., $t \in \mathbb{R}^+$) for continuous time models (vector-valued time is also a possibility).

These three fundamental sets establish the type of world that is being modeled. We will consider models in which S is finite (finite state models), as well as, models in which S is countably infinite (infinite state models). The size of the set E indicates the number of distinct types of events that can occur and must always be finite. Finally, the time set, T may be countably infinite (discrete time models) or uncountably infinite (continuous time models).

In addition to these three sets, three functions are needed to define the structure of the model (or drive the model). These driving functions will allow the definition of many types of State-Oriented models including Generalized Semi-Markov Processes (GSMP). Other types of Markov models allow for simpler definitions, which we will indicate as restrictions on these functions. To provide a more intuitive feel for these functions, we first define them as deterministic functions, although any or all of them may be stochastic.

—Activation Function - $a : S \times E \rightarrow \{0, 1\}$

The activation function a is a Boolean function indicating which events a state enables. Enabling an event creates an instance of that event which is said to be active until it fires or is cancelled. This function determines *what can happen*.

—Clock Function - $c : S \times E \rightarrow T$

When each event instance is created, a clock timer is set and when it reaches zero, the event fires (or occurs), unless already cancelled. Other functional forms for clock functions are also possible. We have chosen to make the clock setting dependent on the current state and the type of event. In Haas [2002], the clock function may also depend on the prior state as well as the combination of events causing the prior transition. This function determines *when things happen*. Note, in the implementation of simulators, the clocking or scheduling of events is typically accomplished by using a future event list.

—State Transition (or Delta) Function - $d : S \times E \rightarrow S$

The occurrence of events drives the model to change over time. The elemental changes from current state s_i to next state s_j are determined by the state transition function, i.e., $s_j = d(s_i, e)$. This function determines *where to go* in response to something happening.

Note, these definitions assume a time-homogeneous model where the laws governing the dynamics do not change over time; otherwise, time would need to be added to the domain for these functions.

We could introduce additional functions to initialize the model. Alternatively, we could extend the domains of c and d to accept *nil* arguments, upon which c gives the starting time and d gives the starting state for the model.

The details of several existing formalisms for State-Oriented models are presented in Appendix B. The Tables II and III summarize this discussion. These models include many popular Markov (e.g. Generalized Semi-Markov Processes and Continuous Time Markov Chains) models as well as classical automata (e.g. Deterministic Finite Automata).

Table II. Sets and functions for State-Oriented models.

Symbol	Name
S	State Space
E	Event Set
T	Time Set
a	Activation Function
c	Clock Function
d	Transition Function

3.2.2 Activity-Oriented Models. State-Oriented models utilize the concepts of state, time and event. Although state can be thought of as a location (where the model is) it is really an indicator of the model's configuration. One way to think about it in terms of location is to imagine there is an *entity* (or token) moving around in the models. The entity's current location marks the current state (as one would do when creating an animation of a Markov model). Doing this is a first

Table III. State-Oriented models.

Model Acronym	Model Name	Citation
GSMP+SE	Generalized Semi-Markov Process (w/ Simultaneous Events)	[Haas and Shedler 1987]
GSMP-SE	Generalized Semi-Markov Process (wo/ Simultaneous Events)	[Glynn 1983]
SMP	Semi-Markov Process	[Levy 1954; Smith 1955]
CTMC	Continuous Time Markov Chain	[Kolmogorov 1933]
DTMC	Discrete Time Markov Chain	[Markov 1913]
STA	Stochastic Timed Automata	[D'Argenio et al. 1995]
TA	Timed Automata	[Alur and Dill 1994]
SA	State Automata	[McCulloch and Pitts 1943]
DFA	Deterministic Finite Automata	[McCulloch and Pitts 1943]

step to filling state-time with entities and having the events move them around (or create or destroy them).

If multiple entities (or tokens) are introduced, the relationship between state, entity and location is no longer so simple. We need locations on all the entities and therefore the state must include all this information, e.g., in some type of state vector. One might try recording the locations of all entities currently in the model, however, since the number of entities varies in time, the state vectors would have varying lengths. To make the vectors have fixed length, a given number of places (or stations) could be specified at which entities (or tokens) reside, until an event (or events) occurs to move them to other places. In this case, the state of model can be represented as a vector \mathbf{n}

$$\mathbf{n}(t) = (n_1(t), \dots, n_k(t))$$

where $n_p(t) \in \mathbb{Z}_0^+$ is the number entities located in place $p \in P$ at time t .

For these types of models, it is also convenient to think of a more composite view of action. For example, service at a station in a Queuing Network may be thought of as an activity.

Therefore, for activity-based models, our fundamental group of sets will replace S and E with P and A as defined below to yield (P, A, T) .

- Place Set (P) - A place is a location within a positional space (e.g., a metric space or topological space) indicating where a token is. Place p is a location within the model (e.g., logically at a node in a graph or physically in an n -dimensional coordinate system).
- Activity Set (A) - An activity can be thought of as atomic behavior and it usually happens over a time interval, which is delimited by a start-event and an end-event. Activities cause entities to move (or using Petri Net terminology, transitions cause tokens to move from place to place).

Appendix C presents the details of the formalisms for Activity-Oriented models including Extended Stochastic Petri Nets as well as Queueing Networks. Other Activity-Oriented model formalisms in the taxonomy can be derived from it. The Tables IV and V summarize this discussion.

Table IV. Sets and functions for Activity-Oriented models.

Symbol	Name
P	Place Space
A	Activity Set
T	Time Set
c	Clock Function
d_i	Input Incidence Function
d_o	Output Incidence Function

Table V. Activity-Oriented models.

Model Acronym	Model Name	Citation
ESPN	Extended Stochastic Petri Net	[Dugan et al. 1984]
GSPN	Generalized Stochastic Petri Net	[Ajmone Marsan et al. 1984]
SPN	Stochastic Petri Net	[Molloy 1981]
ACD	Activity Cycle Diagrams	[Hills and Poole 1969]
TPN	Timed Petri Net	[Ramchandani 1974; Molloy 1981]
PN	Petri Net	[Petri 1962]
FCN	Free Choice Net	[Hack 1972]
ETG	Extended Task Graph	[Baer 1968]
QN	Queuing Network	[Jackson 1963]
PFQN	Product-Form Queuing Network	[Chandy et al. 1975]
MG	Marked Graph	[Genrich and Lautenbach 1973]
DFN	Decision Free Net	[Genrich and Lautenbach 1973]
TG	Task Graph	[Baer 1968]

Note, we have yet to include Colored Petri Nets [Jensen 1979; 1981] in our ontology. In Colored Petri Nets, the tokens (entities) are classified into multiple classes distinguished by different colors. For Queuing Networks, this is the same as having multiple classes of customers. In the similar manner other conceptual additions can give rise to new formalisms.

3.2.3 Event-Oriented Models. It is often the case in State-Oriented models that the size of the state space becomes very large. Usually, the number of distinct types of events stays relatively small. One could therefore turn a state transition diagram inside out, by changing the nodes from representing states to representing events. The three primitive sets (S , E , T) would still apply, with events and relationships between events now taking center stage.

Event Graphs (EG) follow this approach [Schruben 1983]. An Event Graph depicts the causality between events. Each node in the directed graph represents an event, while arcs indicate that one event may cause another event. The arcs may be also labeled with conditions and time delays. State changes are indicated by a set of assignments to state variables at each node. In other words, the state space need not be defined explicitly, values of the state variables define a state of a system at each point of time.

Simulation Event Graph models [Yucesan and Schruben 1992] provide an extended formal framework for Event Graphs. We use it as a top formalism for Event-Oriented models.

The set of events (event types) is explicitly specified. This event set E is identical in meaning to an event set in State-Oriented models. In Simulation Event Graphs, events correspond to nodes (vertices).

Instead of explicitly specifying the state space S of the system, however, state variables are defined in Event-Oriented models. Each event has a set of state variable changes associated with it. Together with initialization of the state variables, this allows one to obtain the state of the system (as a combination of state variable values) at any point during the simulation of the model, but not to specify the state space explicitly before running the model.

Sets of scheduling edges and canceling edges relate events to each other: if an event e_1 is connected to an event e_2 with a scheduling/canceling edge, that means that the occurrence of event e_1 prompts scheduling/canceling of event e_2 .

Our notion of clock function from the previous section corresponds to Yücesan and Schruben's set of edge delay times, e.g., if occurrence of event A schedules an event B , then the edge connecting A and B may have time delay τ associated with it (i.e., B is scheduled to occur τ units from occurrence of event A).

A transition function (how does the state change) can also be defined in a similar manner, in order to determine how the next state is computed from the current state and current event.

Following the notation of Yücesan and Schruben's, a set of edge conditions needs to be specified. This roughly corresponds to the activation function of a previous section. The current event along with the current state, determine the next event.

In order to allow for tie breaking of the simultaneously occurring events the set of event priorities should be specified as well. It can be defined as a priority ranking function.

In the event-scheduling interpretation, the mechanics of the model will work the following way. At the initialization event all the state variables are initialized. Then all the scheduling edges coming out of the initialization event are examined, their conditions checked and if they hold the corresponding events are scheduled according to the time delays and priority ranking function. The first event on the resulting event list then occurs. The process is now repeated with canceling edges also considered.

It is interesting to note that Simulation Graph Models can be transformed from event scheduling world view to activity scanning world view and vice versa [Schruben and Yücesan 1989]. This sort of duality is important to keep in mind when constructing an ontology.

Table VI. Event-Oriented models.

Model Acronym	Model Name	Citation
EG	Event Graph	[Schruben 1983]
SGM	Simulation Graph Models	[Yücesan and Schruben 1992]

3.2.4 Process-Oriented Models. The process interaction world view is richer and more widely varied than the other world views. Consequently, developing a concise and adequate formalism is quite a challenge.

There has been less work on formal definitions for the process interaction world view than for the other world views. Zeigler [1976] describes a structured model for process interaction using elements of the DEVS formalism. Cota and Sargent [1992] modify the process interaction world view to support full encapsulation, while still allowing preemption of one process by another process. Schriber and Brunner [1996] provide terminology for the process interaction world view where they discuss the transaction flow approach to process interaction simulation. They describe processes as transactions that are discrete units of traffic moving from point-to-point in a system, while competing with each other for scarce resources. Silver and others [2006] review the work of Zeigler, Cota and Sargent, and Schriber and Brunner as a basis for developing a process interaction ontology to be used in ontology based representations of Process-Oriented models.

Following these ideas for the transaction flow approach to process-interaction world-view, simulation models that conform to this world-view may be described as having two types of components - active (processes) and passive (resources). Processes take actions which can change the state of the system, while resources are unable to directly take actions. A process can be thought of as being made up of a set of descriptive variables, a time left in the current state, and a set of computation segments. Each of a process' computation segments consists of a condition C under which the segment will be activated, an activation function f that determines the actions to be taken when the segment is activated, and a label l which points to the first statement of the segment.

When a simulation begins, activation notices are placed onto a list known as a Future Activations List (FAL). Each notice contains a process, the reactivation time of the process, and label value representing the reactivation point of the process. The FAL is ordered by reactivation time. It is then processed in sequential order by removing the first activation notice which contains the current process. The simulation clock is set to the current process' reactivation time and execution of the process begins at the computation segment specified by the reactivation point. If the condition C associated with the segment evaluates to true, the segment's activation function f is applied. Among other things f may change the state of the system and schedule activation notices for the process or its influencees.

Components within the process-interaction simulation interact with one another as influencers and influencees. Influencers (components which influence other components) and influencees (components that are influenced by other components) can be represented using a graph where the nodes are components of the model and the edges indicate which components influence other components. The state of influencers may be used in the computations of influencees' conditions and activation functions. Influencers' functions may also change the state of influencees. A more detailed treatment of process-interaction simulation is covered in [Silver et al. 2006].

Overall, there are two common approaches to process interaction simulation: the transaction flow approach and the active server approach, with the former being more popular [Henrikson 1981; Carson 1993]. With the transaction flow approach the focus of the model is on the actions taken by active processes (transactions or entities) on the systems passive resources (e.g., servers) [Cota and Sargent 1992]. Entities are viewed as executing simulation logic as they pass through the system

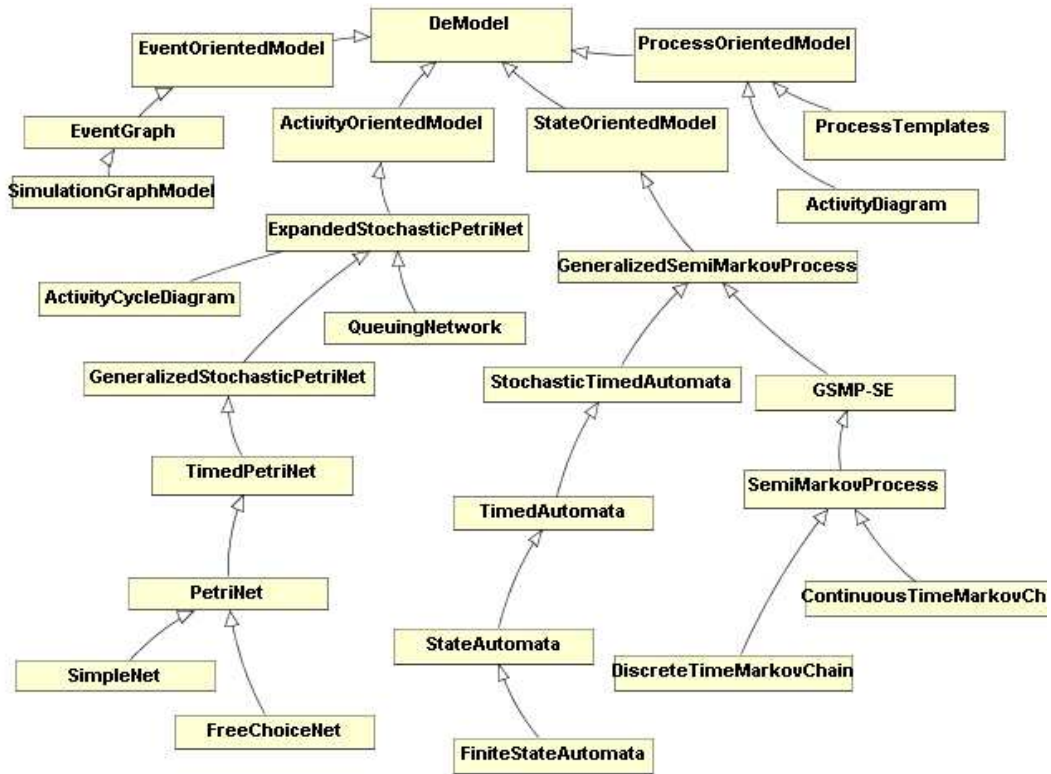


Fig. 1. Portion of the DeMO’s backbone taxonomy.

which may be represented as a block diagram or a directed graph. The entities (processes) in this approach may be implemented using threads or coroutines. The active server approach differs from the transaction flow approach in that it focuses on the actions taken by the system’s resources. In this approach the entities are seen as passive and the resources are seen as active. In the active server approach, resources may be implemented using threads or coroutines that wait for the arrival of entities and then execute logic to provide service for those entities [Pasquini and Rego 1998].

Currently, DeMO includes a subset of the ontology in [Silver et al. 2006]. This part of DeMO, however, needs to be expanded and better integrated with the rest of DeMO. Because of the complexity of the process-interaction world-view, this is no easy task.

4. DEMO ONTOLOGY

The previous section discussed a backbone taxonomy (see Figure 1) for discrete-event models to serve as a foundation for the development of the Discrete-event Modeling Ontology (DeMO). The choice of the structure of the taxonomy, as well as the particular details of the model hierarchy may be open to debate and/or modifications. In our opinion, the resulting backbone taxonomy must be a compromise

between understandability, elegance, current practice and uniformity.

As mentioned before, an ontology is more than a single taxonomy – it should contain formal specifications of all concepts related to the knowledge domain in consideration and the relationships between these concepts. It may indeed implicitly capture multiple taxonomies. In fact, it is not absolutely necessary to have a backbone taxonomy in order to build an ontology. However, having a well-defined taxonomy makes it easier to “grow” the ontology, to categorize the underlying concepts, and to recognize the existing relationships. In this section, we will put all the pieces of the discrete-event modeling knowledge domain together and indicate how they are mapped into the constructs provided by the Web Ontology Language (OWL).

It must be noted that we do not claim to have all possible discrete-event modeling techniques included in the DeMO ontology or to capture all the nuances of the modeling formalisms. Nor do we claim that the approach used in building this ontology is in some way the best. DeMO was designed as an OWL-based prototype ontology whose purpose was to demonstrate the feasibility of this approach in general, as well as, to bring to the forefront potential benefits and pitfalls of the methods and principles used in its construction. Some of the decisions made in the process of building DeMO were not rigorously justified, but rather based on authors’ intuition and sometimes directed by the limitations of the tools currently available. Further developments and research are anticipated before any finalized version of the modeling ontology will take shape. The eventual choice should be made by consensus, as has been done in other fields that have developed ontologies.

4.1 How Much Knowledge Should DeMO Have?

When building DeMO, we are not focused on supporting any one particular application or usage (though some of the possible applications are discussed in Section 5), rather it aims to be “generally useful”. It may seem, therefore, that our main goal at the moment is to capture as much knowledge about discrete-event modeling domain as possible, i.e., provide machine-processable formal specifications for all related concepts and their relationships (at least within the scope defined by our backbone taxonomy). This, of course, may prove to be a neverending task, since most concepts are based on other concepts, which, in turn, may be defined in terms of yet other concepts, and so on. The process may, in principle, go on forever and thus a decision about cut-off points of conceptualization has to be made. The cut-off points (in general, application specific) may serve as a set of axioms or base concepts on which the ontology is developed. Since we did not want to commit to any particular application, our choices of base concepts were mostly dictated by common sense and convenience (a compromise between the desire to include as many details as possible and the increasing complexity of the expanding concept space).

To create any ontology with a high level of precision does require the definition of many base concepts. It is often the case, however, that some of these concepts come from different fields and it is better to have them formalized by experts in these fields. The DeMO ontology, for example, may need some fundamental mathematical concepts to be included. One way to do this is to link the ontology to another already developed ontology (sometimes called the upper ontology) such as SUMO

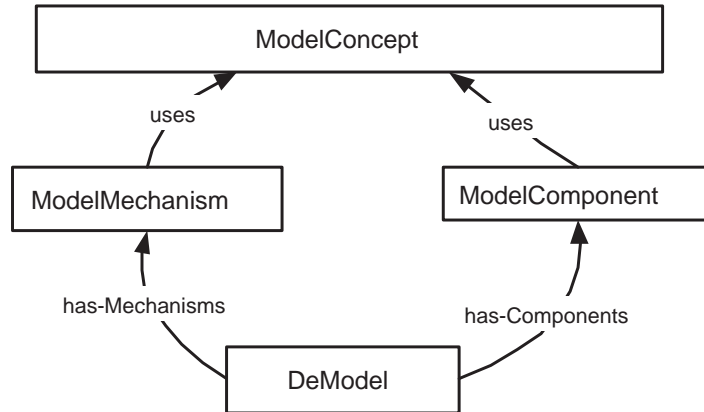


Fig. 2. Schematic representation of the rationale behind DeMO design. Any DeModel is built from ModelComponents and is “put in motion” by ModelMechanisms. These, in turn, are defined using fundamental ModelConcepts.

or SUO, for example. This approach seems to be promising and will probably be widely used in the future when ontologies from base knowledge domains are finalized.

It should be mentioned that a growing effort to create machine-processable specifications for mathematics exists in the Web community. MathML [Carlisle et al. 2003] and OpenMath [Caprotti et al. 2002] are two examples of standards for representing mathematical objects (OpenMath in addition provides means of encoding the semantics of mathematics). The Mathematics on the Net [MONET 2004] project is subsequently developing OWL-based mathematical ontologies using the OpenMath and MathML initiatives.

At the moment, since SUMO, MathML, and OpenMath are still being refined (and to avoid unnecessary complications), we decided to investigate the option of using them at a later time, so at this point our ontology defines some simple mathematical concepts such as set and function on its own.

Yet another point to consider is that the ontology may also serve as a template for defining modeling formalisms and/or techniques currently not included. Therefore it is desirable for the structure of the ontology to allow such scalability.

4.2 DeMO Building Principles

As mentioned in Section 2.4 in OWL DL (which is based on description logic) the concepts are defined as classes with sets of properties and relationships with other classes. New classes can be defined using enumerations of instances, restrictions on properties, and logical statements.

We start from stating the obvious fact that all discrete-event models should have some basic components from which they are built, as well as, some rules (mechanisms) providing specifics of how the models should run (syntax and semantics of the model). To define the components and mechanism we need, in turn, a collection of some fundamental concepts. This is schematically illustrated in Figure 2.

To capture this rationale in the ontology, top abstract classes DeModel, Mod-

elComponent, ModelMechanism, and ModelConcepts are created. The subclasses of the class DeModel correspond to modeling techniques defined in our backbone taxonomy, such as Markov Chains, Queuing Networks, Petri Nets, etc. The subclasses of the class ModelComponent define the building blocks of the model. In our approach, they roughly correspond to the elements of the n -tuples used in formal definitions of the models. The subclasses of the class ModelMechanism define how the components operate within the model. The class ModelConcept has subclasses representing the most fundamental concepts pertaining to this knowledge domain.

In other words, in order to define the DeModel subclass within the ontology one should define the appropriate subclasses of ModelComponent, place them in the DeModel (i.e., relate the DeModel class with these components), define the appropriate subclasses of the ModelMechanism explaining how these components work, and relate the component classes to mechanisms. In our view, this approach will “keep the door open” for other upper-level formalisms not yet included in DeMO.

The distinction between ModelComponents and ModelMechanisms seems to be a bit arbitrary - some models may be reformulated in such a way that a mechanism can become a component and vice versa. However, it also seems that separating these two concepts provides for greater flexibility and usability of the model specifications. (More generally, this situation can be interpreted as defining a borderline between the syntax and the semantics of the model. Having such an explicit borderline and being able to move it by redefining model formalisms may be very useful when reasoning about the ontology).

Note that the subclasses of ModelComponent (as well as ModelMechanism) may themselves form hierarchies and even be organized in taxonomies. These hierarchies are inherited by the DeModel subclasses and give rise to natural taxonomies (hierarchies) within DeModel class different from our backbone taxonomy. We say that these hierarchies are implicitly defined for the ontology. One may argue that ultimately a good ontology should seamlessly capture all (or most) of the naturally existing hierarchies within itself (something fundamentally different from a taxonomy). That may include parts of the backbone taxonomy as well (even though we define it explicitly by enforcing `a-subclass-of` relations, it should still “emerge” as a natural hierarchy).

4.3 Defining the Base Terminology

ModelComponents and ModelMechanisms should be formally defined in terms of the subclasses of the abstract class ModelConcept which are the basic concepts of the DeMO ontology (such as state, event, time, etc.). This means that the subclasses of ModelConcept are not defined in terms of other concepts (some basic concepts may be defined in terms of each other though) and represent the cut-off in our categorization. If none of these fundamental concepts reflect the relations to other non-basic classes in their definitions, then we can achieve a greater flexibility by placing ModelConcept in a separate ontology (a glossary of basic terms, if you like). This modular approach may prove useful if several different ontologies operating with the same glossary are considered.

The subclasses of the ModelConcept are *State*, *Event*, *Time*, *Transition*, *Activity*, *Place*, *Token*, *Entity*, *Process*, *Resource*, *Color* and *Clock* (see Figure 3).

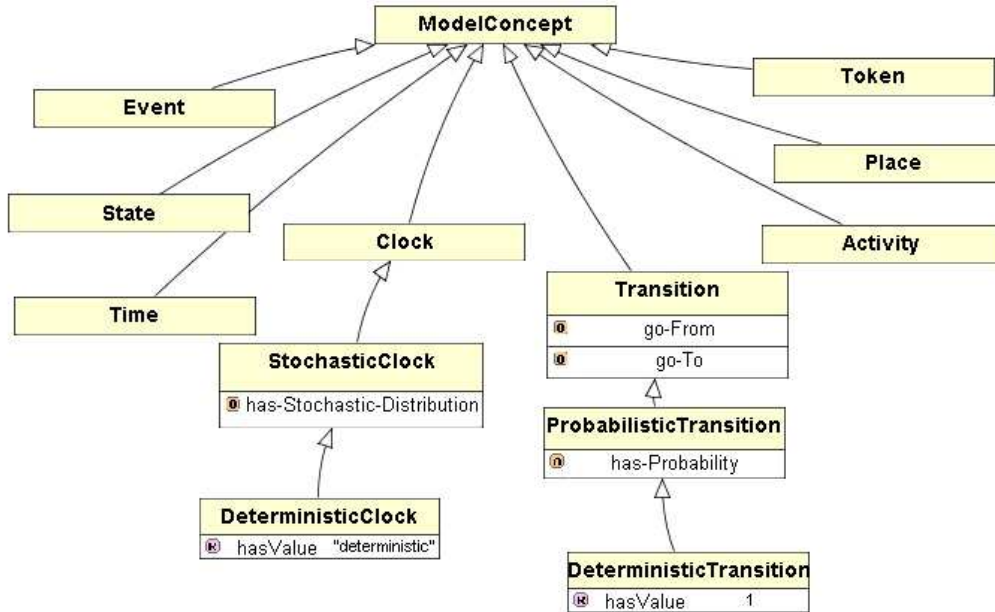


Fig. 3. Portion of the ModelConcept ontology.

Concepts like state, event, and time are fundamental concepts and usually are not formally defined even in English terms. Instead a loose definition (explanation) is usually provided that is intuitively clear for a human reader. We cannot hope that these concepts will be “intuitively clear” for a computer, without providing some sort of formal specification. However, it may not be necessary for many applications – the “translation” of the basic concepts may be hard-wired in the application itself. What we do hope for is that once a machine “understands” the basic terms the rest of the ontology also becomes machine-understandable.

Within Protégé’s OWL environment, we then simply provide an explanation in English and place it in a Documentation slot. For example, class State is described as follows:

“A model changes over time from state to state. Its current state at a particular time is quantified by the values of a collection of indicators. Indicators record various properties of the model.”

This description is designed to provide a short explanation for a human reader, but is absolutely obscure for any machine. In addition, properties such as Name and Value are assigned to the class State.

The class Time is simply defined as: “A nonnegative number representing the passage of time.” Note that one may still choose to define these classes in terms of other, new classes (e.g., define State in terms of a new class StateVariable and its subclasses), or classes from other ontologies (e.g., a class RealNumber from an upper mathematical ontology may be used to define Time), thus expanding the categorization. As we noted before, this choice must be made prior to constructing an ontology. (Alternatively, one may think of a way of moving the cut-off points of

the ontology, based on some control parameters. This will create a kind of multi-leveled ontology, where level 1, for example, has the class State as a basic concept, and level 2 has subclasses of StateVariable class as basic concepts.)

Some of the basic concepts are related to each other and that can be used to effectively “define” one concept in terms of the others. For example, we may associate time with the occurrence of events. Thus the class Event will have a property *canOccurAt-instance-of-class-Time*.³ In other words we “define”, though incompletely, the class Event as the class having the aforementioned property.

Yet another possibility is to “define” the basic concepts in terms of their relations to other existing concepts, not in the ModelConcept class. Consider, for example, the class Transition representing “a state change that is caused by one or more events”. We have a pair of ModelMechanisms in DeMO that are associated with the class Transition. A transition can be enabled using some instance of a TransitionEnabling mechanism. It can also be triggered using an instance of TransitionTriggering mechanism. There are a couple of ways to implement these relationships. One is to assign a property to each of the two ModelMechanisms relating them to the class Transition. Say, *isAppliedTo-instance-of-class-Transition*. Another is to assign two properties to the class Transition: *isEnabledBy* and *isTriggeredBy*. In the latter case, the class Transition is effectively defined as a class with these two properties related to the two ModelMechanisms (i.e., in this case the two ModelMechanisms are, in fact, playing the role of the basic concepts). As in the case of backbone taxonomy before, we can see that the “quality” assigned to a certain concept may become ephemeral with a slight difference in approaches.

It is hard to decide in general which approach is better in the case above. On one hand, once we define a Transition in terms of ModelMechanisms it ceases to be a basic concept (understood as being independent of concepts other than the basic concepts). Then, placing a ModelConcept in a separate ontology makes no sense, since it is now “dependent” on another ontology, and we lose the structural modularity offered by the ontology separation. On the other hand the links from ModelConcepts to ModelMechanisms and/or other classes may turn out to be beneficial, allowing, for example, for more efficient inference of information. As usual, the answer probably depends on the application.

4.4 Adding the Backbone Taxonomy

Once we decided on the ontology building principles and defined the basic concepts and the model components, we can recreate our backbone taxonomy of discrete-event modeling techniques. As stipulated in Section 4.2 the abstract class *DiscreteEventModel* (called here DeModel for short) splits into four first-level subclasses:

- State-Oriented Model*,
- Event-Oriented Model*,
- Activity-Oriented Model*,

³Following Protégé conventions for OWL we capitalize the first and remaining words in *class names* while keeping the first word in small letters for *properties* and italicizing it. A *property together with its range* is denoted using the words instance-of-class-SomeClassName e.g. *canOccurAt-instance-of-class-Time*. Class *instances* are denoted using typewriter font style with all words capitalized.

—*Process-Oriented Model.*

This is a *first-level classification*. Each of these classes defines a top formalism for the underlying subclasses in accordance with specifications discussed in Section 3 and Appendices B and C. One way to implement it in OWL is to define the subclasses of `ModelComponent` and `ModelMechanism`: `StateOrientedComponent`, `StateOrientedMechanism`, `EventOrientedComponents`, etc. Then, obviously, any state-oriented modeling technique (any subclass of `StateOrientedModel`) is defined by putting restrictions on the properties defined by subclasses of `StateOrientedComponent` and `StateOrientedMechanism`.

Let us look closer at the State-Oriented Models. The top abstract class (abstract class has no instances) has the following properties⁴:

Property Name	Property Range (<i>Instance of</i>)
<i>has-StateSpace</i>	<code>DiscreteStateSpace</code>
<i>has-EventSet</i>	<code>FiniteEventSet</code>
<i>has-TimeSet</i>	<code>TimeSet</code>
<i>has-ActivationFunction</i>	<code>ActivationFunction</code>
<i>has-ClockFunction</i>	<code>ClockFunction</code>
<i>has-InitialState</i>	<code>InitialState</code>

Note that while the `StateSpace` and the `EventSet` are presented by their restricted subclasses `DiscreteEventSpace` and `FiniteEventSet`, respectively, none of the functions is restricted. The rest of the State-Oriented Models are rooted in this class.

For the `StateOrientedModel` the top subclass is a Generalized Semi-Markov Process (GSMP) model; for the `EventOrientedModel` the top subclass is Simulation Event Graph (SGM); and for `ActivityOrientedModel` this is Extended Stochastic Petri-Net (ESPN).

Consider, for example, the Generalized Semi-Markov Process (see Appendix B), the highest concrete class in the State-Oriented model formalism (Figure 4). It is defined by placing the following restrictions on the properties presented above:

ClockFunction is restricted to **StochasticClockFunction**
TransitionFunction is restricted to **ProbabilisticTransitionFunction**

In addition it uses the following instances of subclasses of `ModelMechanism`:

Mechanism Name	Value
<code>TransitionTriggering</code>	MultipleEventTriggering mechanism
<code>TransitionEnabling</code>	EventEnabling mechanism
<code>EventScheduling</code>	StateMachineScheduling mechanism.

Here the **MultipleEventTriggering** mechanism is an instance of the `TransitionTriggering` mechanism, which allows simultaneous event occurrence to trigger a transition, the **EventEnabling** mechanism means that the transitions are enabled together with events, and the **StateMachineScheduling** mechanism determines the set of imminent events.

The restriction on the `TransitionTriggering` mechanism is the only difference between GSMP and GSMP-SE. In Protégé’s OWL, this is specified merely by overriding this particular restriction:

⁴All *has-* properties are defined as subproperties of a more general *has-a* property.

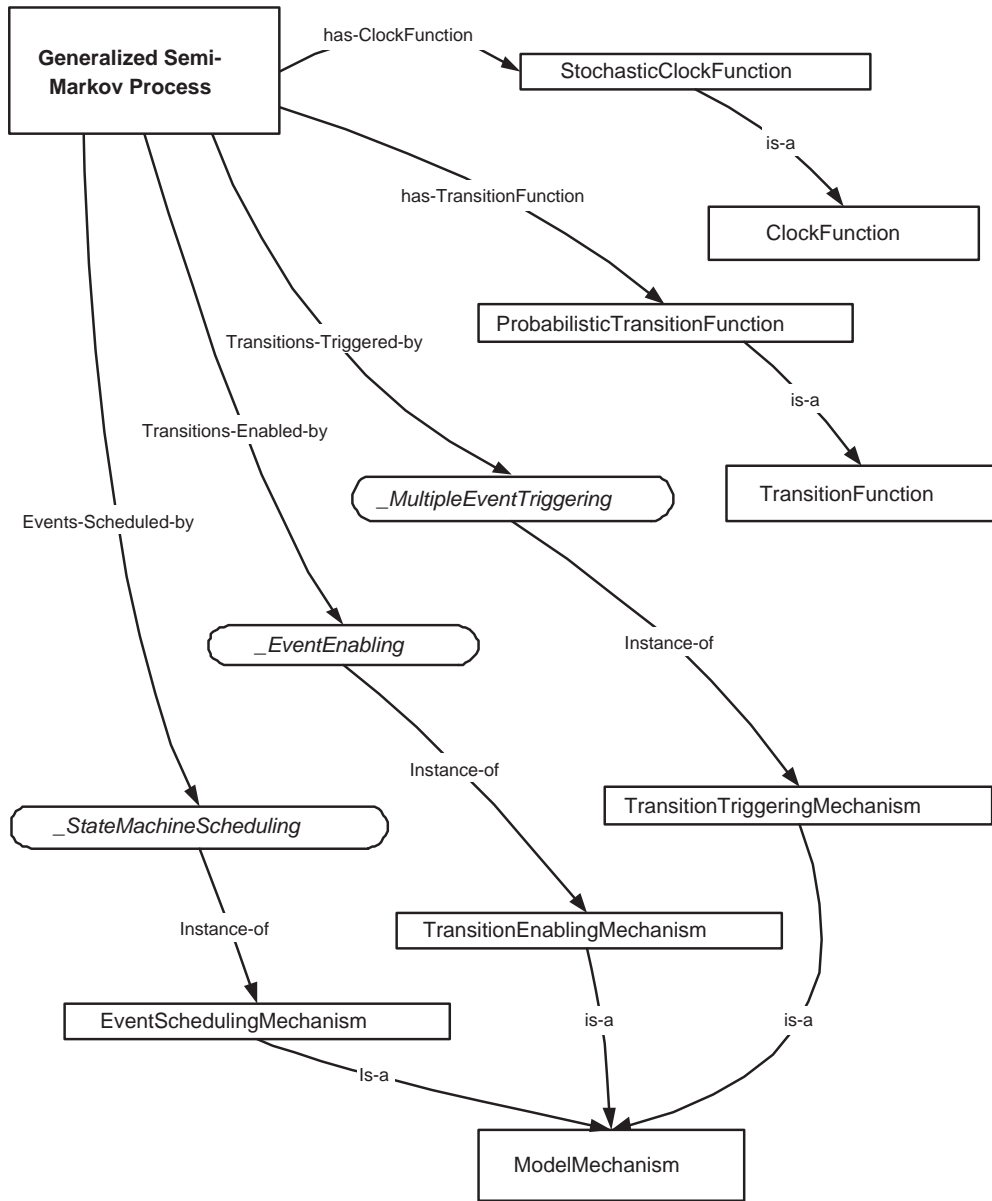


Fig. 4. Graphical Representation of GSMP Portion of the DeMO Ontology.

TransitionTriggering has value **SingleEventTriggering** mechanism,

where **SingleEventTriggering** mechanism is defined to allow only the single event occurrence to trigger the transition.

All other properties are inherited from GSMP and need not be redefined. The class **StochasticTimedAutomata** is defined exactly the same way and constitutes a

class synonym of GSMP-SE.

It is easy to see how other state-oriented models can be defined in the same manner. `StateAutomata` is defined by restricting the `ClockFunction` to be `NoClock`, and `TransitionFunction` to `DeterministicTransitionFunction`. `DeterministicFiniteAutomata` is defined by additionally restricting the property *has-StateSpace* to be an instance of `FiniteStateSpace`.

An interesting phenomenon takes place in case of `SemiMarkovProcess`. This class 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 the `MarkovianSheduling` mechanism. On the other hand, effectively the same model will be produced by putting restrictions on the cardinality of the `EventSet`, namely, setting it equal to one. (This ambiguity in derivation may not be desirable, but may also turn out to be unavoidable. This issue will be addressed with more attention in our future work.)

When considering Activity-Oriented models in general and Petri Nets in particular, we need to define the properties that restrict the topologies of the underlying Place-Transition nets. Indeed a number of different classes of Petri Nets are defined by merely placing topological restrictions on the P-T nets. These types of restrictions are hard to define using OWL and currently (just as the subclasses of `ModelMechanism`) the topological restrictions are defined using English descriptions.

The resulting taxonomy of P-T nets can then be superimposed with other properties of Activity-Oriented models, such as related to `ClockFunction`, for example. A number of new classes can be defined that way. This raises an interesting question: what do we do with modeling techniques that can be easily derived from the higher formalism, but have no common name in the scientific literature? Do we include them in the ontology using some automatically generated names or do we lump them together under the common general name? How do we differentiate Free-Choice Nets with stochastic timed transitions from the Free-Choice Nets with immediate transitions? A related question is – what is the best way to deal with the names of modeling formalisms that are used interchangeably in the literature? List all the synonyms, or enforce a single standard? Ideally, we feel that the answers to these questions should be sought by consensus.

4.5 Model Morphisms

Many formalisms can be mapped onto other formalisms, by some sort of transformations. These transformations will take a model instance of one formalism to an instance of another and can be defined in ontology as well. This can be done using SWRL rules, for example, that will take model components and mechanisms of one formalism and morph them in components and mechanisms of another. In some cases such morphisms are easily achieved, in some it may be nearly impossible, but it is most likely that there will be always at least an indirect “path” from one formalism to another, through a series of such transformations. In other words, the transformation graph (vertices – formalisms, edges – transformations) will most likely have one connected component.

Each such morphism can be defined as an ontological class and represent a property of a DeModel: *formalism A maps to formalism B using morphism M*. In fact,

a formalism A may have several different morphisms to a formalism B.

Model mapping can be approached from different directions. One approach is to find some common formalism, that most other formalisms easily map to and use it as a “mapping hub”. For example, Vangheluwe, proposes DEVS as a common target for many formalism transformations [Vangheluwe 2000]. On the other hand it may be more efficient to find direct mappings between different pairs of formalism whenever possible and use series of transformations between “distant” formalisms. (See also Caplat and Sourrouille [2002] on model mapping in model driven architecture.) Whatever the case, there is no reason why all of these should not be eventually incorporated in the ontology.

Currently, morphisms are not included in DeMO, but the plans exist to do so, for they are envisioned to play key roles in some of the usages of the ontology.

4.6 Observable Models

A model is run to produce results. The results, typically, come from the output produced. The output could be very simple or quite complex. For example, for a DFA, an output of 1 means accept, while 0 means reject, thereby, allowing one to check if a string is in a language. At the other end of the spectrum, a GSMP may produce a set of sample paths that require statistical analysis to have any meaningful results.

We have not defined the output for the models so far, since we view the model as exhibiting behavior and then one may observe it in different ways to produce distinct kinds of output (e.g., DFA may give rise to both Moore machines in which outputs depend only on state information [Moore 1956] or Mealy machines in which outputs depend on both states and inputs [Mealy 1955]).

When talking about ontological description, it may be beneficial to decouple the model dynamics (what is really happening as the model executes over time) from how it is observed [Miller and Baramidze 2005]. This way the same model can be “observed” in many different ways. In the ontology it can be represented by “attaching” different output functions to the modeling formalism, allowing one to capture more knowledge with a smaller ontology.

A “perfect” output function gives the observer a direct access to all dynamic variables of the model. In general, however, the observation window may be restricted to only certain aspects of a modeled system. This may be represented by an output function that takes the “perfect” output to a more restricted form. This function may have stochastic behavior as well. In the case of Hidden Markov Models, for example, we can imagine a regular Markov Chain formalism with a stochastic output function that probabilistically maps a current state of the system to one of the output symbols.

The DeMO ontology concentrates only on model specifications. Currently, output function and related concepts are not represented in DeMO, and generally may constitute an extended ontology of their own, that, when combined with DeMO, will give rise to an Observable Model paradigm.

Similarly, the model can be decoupled from how it is simulated. Different simulators may have vastly different implementation approaches for identical models. Just as with output functions, one can imagine “attaching” different simulators to observable model formalisms.

5. PUTTING THE DEMO ONTOLOGY INTO PRACTICE

An ontology for discrete-event modeling and simulation, such as DeMO, ultimately should be judged by the benefits it brings. Generally, an ontology may be looked at as a knowledge repository about a specific domain. The ontology for discrete-event modeling should therefore specify the concepts used in this domain and relations between them. It must be emphasized that such an ontology should not be considered as some sort of unifying superformalism for discrete-event models. Instead, it is a collection of explicit knowledge concepts about different existing formalisms. Any discrete-event modeling formalism may, in principle, be “inserted” in this repository and the relations to other formalisms and the underlying concepts may be represented as well. Placing the ontology on the Web makes it a centralized, machine-readable knowledge repository that can be accessed and used by either modelers or computer applications in many different ways. The presence of this explicit knowledge on the Web organized in such a fashion is a useful step in the evolution of Web-based modeling and simulation. The ontology itself will grow and evolve through combined efforts of the entire modeling community. As a result, new, maybe unexpected, uses of and directions for DeMO ontology development would emerge in time, driven by the needs of modelers and the demands of different application domains.

5.1 Ontology Driven Modeling

One of the potentially useful applications of DeMO is ontology driven model construction and simulation modeling. Consider the entire process of model development. Usually a modeler is presented with a real-world problem and has to come up with a model of a relevant system, which will imitate the system and facilitate the solution of the original problem. This means that the model should allow the use of some readily available tools that can help to analyze or solve the original real-world problem. These tools may be simulators, analyzers, solvers, mathematical formulas, etc. Thus the modeler’s task is to take the real-world problem, abstract and translate it to some formal model (explicitly or implicitly), and then find and apply relevant tool(s) to this model.

With this model-centric view in mind consider placing a modeling ontology (e.g., DeMO) in the middle of this paradigm. We believe that this may greatly help mapping from the application domain (real-world problem) to the modeling and simulation domain and provide robustness and flexibility to the modeler.

5.1.1 Problem \rightarrow Model. Let us restrict ourselves to discrete-event systems modeling. DeMO may serve as a middleman between the modeler/researcher and the simulation and analysis tool. The knowledge expert in some particular field may construct a logical model of the relevant system. The user can then map this logical model to one or several formalisms in DeMO ontology (e.g. Stochastic Petri Nets, or Markov Chains). Intermediate tools can be developed to facilitate this step.

In addition, we can argue that the modeler essentially “maps” her conceptual model on the portion of the ontology. Indeed, assuming that relationships between different formalisms are either explicitly represented in the ontology or discovered by some inference engine, it is sufficient to map the conceptual model on only one of the

formalisms and it will be automatically “remapped” to others. This is a sort of “top-down” approach. Alternatively, one can consider matching parts of the conceptual model with model components (building blocks of different modeling formalisms) present in the ontology, and then looking for the formalisms that contain these components – a “bottom-up” approach.

Traditional modeling approaches. One of the first and necessary steps in model construction is the development of a conceptual model of the system in question.

For example, assume that the modeler already has a conceptual model of some sort in her head. It may be a clear and concise understanding of different entities interacting within the system, or an explicit picture of a finite set of states that the system can go through, or the sorts of activities that can happen in the system and under what circumstances. This itself can be represented diagrammatically, e.g., a sketch of activities for a supply chain management, or a graph representation of the protein-protein interaction network, etc.

The next step is to translate this conceptual model into some formal model description. Usually the modeler has to choose the modeling formalism (or, at least, a class of formalisms) beforehand, and use an applicable model design tool for this particular formalism. With DeMO in the middle, this step may instead turn in the process of matching the conceptual model with one or more of the modeling formalisms in DeMO.

Suppose now that the modeler is at the very first stages of the process, when the conceptual modeling is not finished yet. We argue that DeMO may be of use at this stage as well. A DeMO-based interface tool may assist the modeler with constructing the conceptual model while simultaneously matching the DeMO terms to domain concepts.

One can imagine a knowledge-based system that tries to find more suitable formalisms based on the user’s needs, assumptions, and preferences. For example, questions along the following lines may be asked: “What are the model inputs and outputs?”, “Does your system have any explicit entities?”, “Can you explicitly define the states of the system?”, “What types of events alter the state of your system?”, etc. Depending on her input, a modeler may be offered a set of formalisms to choose from. The modeler can then manually “tune up” her model instance.

Since model design is generally regarded as an art, these interactive processes (possibly with some degree of automatic inferencing) clearly can not be guaranteed to match the real-world problem to the best model instance or even the best modeling formalism. Much still depends on the skill of the modeler, but, in our view, using DeMO ontology may greatly enhance this step of model construction.

Case for markup languages. Many disciplines realized the need for a common format for model description. This often results in the development of XML-based markup languages. Consider the problem of modeling biological systems. These may include biochemical reactions, gene regulation, protein-protein interaction, various cell processes, etc. In recent years a great deal of effort is geared towards properly understanding, representing and modeling these systems. Various ways to model biological systems have been employed (Ordinary Differential Equations (ODE), Boolean networks, Petri Nets, etc.).

Biological systems, especially after the genomic revolution, are characterized by

incredible complexity and a vast amount of data that researchers have to deal with. This overwhelming amount of information that biologists have to deal with forced the biological community to find ways to represent this information in a machine-readable way. Various XML-based ontologies and markup languages have been created and successfully used for information exchange and knowledge discovery.

This also provides a novel and flexible way of representing biological systems for modeling purposes. Using markup languages (e.g., Systems Biology Markup Language (SBML), or CellML) researchers can describe the system of interest without specific applications in mind. Many SBML-compatible tools and applications already exist (as of now more than 90 software systems are supporting SBML and many more are being created). These tools can take a SBML description of a system as an input and produce various operation based on that. These may include model visualization, fast simulations, network analysis and many more. This approach provides biologists access to a great variety of tools based only on SBML description and facilitates exchange of biological models.

The use of markup languages in different domains can bring similar benefits. However, this means that specific tools should be created based on each markup language which besides the extra effort also raises artificial barriers between various knowledge domains.

We believe that modeling and simulation ontologies can also play a role of a mediator between domain specific markup languages and modeling and simulation tools. Indeed, it may be sufficient to have a translator from a markup language to DeMO ontology in order to “map” a system described with the markup language to one or more DeMO formalisms. For example, tools for translating SBML to Petri Net Markup Language (PNML) already exist. *Applying Petri Nets to Systems Biology Using XML Technologies* [Shaw et al. 2004] gives an example of an approach that allows one to translate Systems Biology Markup Language to Petri Net Markup Language. In particular, they take a SBML description of *Saccharomyces cerevisiae* glycolysis pathway and construct a Petri Net model with PNML description. In their approach, each biochemical reaction is represented as a Petri Net transition and reacting species are represented by places.

It must be noted that this approach is natural with biological systems and domains where the concepts are well defined and the logical/conceptual models are straightforward to construct, but may not be as convenient or complete for other domains. Not every domain allows for a straightforward creation of a markup language and, obviously, there would always be a system to be modeled that does not have an appropriate description in any existing markup language (let alone situations when the modeler is simply not familiar with the domain markup language and wants to skip the costly learning curve). DeMO can still be used as a mediator between the application domain and the model as described above in a previous subsection.

Case for domain ontologies. If the knowledge domain is well defined (here this is equivalent to existence of a good domain ontology), the whole domain can be already mapped on DeMO. Consider for example a small domain with relatively few concepts and relations. A small ontology can be used to represent the knowledge about this domain. These concepts may be directly linked to some model compo-

nents in DeMO formalism. For example “*Customer*”-“*Entity*”, and/or “*Customer*”-“*Process*”, “*Arrival*”-“*Event*”, etc. These mappings may give a modeler a direct choice of one or more modeling formalisms, which she can choose from to model some system described by this domain ontology. This may be done once, and reused by different researchers, from that point forward.

5.1.2 *Model* \longrightarrow *Tool*. Various tools can be “plugged” in DeMO from the other side. These tools may include simulators, analytical programs, visualization tools etc. Once the model is mapped onto DeMO, the user can define the instance of the model, which will then be automatically translated in the OWL instances of DeMO formalisms. These instances, in turn, can be translated to XML formats suitable for particular simulators. Here markup languages can also come into play. For example, the Petri Net Markup Language (PNML) is currently being used by many Petri Net Simulators. Other markup languages also exist. Translating DeMO instances to one of these markup languages, is typically straightforward (using XSLT for example).

We can envision many different simulators “linked” to DeMO through XML-based interface, thus giving the modeler a choice of simulators that she may use. This approach, in fact, allows one to separate the user-model interface from the end application. Completely different interfaces can connect through DeMO to different tools and the researchers are presented with a very powerful and flexible system. They can, for example, try to find the fastest simulator for their models or even run several Web-based tools simultaneously and compare the results.

Yet another dimension of flexibility emerges from the fact that modelers may also want to use some analytical tools to study, for example, the properties of the resulting nets or to estimate some parameters of the model (e.g., in the case of Hidden Markov Model). Just as with simulation tools, appropriate XML based analytical tools and solvers may be “plugged in”. As an additional extension to DeMO, previously discovered analytical results related to existing formalisms can be reflected in the ontology as well.

5.2 Ontology-based Knowledge Exchange and Reuse Between Different Domains

Another, potentially very powerful role of DeMO may become a facilitator of knowledge interchange between different scientific communities.

It is often the case that the shortage of modelers capable of designing discrete-event models and simulations results in the situation when many researchers and managers are not aware of the potential power of this approach. Discrete-event modeling ontology and attached interfacing tools may help to expose many more researchers from other areas to discrete-event modeling. Through intermediary translation tools they can even continue to use the language they are used to (e.g., reaction rate for biochemistry, or service time for business management).

The use of a Web-accessible ontology allows one to store and reuse different models and model parts. Moreover, these models may be mapped/translated to other knowledge domain dictionaries and reused by researchers in other areas as well. If, for example, an extensive research and analysis is done on a particular model related to some business process, there is no need to not make these results available to the researchers in other areas. Translating the results into the language

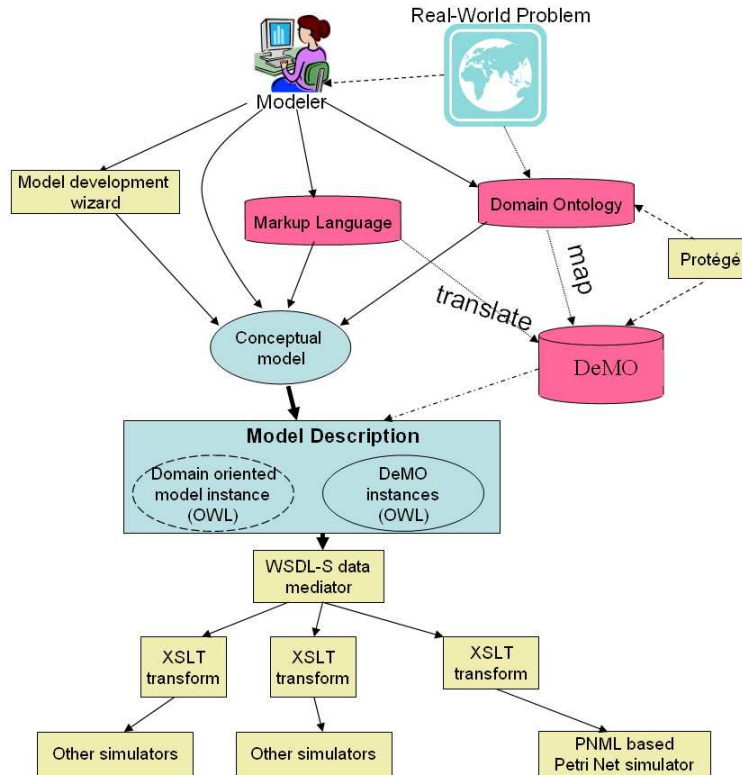


Fig. 5. Ontology driven model development and use.

of DeMO makes them available for ecological modeling, for example, which very well may have a compatible formalism underlying the description of the system they are studying.

Directly going from application domain to simulation tools, bypassing the modeling formalisms (modeling ontology), may raise artificial barriers between different scientific domains. For example, Petri Nets may become increasingly successful and popular in systems biology. If the community will then concentrate their efforts on building tools directly taking systems biology descriptive language to Petri Net formalisms, they, in a sense, separate themselves from other available formalisms and whatever developments that may be happening with them. For example, suppose a new formalism (let's call it Amazing Nets) is being introduced by members of another modeling community (say financial modelers) and proves to be very efficient in solving certain types of problems. Suppose also that someone shows that this formalism can be very successfully applied to certain biological systems as well. It may take years, however, for biologists to start using Amazing Nets, to write new tools or rewrite old ones to translate their problems in the language of new formalism, let alone retraining biologists to use it. If, however, the systems biology modeling process is wired through discrete-event modeling ontology as discussed in

the previous section, and Amazing Nets are incorporated in the ontology as well, then the biologists would almost instantaneously gain access to all the tools related to Amazing Nets the moment they are “plugged” in the ontology.

5.3 Additional Benefits

We envisage several additional benefits or usages for modeling ontologies such as DeMO.

- 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 and Jambalaya plug-ins provide GUI-oriented visualization of the same hierarchy. Figure 6, for example, shows a screen-shot of a part of DeMO ontology using the Jambalaya tool. Each node represents an ontology class, and each edge symbolizes a subclass relation. Other relations can also be visualized in a similar manner. Also, there is some initial work on making OWL tools interoperate with UML tools.
- Querying.* Query languages are being developed to provide powerful querying capabilities for one or more ontologies. Such languages and tools will augment today’s search engines. There are many such languages under development (e.g., RDQL, SPARQL, 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 rule/logic layer (e.g., SWRL) and query languages will follow (e.g., Hoolet (<http://owl.man.ac.uk/hoolet/>)). Given such query capabilities, several types of queries on DeMO would be useful. One could ask queries to find a list of software packages that run for instance discrete-event simulation, process-interaction models.
- Service Discovery.* One could look for a Web service to perform a certain modeling task. For using Web services in simulation modeling see [Chandrasekaran et al. 2002].
- Components.* By providing an ontology, we move towards creation of a Web-based infrastructure for storing and retrieving executable simulation model components. For example, consider `ProbabilisticTransitionFunction` in Figure 4. Code implementations of specific probability density functions can be attached directly to this link, and they are made available to those searching for them. This type of infrastructure can facilitate reuse and composability of simulation models.
- Hypothesis Testing.* The LSDIS Lab is currently carrying out funded research to allow hypothesis testing to be carried out using the Semantic Web. 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 Corporation [Sheth et al. 2002].
- Facilitate Collaboration.* By establishing a shared conceptual framework, opportunities for increased collaboration between researchers are improved. This includes interoperability of simulation tools, model reuse and data sharing.

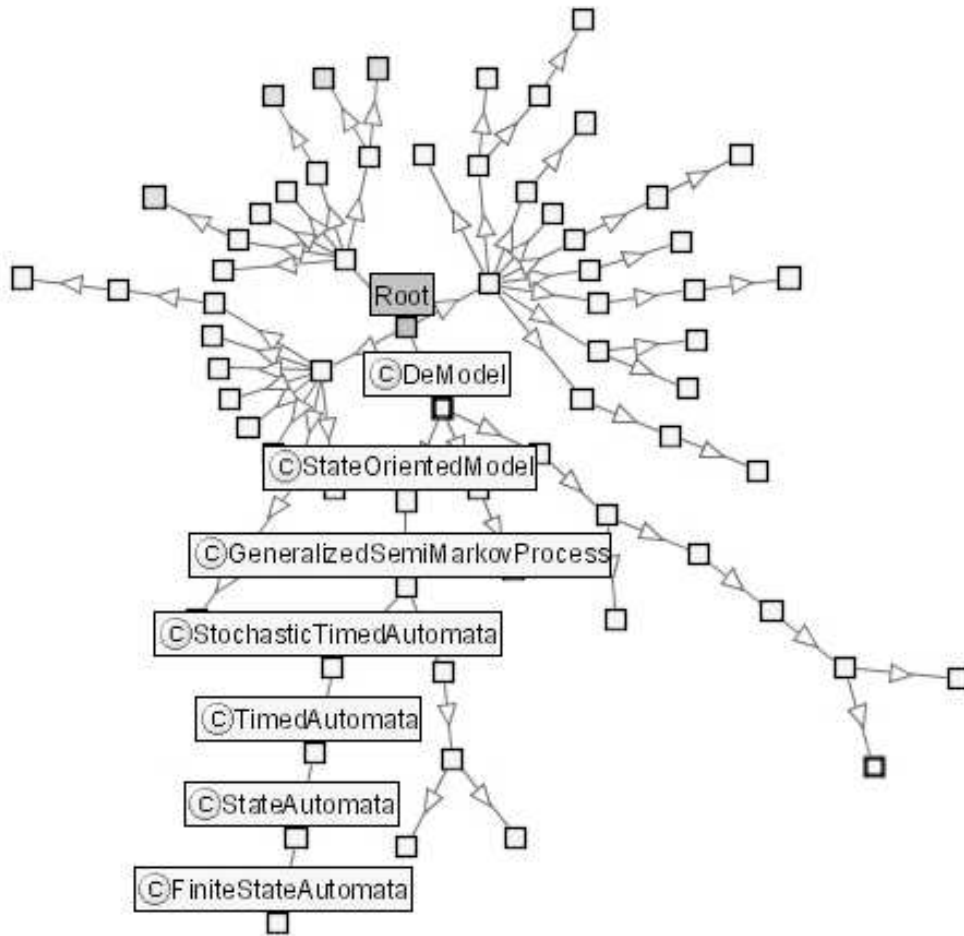


Fig. 6. Portion of the DeMO ontology visualized using Jambalaya Protégé plug-in. In this radial view square nodes represent ontology classes (concepts) and edges symbolize subclass relations. Other relations between the concepts are not shown. The class names presented here represent the path from the root of the ontology to the FiniteStateAutomata node.

—*Consistency Checking.* Since OWL DL is based on Description Logic, there exist reasoners or inference engines that can check the consistency of the concept definitions (e.g., FaCT (<http://www.cs.man.ac.uk/~horrocks/FaCT/>) or Racer [Haarslev and Möller 2001]).

6. CONCLUSIONS AND FUTURE WORK

This paper represents an initial effort 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. Furthermore, ontology driven simulation has a substantial, yet largely untapped, potential.

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 a group 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. When SUMO/SUO becomes a stable, useful and well-accepted upper ontology, it should probably be used.
- Compatibility with Other Ontologies. It is expected that other related ontologies will be developed in the near future. The most relevant to this work would be a future Math ontology or a Simulation Analysis ontology.
- The DeMO ontology has focused on the fundamentals of discrete-event modeling. In the future, it needs to be extended via related and linked ontologies that cover broader aspects of simulation. In this effort, the Body of Knowledge of Modeling and Simulation (M&SBOK) being developed by Ören [2005] and based upon his earlier development of taxonomies (<http://www.site.uottawa.ca/~oren/pubsList/taxonomies.htm>) would serve as a good guidepost due to its comprehensive nature.
- Breadth of the Ontology. If the domain ontology is too broad it may become too complex and disconnected. Ambiguities may be quite difficult to resolve. On the other hand, if it is too narrow, it is of limited use.
- 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.).
- Incorporate Network Diagram (ND) models for simulators such as SLAM, SIMAN and GPSS into our ontology. As mentioned, there is little work on formalization of ND models.
- Incorporate or link to Discrete Event System Specification (DEVS) models which could be viewed as a taxonomy/ontology of their own right [Sarjoughian et al. 2001].
- Semantics in OWL ontologies often relies on the use of natural language to supply meaning where OWL constructs cannot. Use of Rule/Logic layer languages such as SWRL will allow more of the semantics to be captured formally (and therefore be more machine processable). As soon as this layer of the Semantic Web stabilizes, we plan to use it to create a new version of DeMO.
- More precisely define the semantics related to model mechanisms. One way to do this is to use operational semantics like it is done in [Birtwistle and Tofts 1997]. An alternative we intend to explore is to use SWRL to define these mechanisms abstractly.

ACKNOWLEDGMENTS

Paul Fishwick would like to thank his graduate students in the RUBE Project (Minho Park, Jinho Lee and Hyunju Shim), the National Science Foundation working via grant EIA-0119532 and the Air Force Research Laboratory working via grant F30602-01-1-05920119532.

John Miller would like to thank Congzhou He for her help in working on the DeMO ontology.

REFERENCES

- AJMONE MARSAN, M., BALBO, G., AND CONTE, G. 1984. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems* 2, 2, 93–122.
- ALUR, R. AND DILL, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126, 183–235. (preliminary versions appeared in Proc. 17th ICALP, LNCS 443, 1990, and Real Time: Theory in Practice, LNCS 600, 1991).
- BAER, J. L. 1968. Graph models of computations in computer systems. Ph.D. thesis, Dept. of Elect. Eng., UCLA, Los Angeles, CA.
- BECHHOFFER, S., HORROCKS, I., AND PAN, J. 2004. WonderWeb: Ontology infrastructure for the Semantic Web. <http://wonderweb.man.ac.uk/deliverables/documents/D3.pdf>.
- BERNERS-LEE, T. 2000. Semantic Web – XML2000, keynote address. <http://www.w3.org/2000/Talks/1206-xml2k-tbl/>.
- BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. 2001. The Semantic Web. *Scientific American* 284, 5, 34–43.
- BIRTWISTLE, G. AND TOFTS, C. 1997. Relating operational and denotational descriptions of π Demos. *Simulation Practice and Theory* 5, 1–33.
- BIRTWISTLE, G. M. 1979. *Discrete Event Modeling in SIMULA*. MacMillan, London, England.
- CAPLAT, G. AND SOURROUILLE, J. L. 2002. Model mapping in MDA. In *Workshop in Software Model Engineering (WISME)*, J. Bezivin and R. France, Eds. Dresden, Germany.
- CAPROTTI, O., CARLISLE, D. P., AND COHEN, A. M. 2002. The OpenMath standard. <http://www.openmath.org/cocoon/openmath/standard/om11/omstd11.pdf>.
- CARLISLE, D. ET AL. 2003. Mathematical Markup Language (MathML) Version 2.0. <http://www.w3.org/TR/MathML>.
- CARSON, J. 1993. Modeling and simulation worldviews. In *Proceedings of the 1993 Winter Simulation Conference*, G. Evans, M. Mollaghasemi, E. Russell, and W. Biles, Eds. IEEE Press, Piscataway, NJ, 18–23.
- CASSANDRAS, C. G. AND LAFORTUNE, S. 1999. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Dordrecht, Holland.
- CHANDRASEKARAN, S., SILVER, G., MILLER, J. A., CARDOSO, J., AND SHETH, A. P. 2002. Web service technologies and their synergy with simulation. In *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, Eds. IEEE Press, Piscataway, NJ, 606–615.
- CHANDY, K. M., HERZOG, U., AND WOO, L. 1975. Parametric analysis of queueing net. *IBM J. Res. Dev.* 19, 36–42.
- COTA, B. A. AND SARGENT, R. G. 1992. A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation* 2, 2 (April), 109–129.
- CUBERT, R. M., GOKTEKIN, T., AND FISHWICK, P. A. 1997. MOOSE: architecture of an object-oriented multimodeling simulation system. In *In Proceedings of Enabling Technology for Simulation Science, Part of SPIE AeroSense '97 Conference*. Orlando, FL, 78–88.
- DAHL, O., MYHRHAUG, B., AND NYGAARD, K. 1968. *SIMULA 67 common base language*. Norsk regnesentral., Oslo, Norway.

- D'ARGENIO, P., KATOEN, J., AND BRINKSMA, E. 1995. A stochastic automata model and its algebraic approach. In *Proceedings of the Fifth Process Algebras and Performance Modelling Workshop*. 1–16. University of Twente, Enschede. CTIT Technical Report 97–14, 1995.
- DECKER, S., MELNIK, S., HARMELEN, F. V., FENSEL, D., KLEIN, M., BROEKSTRA, J., ERDMANN, M., AND HORROCKS, I. 2000. The semantic web: The roles of xml and rdf. *IEEE Internet Computing* 15, 3 (October), 63–74.
- DENNY, M. 2004. Ontology tools survey, revisited. <http://www.xml.com/pub/a/2004/07/14/onto.html>.
- DILL, S., EIRON, N., GIBSON, D., GRUHL, D., GUHA, R., JHINGRAN, A., KANUNGO, T., RAJAGOPALAN, S., TOMKINS, A., TOMLIN, J. A., AND ZIEN, J. Y. 2003. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th International WWW Conference (WWW 2003), Budapest, Hungary*. ACM Press, New York, NY, 178–186.
- DUGAN, J. B., TRIVEDI, K. S., GEIST, R. M., AND NICOLA, V. F. 1984. Extended stochastic petri nets: Applications and analysis. In *PERFORMANCE'84: Models of Comput. System Performance, Proc. of the 10th Int. Symp., Paris*, Gelenbe, E, Ed. Elsevier, Amsterdam, 507–519.
- FISHWICK, P. AND ZEIGLER, B. 1992. A multimodel methodology for qualitative model engineering. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2, 1, 52–81.
- FISHWICK, P. A. 1995. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice-Hall, Inc., Englewood Cliffs, NJ.
- FISHWICK, P. A. 1996a. A taxonomy for simulation modeling based on programming language principles. *IIE Transactions on IE Research* 30, 811–820.
- FISHWICK, P. A. 1996b. Web-based simulation: Some personal observations. In *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, Eds. IEEE Press, Piscataway, NJ, 772–779.
- FISHWICK, P. A. 2002a. RUBE: An XML-based architecture for 3D process modeling and model fusion. In *Proceedings of Enabling Technology for Simulation Science, Part of SPIE Aerosense '02 Conference*. SPIE Press, 330–335.
- FISHWICK, P. A. 2002b. Using XML for simulation modeling. In *Proceedings of the 2002 Winter Simulation Conference*, E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, Eds. IEEE Press, Piscataway, NJ, 616–622.
- FISHWICK, P. A., LEE, J., PARK, M., AND SHIM, H. 2003. RUBE: A customized 2D and 3D modeling framework for simulation. In *Proceedings of the 2003 Winter Simulation Conference*, S. Chuck, P. J. Sanchez, and D. J. Morrice, Eds. IEEE Press, Piscataway, NJ, 755–762.
- FISHWICK, P. A. AND MILLER, J. A. 2004. Ontologies for modeling and simulation: Issues and approaches. In *Proceedings of the 2004 Winter Simulation Conference*, R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, Eds. IEEE Press, Piscataway, NJ, 251–256.
- The Gene Ontology Consortium. 2000. Gene ontology: Tool for the unification of biology. *Natural Genetics* 25, 25–29.
- GENRICH, H. J. AND LAUTENBACH, K. 1973. Synchronisationsgraphen. *Acta Informatica* 2, 143–161.
- GLYNN, P. 1983. On the role of generalized semi-Markov processes in simulation output analysis. In *Proceedings of the 1983 Winter Simulation Conference*, S. Roberts, J. Banks, and B. Schmeiser, Eds. IEEE Press, Piscataway, NJ, 38–42.
- GÖDEL, K. 1930. Die vollständigkeit der axiome des logischen funktionen-kalküls. *Monatshefte für Mathematik und Physik* 37, 349–360.
- GÖDEL, K. 1931. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme. *Monatshefte für Mathematik und Physik* 38, 173–198.
- GRUBER, T. R. 1995. Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal of Human-Computer Studies* 43, 907–928.
- HAARSLEV, V. AND MÖLLER, R. 2001. Description of the racer system and its applications. In *Proceedings of the International Workshop on Description Logics (DL-2001)*. Springer-Verlag, Berlin/Heidelberg, Germany, 132–141.
- ACM Journal Name, Vol. V, No. N, M 20YY.

- HAAS, P. J. 2002. *Stochastic Petri Nets: Modeling, Stability, Simulation*. Springer Verlag, New York.
- HAAS, P. J. AND SHEDLER, G. S. 1987. Regenerative generalized semi-markov processes. *Communications in Statistics-Stochastic Models* 3, 3, 409–438.
- HACK, M. H. T. 1972. Analysis production schemata by Petri nets. M.S. thesis, MIT, Cambridge, Mass.
- HAMMOND, B., SHETH, A. P., AND KOCHUT, K. J. 2002. Semantic enhancement engine: A modular document enhancement platform for semantic applications over heterogeneous content. In *Real World Semantic Web Applications*, V. Kashyap and L. Shklar, Eds. IOS Press, Amsterdam, 29–49.
- HENRIKSON, J. 1981. GPSS-finding the appropriate world view. In *Proceedings of the 1981 Winter Simulation Conference*, T. I. Ören, C. Delfosse, and C. Shub, Eds. IEEE Press, Piscataway, NJ, 505–516.
- HILLS, B. R. AND POOLE, T. G. 1969. A method for simplifying the production of computer simulation models. In *TIMS Tenth American Meeting in Atlanta, GA*. Published in *Flexible Manufacturing Systems in Practice: Design, Analysis, and Simulation* in 1988, Marcel Dekker, Inc., New York, NY.
- HOPKINS, J. F. AND FISHWICK, P. A. 2002. The rube framework for personalized 3-d software visualization. In *Software Visualization. International Seminar, Dagstuhl Castle, Germany, May 20-25, 2001, Revised Lectures*. Lecture Notes in Computer Science, vol. 2269. Springer-Verlag, Berlin, Germany, 368–380.
- HUANG, X. AND MILLER, J. A. 2001. Building a web-based federated simulation system with Jini and XML. In *Proceedings of the 34th Annual Simulation Symposium (ANSS'01)*. IEEE CS Press, Los Alamitos, CA, 143–150.
- INCOSE 2002. SE tools taxonomy – simulation tools. http://www.incose.org/tools/tooltax/simulation_tools.html.
- JACKSON, J. R. 1963. Jobshop-like queueing systems. *Management Science* 10, 1, 131–142.
- JENSEN, K. 1979. *Coloured Petri Nets and the Invariant-Method*. Aarhus University, Computer Science Department, DAIMI PB-104.
- JENSEN, K. 1981. Coloured petri nets and the invariant-method. *Theor. Comp. Science* 14, 317–336.
- JOINES, J. A., JR., K. A. P., AND ROBERTS, S. D. 1992. Object-oriented modeling and simulation with C++. In *Proceedings of the 1992 Winter Simulation Conference*, J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, Eds. IEEE Press, Piscataway, NJ, 145–153.
- KANT, K. 1992. *Introduction to computer system performance evaluation*. McGraw-Hill, Inc. with contributions by M. M. Srinivasan.
- KIM, T., LEE, J., AND FISHWICK, P. 2002. A two-stage modeling and simulation process for web-based modeling and simulation. *ACM Transactions on Modeling and Computer Simulation* 12, 3 (July), 230–248.
- KIM, T. G. 1995. DEVS formalism: Reusable model specification in an object-oriented framework. *International Journal in Computer Simulation* 5, 4, 397–416.
- KIVIAT, P. J. 1969. Digital computer simulation: Computer programming languages. RAND Memo. RM-5883-PR.
- KOLMOGOROV, A. N. 1933. On the theory of continuous random processes. In *Selected works of A.N. Kolmogorov*, A. N. Shirayev, Ed. Vol. II. Kluwer Acad. Publ., 1992, 62–108.
- KREUTZER, W. 1986. *System Simulation: programming styles and languages*. Addison-Wesley, Sydney.
- LACKNER, M. R. 1962. Toward a general simulation capability. In *Proceedings of the AFIPS Spring Joint Computer Conference*. AFIPS Press, Montvale, N.J., 1–14.
- LACLAUSTRA, J. C. 1990. Performance bounds for synchronized queueing networks. Ph.D. thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain. Research Report GISI-RR-90-20.
- LACY, L. 2001. Semantic web applications for modeling and simulation. <http://www.daml.org/2001/07/dmso-applications/semantic-web-071101.ppt>.

- LEVY, P. 1954. Processus Semi-Markoviens. In *Proceedings of International Congress of Mathematics*. Amsterdam.
- MARKOV, A. A. 1913. Primer statisticheskogo issledovanija nad tekstem ‘Evgenija Onegina’ illjustrirujuschij svjaz’ ispytanij v tsep (An example of statistical study on the text of ‘Eugene Onegin’ illustrating the linking of events to a chain). *Izvestija Imp. Akademii nauk, serija VI 3*, 153–162.
- MCCULLOCH, W. S. AND PITTS, W. H. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5*, 115–133.
- MCGUINNESS, D. L. AND VAN HARMELEN, F. 2003. OWL Web Ontology Language overview. <http://www.w3.org/TR/owl-features>.
- MEALY, G. H. 1955. A method for synthesizing sequential circuits. *Bell System Tech. J. 34*, 1045–1079.
- MILLER, J. A. AND BARAMIDZE, G. T. 2005. Simulation and the semantic web. In *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, Eds. IEEE Press, Piscataway, NJ, 2371–2377.
- MILLER, J. A., BARAMIDZE, G. T., FISHWICK, P. A., AND SHETH, A. P. 2004. Investigating ontologies for simulation modeling. In *Proceedings of the 37th Annual Simulation Symposium*. IEEE CS Press, Los Alamitos, CA, 55–71.
- MILLER, J. A., FISHWICK, P. A., TAYLOR, S. J. E., BENJAMIN, P., AND SZYMANSKI, B. 2001. Research and commercial opportunities in web-based simulation. *Simulation Practice and Theory. Special Issue on Web-Based Simulation 9*, 1-2 (October), 55–72.
- MILLER, J. A., POTTER, W. D., KOCHUT, K. J., AND RAMESH, D. 1997. *Object-Oriented Simulation*. IEEE Press, Piscataway, NJ, Chapter Object-Oriented Simulation Languages and Environments: A Four-Level Architecture, 53–88.
- MILLER, J. A., SEILA, A. F., AND TAO, J. 2000. Finding a substrate for federated components on the web. In *Proceedings of the 2000 Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 1849–1854.
- MILLER, J. A., SHETH, A. P., AND KOCHUT, K. J. 1997. Perspectives in modeling: Simulation, database and workflow. In *Proceedings of the 16th International Conference on Conceptual Modeling: Preconference Symposium*, P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Lecture Notes in Computer Science. Springer-Verlag, Berlin/Heidelberg, Germany, 154–167.
- MOLLOY, M. K. 1981. On the integration of delay and throughput measures in distributed processing models. Ph.D. thesis, UCLA.
- MONET 2004. The MONET Ontologies. http://monet.nag.co.uk/cocoon/monet/publicdocs/monet_ontologies.html.
- MOORE, E. F. 1956. Gedanken-experiments on sequential machines. In *Automata Studies*, C. E. Shannon and J. McCarthy, Eds. Number 341 in Annals of Mathematics Studies. Princeton U. Press, Princeton, NJ, 129–153.
- NAIR, R., MILLER, J. A., AND ZHANG, Z. 1996. A Java-based query driven simulation environment. In *Proceedings of the 1996 Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 786–793.
- NANCE, R. E. 1993. A history of discrete event simulation programming languages. In *Proceedings of the Second ACM SIGPLAN History of Programming Languages Conference*. Reprinted in ACM SIGPLAN Notices, Vol. 28, No. 3. ACM Press, New York, NY, 149–175.
- ÖREN, T. I. 2005. Toward the body of knowledge of modeling and simulation (invited tutorial). In *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*. Orlando, FL, 1–19.
- PAGE, JR, E. H. 1994. Simulation modeling methodology: Principles and etiology of decision support. Ph.D. thesis, Virginia Tech. <http://www.thesimguy.com/ernie/papers/unref/dissert/dw.html>.
- PAGE, JR, E. H., FISHWICK, P. A., HEALY, K. J., NANCE, R. E., AND PAUL, R. J. 2000. Web-based simulations: Revolution or evolution. *ACM Transactions on Modeling and Computer Simulation 10*, 1, 3–17.
- ACM Journal Name, Vol. V, No. N, M 20YY.

- PASQUINI, R. AND REGO, V. 1998. Efficient process interaction with threads in parallel discrete event simulation. In *Proceedings of the 1998 Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 451–458.
- PEGDEN, C. D., SHANNON, R. E., AND SADOWSKI, R. P. 1990. *Introduction to Simulation Using SIMAN*. McGraw-Hill, NY.
- PETERSON, J. L. 1977. Petri Nets. *ACM Computing Surveys* 9, 223–252.
- PETRI, C. A. 1962. Fundamentals of a theory of asynchronous information flow. In *Proceedings of IFIP Congress 62*. North-Holland Pub. Co, Amsterdam, 386–390.
- PIDD, M. 1992. *Computer Simulation in Management Science*, 3 ed. Wiley, Chichester, England.
- PRITSKER, A. A. B. 1979. *Introduction to Simulation with SLAM*. Wiley, New York, NY.
- RAMCHANDANI, C. 1974. Analysis of asynchronous concurrent systems by petri nets. Tech. rep., MIT, Laboratory of Computer Science, Cambridge, Mass. February.
- SARJOUGHIAN, H. S., CELLIER, F. E., AND ZEIGLER, B. P., Eds. 2001. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler*. Springer-Verlag, Berlin/Heidelberg.
- SCHRIBER, T. J. 1974. *Simulation Using GPSS*. Wiley, NY.
- SCHRIBER, T. J. AND BRUNNER, D. T. 1996. Inside simulation software: How it works and why it matters. In *Proceedings of the 1996 Winter Simulation Conference*. IEEE Press, Piscataway, NJ, 23–30.
- SCHRUBEN, L. 1983. Simulation modeling with event graphs. *Communications of the ACM* 26, 957–963.
- SCHRUBEN, L. AND ROEDER, T. 2003. Fast simulations of large-scale highly-congested systems. *Simulation: Transactions of the Society for Modeling and Simulation International* 79, 3, 1–11.
- SCHRUBEN, L. AND YÜCESAN, E. 1989. Simulation graph duality: A world view transformation for simple queueing models. In *Proceedings of the 21st conference on Winter simulation*, E. MacNair, K. Musselman, and P. Heidelberger, Eds. IEEE Press, Piscataway, NJ, 738–745.
- SCHWETMAN, H. 1995. Object-oriented simulation modeling with c++/csim. In *Proceedings of the 1995 Winter Simulation Conference*, C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, Eds. IEEE Press, Piscataway, NJ, 529–533.
- SHAW, O., KOELMANS, A., STEGGLES, L., AND WIPAT, A. 2004. Applying Petri Nets to systems biology using XML technologies. Tech. Rep. CS-TR: 953, University of Newcastle upon Tyne.
- SHETH, A., BERTRAM, C., AVANT, D., HAMMOND, B., AND K. KOCHUT, Y. W. 2002. Managing semantic content for the web. *IEEE Internet Computing* 6, 4 (July/August), 80–87.
- SILVER, G. A., LACY, L., AND MILLER, J. A. 2006. Ontology based representations of simulation models following the process interaction world view. In *Proceedings of the 2006 Winter Simulation Conference*. (To appear).
- SMITH, W. L. 1955. Regenerative stochastic processes. In *Proc. of Roy. Soc. London, Ser. A*. Vol. 232. London, 6–31.
- TOCHER, K. D. 1963. *The Art of Simulation*. Van Nostrand Company, Princeton, NJ.
- VANGHELuwe, H. L. 2000. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design*. IEEE CS Press, Los Alamitos, CA, 129–134.
- VANGHELuwe, H. L., DE LARA, J., AND MOSTERMAN, P. J. 2002. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 Conference*, F. Barros and N. Giambiasi, Eds. SCS International, San Diego, CA, 9–20.
- YÜCESAN, E. AND SCHRUBEN, L. 1992. Structural and behavioral equivalence of simulation models. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2, 1, 82–103.
- ZEIGLER, B. P. 1976. *Theory of Modelling and Simulation*. Wiley Interscience, New York, NY.
- ZEIGLER, B. P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, Boston, Mass.
- ZEIGLER, B. P., PRAEHOFER, H., AND KIM, T. G. 2000. *Theory of Modelling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, San Diego, CA.

Appendix A- Lower Layers of the Semantic Web

Resource/Data Layer - Extensible Markup Languages (XML)

The initial Web was built using the Hyper-Text Markup Language (HTML) and the Hyper-Text Transport Protocol (HTTP). The Extensible Markup Language (XML) later became popular since it was like HTML except that its tags are more meaningful than those used for HTML, which are basically for formatting.

Although the Resource/Data Layer consists of multiple resource types, XML is currently one of the most widely used markup languages for structured documents. XML features extensible rather than predefined tags for self description of the data and it can be syntactically validated. It is particularly useful for transmitting richly structured data across platforms.

Putting Web data into XML documents makes the data more understandable and processable and is indeed a first step to making a Semantic Web.

Recently, the use of Extensible Markup Language (XML) in modeling and simulation has been investigated as a medium for communication between simulation components [Miller et al. 2000; Huang and Miller 2001] and in specifying simulation models [Fishwick 2002b; Kim et al. 2002]. This increases interoperability since XML is a language which is universally supported and platform independent.

Need for a markup language for specifying models in a programming language independent way led to the development of RUBE-XML [Hopkins and Fishwick 2002; Fishwick 2002a; Kim et al. 2002; Fishwick et al. 2003]. RUBE is an implementation using the Blender 3D interface, to explore the custom construction of dynamic models in 3D. The dynamic models include finite state automata, Markov chains, Petri nets, System Dynamic models, and Ordinary Differential Equations (ODE). Most of these models have an XML schema associated with them, except for the ODE, where MathML is used.

The RUBE interface works as follows. A user launches Blender, and is presented with a Python interface that is organized by “theme”. A theme suggests the type of visual icons to be used. For example, a “factory theme” uses a machine object for a state and a belt transport for a state transition. The user picks the objects, which appear on the canvas, and can be further manipulated to achieve an optimal layout or design. Then, the user can simulate the system, which involves the model structure being translated into two XML languages: MXL and DXL. MXL (Multimodeling eXchange Language) encodes the high level multimodel syntax, where a multimodel may be composed of several heterogeneous abstractions. DXL (Dynamics eXchange Language) encodes the multimodel as a single, homogeneous block-based circuit-like language. DXL blocks are further translated into target code such as Java or Python.

The ontology aspects of RUBE are fairly new, and expanding, beginning with a RUBE-based ontology editor that uses OWL to encode the semantic network, containing links that connect the scene with its dynamic and geometric models.

In the long run, the community needs to develop an agreed upon XML markup language (or languages) for the domain (e.g., a Modeling and Simulation Markup Language (MSML)). As we have argued, such a markup language should be anchored in an ontology, so that its tags are precisely defined (especially, in a machine processable fashion – not just in natural language).

As discussed in the Introduction, XML documents with their use of more meaningful tags carry some semantics to humans. However, XML is lacking in several ways: it does not support inheritance, named relationships, inferencing and interoperability [Decker et al. 2000]. These are all useful to providing machine processable semantics. Because of these limitations Berners-Lee and others proposed the upper layers of the Semantic Web [Berners-Lee et al. 2001].

Meta-Data Layer - Resource Description Framework (RDF)

The Web is made up of documents (e.g., in HTML, XML, PDF, etc.) as well as applications and services (e.g., servlets, and increasingly implemented as Web services). In general, these are referred to as Web resources which are uniquely identified by a Uniform Resource Identifier (URI). The Resource Description Framework (RDF) is the primary language used to describe Web resources (although there exist other more special-purpose languages such as the Web Services Description Language (WSDL)). These descriptions may be further abstracted for more meaningful representation as meta-data (or data about Web resources). A key advantage of RDF is that it allows the descriptions to be processed and indexed, making the search for resources more precise.

Ontologies typically include some amount of instance data, much of which could be RDF meta-data. Some of this could be entered by hand, but more useful would be the use of extractors and crawlers to keep the instance data automatically updated, as demonstrated successfully by the Semagix [Hammond et al. 2002] and IBM [Dill et al. 2003].

Appendix B - State-Oriented Formalisms

Generalized Semi-Markov Processes

For simplicity, we have initially treated the three driving functions as deterministic functions. The simplest way to make these functions stochastic is to define associated probability mass functions (pmf's) for them. However, since c may not be discrete, we need to define an associated distribution function (df) F_c for it.

$$\text{Activation Function - } p_a : S \times E \rightarrow [0, 1],$$

where $p_a(s_i, e) = Pr\{a(s_i, e) = 1\}$ is the probability that state s_i enables event e .

$$\text{Clock Function - } c : S \times E \rightarrow T \text{ according to } F_c,$$

where the clock distribution function $F_c(s_i, e; \tau) = Pr\{c(s_i, e) \leq \tau\}$ is the probability that from state s_i the clock for event e will be set to at most τ time units.

$$\text{State Transition (or Delta) Function - } p_d : S \times E \times S \rightarrow [0, 1],$$

where $p_d(s_i, e, s_j) = Pr\{d(s_i, e) = s_j\}$ is the probability that from state s_i when event e fires, there will be a transition to state s_j .

Note, if the activation function a is deterministic, then it is possible to derive a from the transition function d .

The mechanics of the model work as follows: Upon entering state s_i , each function is evaluated in turn. All of this occurs without any advance of the current time t . From the set of candidate events e_k , let the state holding time τ^* to be the minimum of all the event clocks,

$$\tau^* = \min\{c(s_i, e_k) : p_a(s_i, e_k) \geq rn_k\},$$

where $rn_k \in (0, 1]$ are independent and identically distributed random numbers (Uniform(0, 1]). If a is deterministic simply replace rn_k with 1. Determine the set of imminent events to be those that will occur when the time t advances τ^* units forward, i.e., when $t = t + \tau^*$,

$$e^* = \{e_k : c(s_i, e_k) = \tau^*\}.$$

If e^* is not a singleton set, there are several ways to proceed: randomly pick one of the events, prioritize the events and take the highest priority, or provide a transition function that can account for multiple simultaneous events, i.e., $d : S \times 2^E \rightarrow S$ as done by [Haas and Shedler 1987] in their definition of GSMP. A simpler formulation of GSMP is by [Glynn 1983].

What happens to other events (those that have yet to fire)? GSMP will keep such events that are still compatible with the next state s_j . This is common in discrete-event simulators which keep such events on a future event list. However, to yield analytic solution, this non-Markovian behavior may be eliminated, so that such events disappear. For a Glynn formulation of GSMP, this restriction produces a Semi-Markov Process (SMP).

Given a single imminent event e , when time advances to its occurrence time, the event fires and the next state s_j is determined by evaluating the state transition function,

$$s_j = d(s_i, e).$$

Note, since nothing happens between event occurrences, discrete-event simulators typically advance time from event occurrences to event occurrences in discontinuous jumps.

For deterministic models, these functions (a , c , d) may be, for example, implemented via table lookup using three matrices:

$$\begin{aligned} a(s_i, e) &= \mathbf{A}[s_i, e] = 1(s_i \text{ enables } e), \text{ an enablement indicator,} \\ c(s_i, e) &= \mathbf{C}[s_i, e] = \tau, \text{ a time increment,} \\ d(s_i, e) &= \mathbf{D}[s_i, e] = s_j, \text{ a state,} \end{aligned}$$

where \mathbf{A} is the event activation matrix, \mathbf{C} is the clock matrix and \mathbf{D} is the state transition matrix.

Since there are three functions (a , c , d), theoretically, there exist seven types of stochastic models that could be defined (eight ways to assign deterministic vs. stochastic to the functions). Note also, that these definitions can be further generalized to, for example, account for simultaneous events [Haas and Shedler 1987] and, if need be, for other extra features.

Semi-Markov Processes

If we restrict our attention to models in which state transitions are only dependent on the current state and not past history (Markov property), we enter the class of Semi-Markov Processes (SMP). Further, if the time to the next event is independent of the time that has elapsed since the last event (memoryless property), we enter the class of Markov Chains.

In order to develop an SMP model, one needs to specify the probabilities using a transition probability matrix \mathbf{P} ,

$$p_d(s_i, s_j) = \mathbf{P}_d[s_i, s_j].$$

In addition, a holding time distribution function F_c needs to be specified for each state s_i ,

$$F_c(s_i; \tau) = Pr\{\tau^* \leq \tau\},$$

which is the probability that upon entering state s_i , the model stays there for at most τ time units.

A Continuous Time Markov Chain (CTMC) model is a special type of SMP model, in which the state holding times follow exponential distributions,

$$F_c(s_i; \tau) = 1 - e^{-\lambda_i \tau} \text{ where } \tau \in \mathbb{R}^+.$$

From a GSMP point of view, the model uses Poisson clocks.

A Discrete Time Markov Chain (DTMC) model is also a special type of SMP model, in which the state holding times follow geometric distributions,

$$F_c(s_i; \tau) = 1 - \{\mathbf{P}_d[s_i, s_i]\}^\tau, \text{ where } \tau \in \mathbb{Z}^+.$$

These two distributions (exponential and geometric) are special in that they are the only ones to exhibit the memoryless property, in which the remaining holding time in a state is independent of how long the model has been in that state.

Stochastic Timed Automata

A simple restriction to Glynn's GSMP definition will yield a Stochastic Timed Automata (STA). Usually, one thinks of automata (state machines) as being driven by external events (or input if you like), so the clock function would typically be independent of state, i.e.,

$$\begin{aligned} c : E \rightarrow T \text{ according to } F_c, \\ F_c(e; \tau) = Pr\{c(e) \leq \tau\}. \end{aligned}$$

One can think of this mechanism as clocking in the next event (or symbol from the alphabet). Basically, the activation function a and transition function d remain the same with a playing more of a checking role,

$$\begin{aligned} a : S \times E \rightarrow \{0, 1\}, \\ d : S \times E \rightarrow S, \\ p_d(s_i, e, s_j) = Pr\{d(s_i, e) = s_j\}. \end{aligned}$$

One could also imagine driving the model from an input stream,

$$\{e_{kl} : k \in E \text{ and } l \in \mathbb{Z}^+\},$$

where e_{kl} is the l^{th} event/input in the stream and is of type k . If e_{kl} is the next event (input symbol) then the functions could be applied as follows:

$$\begin{aligned} \tau = \mathbf{if } a(s_i, e_{kl}) \mathbf{ then } c(e_{kl}) \mathbf{ else throw exception,} \\ s_j = d(s_i, e_{kl}) \text{ with probability } p_d(s_i, e_{kl}, s_j). \end{aligned}$$

Rather than have very long (or infinite) input streams, the following alternative interpretation may be preferred. Imagine that there is a clock timer for every possible type of event. It will run until it reaches zero causing the event to fire (i.e., the event shows up as the next input). The clocks then work as event/input generators.

Just like GSMP, the clock and transition functions for STA are stochastic. If we make both of them deterministic, we will end up with a Timed Automata (TA). Again, other possibilities exist, but we believe them to be of less practical use. The three functions now become the following:

$$\begin{aligned} a(s_i, e) &= \mathbf{A}[s_i, e] = 1(s_i \text{ enables } e), \text{ an input checker,} \\ c(e) &= \mathbf{c}[e] = \tau, \text{ a time increment,} \\ d(s_i, e) &= \mathbf{D}[s_i, e] = s_j, \text{ a state,} \end{aligned}$$

where \mathbf{A} is the event activation matrix, \mathbf{c} is the clock vector and \mathbf{D} is the state transition matrix.

Finally, if we measure time by counting event occurrences (or equivalently the number of input symbols consumed), then we may define the clock setting for event e to be a positive integer indicating the number of event occurrences until event e reoccurs,

$$c(e) = \mathbf{c}[e] = \text{event reoccurrence time.}$$

As time has been reduced to a counting role, one could even take it out of the model. We do not do this for the sake of uniformity though. TA without time (or time reduced to a counting role) are referred to as State Automata (SA). If the state space is finite, these are said to be Finite State Automata (FSA) of the deterministic variety, i.e., Deterministic Finite Automata (DFA).⁵

Appendix C - Activity-Oriented Formalisms

Extended Stochastic Petri Nets

In queuing networks, simple interactions between entities can be modeled (e.g., entities competing with each other for service), but complex interactions/synchronizations can not be modeled in a direct way. A more general modeling technique that supports complex interactions of entities/tokens is provided by Petri Nets (PN).

In Petri Net literature, the state $\mathbf{n}(t)$ is referred to as a marking (a vector indicating the number of tokens in each place). A transition causes the marking to change by moving tokens from place to place (or possibly by creating or destroying tokens). Each transition has an implicit event when it is enabled.

In the Timed Petri Nets formalism transitions are timed. That means that once a transition is enabled, after an interval of time, known as the firing time, an end-event occurs that transforms the state. Firing time essentially represents the time taken by the corresponding activity (assuming non-atomic firing semantics [Kant 1992]). Thus, Petri Net transitions may be associated with activities described in the model.

⁵At this point in the development of the DeMO ontology we are not considering nondeterministic automata such as Nondeterministic Finite Automata (NFA).

Here we consider *Extended Stochastic Petri Nets (ESPN)* [Dugan et al. 1984] as the top formalism for Activity-Oriented models. The basic sets used in this formalism are (P, A, T) (where, unlike the traditional Petri Net notation, A corresponds to a set of transitions, and T to a time set), but the driving functions are modified. The activation function together with the transition function are transformed into input and output incidence functions:

$$\begin{aligned} d_i &: P \times A \rightarrow \mathbb{Z}^+, \\ d_o &: A \times P \rightarrow \mathbb{Z}^+. \end{aligned}$$

Here $d_i(p, a) = n$ means that a transition (activity) a can be enabled only if an input place p has at least n tokens and firing this transition will result in consuming these n tokens. Similarly $d_o(a, p) = m$ means that once an activity a is finished (transition fires), m tokens will be placed into an output place p . In ESPN models, an additional feature is introduced to allow probabilistic routing: *probabilistic arcs* from transitions to a set of output places. The output incidence function is then modified to reflect this:

$$d_o : A \times P \rightarrow \mathbb{Z}^+ \times [0, 1].$$

Without probabilistic routing the model dynamics can be captured by defining an $|A|$ by $|P|$ matrix D called the incidence matrix,

$$D[a, p] = \text{number of tokens added to place } p \text{ when the transition } a \text{ fires.}$$

Note, if the value is negative, the place loses tokens.

Given the current marking (or state) $\mathbf{n}(t)$ and the transition to fire a_k , the new marking simply adds the product of the incidence matrix with the following indicator column vector:

$$\begin{aligned} \mathbf{1}(a_j, a_k) &= \text{if } j = k \text{ then } 1 \text{ else } 0, \\ \mathbf{n}(t+1) &= d(\mathbf{n}(t), a_k) = \mathbf{n}(t) + \mathbf{D}\mathbf{1}(a_j, a_k). \end{aligned}$$

The clock function determines the duration of an activity (transition),

$$c : A \rightarrow T.$$

For generality, usually both timed and immediate (untimed) transitions are considered:

$$\begin{aligned} A_1 &\subseteq A - \text{timed transitions,} \\ A_2 &\subseteq A - \text{immediate transitions,} \\ A_1 \cap A_2 &= \emptyset, A_1 \cup A_2 = A. \end{aligned}$$

Stochastic features of Extended Stochastic Petri Nets (ESPN) include stochastic clocks given as arbitrary distributions as well as random selections from simultaneously occurring sets of transitions (events). Additional features, such as inhibitor arcs, are also usually considered, but are skipped here for simplicity.

Restrictions on the clock function c lead to many popular types of Petri Nets.

—Poisson Clocks, Timed and Untimed Transitions, no probabilistic arcs.

Generalized Stochastic Petri Nets (GSPN) allow a mixture of both timed and untimed (immediate) transitions where the firing times for timed transitions are given by exponential distribution functions. No probabilistic routing is allowed.

- Poisson Clocks, Timed Transitions Only, no probabilistic arcs. If the transition firing times are given by exponential distribution function, we have a *Stochastic Petri Net (SPN)*.
- Deterministic Clocks. If the transition firing times are deterministic, we have a *Timed Petri Net (TPN)*.
- Untimed Transitions Only. If all transitions are immediate (no time delay), we have a classical *Petri Net (PN)*.

One way to view the situation is to collect all the places coming into a transition and imagine them as forming a queue. This will yield a Queuing Network (QN) view of the net. Such a net must allow probabilistic branching/routing. For a more complete discussion of the similarities and differences between QN and SPN see [Laclaustra 1990].

Activity Cycle Diagrams (ACD) [Tocher 1963; Hills and Poole 1969] can also be viewed as restricted versions of Stochastic Petri Nets (SPN). Timed transitions become activities and places become wait states (or queues). Topological restrictions are imposed to make sure that the queues and activities alternate and the activity cycles are closed.

Appendix D - Object-Oriented Simulation

Object-Oriented Simulation (OOS) began quietly with Simula-67 in the 1960's [Dahl et al. 1968]. As Smalltalk and more so C++ made object-oriented programming popular in the 1980's, the 1990's saw the reemergence of OOS in the 1990's with numerous publications (e.g., [Joines et al. 1992; Zeigler 1990; Schwetman 1995; Cubert et al. 1997; Miller et al. 1997]). Unlike the traditional world-views, OOS also relates to larger issues, such as model reuse, in the design of complex simulation models. As such, a simple self-contained formalism is not as straightforward as it is for the traditional world-views. Although there does not exist a clear and well-agreed upon definition of object-oriented simulation, in general, object-oriented programming can be used with any of the world-views. In particular, Kreutzer [1986] discusses how object-oriented programming techniques can be applied to the process-interaction world-view.

From a more formal point of view, Zeigler [1990] and Kim [Kim 1995] discuss how the object-oriented paradigm fits with modular model specifications (which itself is based on mathematical system theory). Modular system models can be adjusted to support encapsulation. Still, modular models have additional requirements that go beyond the object-oriented paradigm, such as providing a time base.

The key benefits of OOS come into play with the inheritance and aggregation hierarchies, which allow components or models to be combined or coupled (e.g., coupled DEVS models). Although developing a comprehensive ontology for OOS would be very useful, a full coverage should include topics that are closely related, such as agent-based simulation, component-based simulation, multi-modeling and modular and hierarchical modeling. As this is a substantial undertaking, we leave a thorough treatment of OOS as future work. This is a natural division, since the current work focuses on the basics of modeling, while the future work would address how multiple models are combined into complex simulation models.

Received Month Year; revised Month Year; accepted Month Year