Ontologies for Modeling and Simulation: Initial Framework

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

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

August 2, 2004

Abstract

[JOHN - redo this last...]

Many fields have or are developing ontologies for their subdomains. The Gene Ontology (GO) is now considered to be a great success in biology, a field that has already developed several extensive ontologies. Similar advantages could accrue to the simulation and modeling community. Ontologies provide a way to establish common vocabularies and capture domain knowledge for organizing the domain with a community wide agreement or with the context of agreement between leading domain experts. They can the be used to provide significantly improved (semantic) search and browsing, integration of heterogeneous information sources, and improved analytics and knowledge discovery capabilities. Such knowledge can be used to establish common vocabularies, nomenclatures and taxonomies with links to detailed information sources. 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.

1 Introduction

One of the ways in which a field matures is through the generation of taxonomies and more recently ontologies. These reflect how words, phrases and concepts representing domain semantics can be grouped and interrelated. Simulation is not that different from other fields in computing, where taxonomies exist on paper, but it is still rather difficult to connect what one research group is doing in Activity Diagrams (Birtwistle, 1979), for instance, with what another group is doing in Event Graphs (Schruben, 1983). How do the components of the models connect? Up until recently, this was somewhat of an academic question since one might relate terms and phrases on paper, but this seems less than complete. What is needed is

an efficient way of effecting links between concepts so that these linkages can be subsequently employed in databases, query engines, and human-computer interaction models. In addition, such organization of knowledge within a field helps increase the interoperability, integration and reuse of simulation artifacts. In the simulation and modeling domain, such artifacts include libraries, components, simulators, animators, simulation tools, and models. Associated with these artifacts are documents such as manuals, tutorials, papers and result repositories.

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

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

"discrete-event simulation" + "process-interaction"

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

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

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

Much of this work involves the use of ontologies (Gruber, 1995) to define terms or concepts in certain domains (narrow and deep ontologies). For a particular domain, types of things are defined as classes, which have properties and relationships. The meaning of a concept is typically given in natural language. Meaning is also captured via the class's position within a taxonomy (`subclass-of/is-a` hierarchy) as well as its properties, relationships and restrictions. Unlike schemas or data models which are application or data, access oriented (i.e., the point is to get the job at hand done) ontologies are meant to define a domain and to be shared and used by many (Denny, 2002). In a way, it provides semantics by agreement (we agree that this term should mean this). Most useful ontologies are therefore created by expert groups (e.g., GO at *www.geneontology.org*). See the table below for a list of ontologies in scientific domains.

Ontology	**Name**	**Domain**
EngMath	Engineering Math	Mathematics
EHEP	Exp. High-Energy Physics	Physics
OntoNova	ONTOlogy-based NOVel Q&A	Chemistry
GO	Gene Ontology	Genetics
MDEG	Microarray Gene Expression Data	Biology
AstroGrid	Astronomy Grid	Astronomy

Table 1: Ontologies for Scientific Domains

There are also efforts to define broad and shallow ontologies, called upper ontologies, such as the Suggested Upper Merged Ontology (SUMO) at *ontology.teknowledge.com* or the Standard Upper Ontology (SUO) at *suo.ieee.org*. Proponents indicate that this facilitates the use of multiple ontologies and greater integration. Another school of thought views monolithic upper ontologies as too ambitious and inflexible. Fixing an upper ontology that numerous domain ontologies (and therefore applications and data models) are dependent upon may lead to a maintenance nightmare. There has been much prior work on Web-Based Simulation, which takes advantage of Web technologies and availability to enhance simulation and modeling. We have discussed the general problem of linking simulation terminology, and how Web-Based Simulation has led us into XML-based approaches. A new wave of technology is emerging that should have an even greater impact on modeling and simulation, namely, the Semantic Web and Web Services (Silver et al., 2002). One of the most important contributions of the Semantic Web is the creation of ontologies for specific domains, which is the focus of this paper.

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. Section 4 discusses existing discrete-event modeling frameworks and builds the backbone taxonomy for our proposed ontology expressed semi-formally in a combination of mathematics and natural language. Section 5 discusses the use and development of ontologies for modeling and simulation. We present strategies and issues that came up as we developed a prototype ontology called DeMO (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. 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 (KIF) 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. KIF 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 checked (FOL is sound), but it is not possible to generate all true statements (FOL is incomplete). Taken together FOL is said to be semi-decidable (REF???). For the Semantic Web, some researcher feel that a decidable language would be more appropriate, since limiting the languages 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 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 subset of RuleML (Horne-like rules)(*http://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 there expressiveness and usage.)

2.1 Structure of the Semantic Web

Before going into more details about ontologies for the Semantic Web, we will briefly overview the architecture (REF) 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 RDF and RDF-S 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 make deductions about the facts and relations not explicitly stated in the ontology using rules. 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, 1999).

Each layer of the Semantic Web architecture (or stack) has an associated Web Language, as indicated in the table 2. In the subsections below, we overview the first four layers/languages and mention some of their uses in modeling and simulation. The upper layers are still very much future work, so we do not discuss them further.

2.2 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.

Layer	Principal Language	Name (URL)
Resource/Data	XML, XML-S	eXtensible Markup Language, XML Schema *(www.w3c.org/XML, www.w3c.org/XML/Schema)*
Meta-Data	RDF, RDF-S	Resource Description Framework, RDF Schema *(www.w3c.org/RDF)*
Ontology	OWL	Web Ontology Langauge *(www.w3.org/2004/OWL/)*
Logic	SWRL	Semantic Web Rule Language *(www.w3.org/Submission/SWRL/)*
Proof/Trust	*Future Work*	

Table 2: Layers of the Semantic Web Architecture

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 eXtensible Markup Language (XML) in modeling and simulation been investigated as a medium for communication between simulation components (Miller et al., 2000; Huang and Miller, 2001). and in specifying simulation models (Fishwick, 2002; Kim et al., 2002). This increases the interoperability since XML is universally supported and is programming language and platform independent. Need for a markup language for specifying models in a language independent way lead to the development of RUBE-XML (REF).

[PAUL - update and expand this subsection]

In the long run, the community needs to develop an agreed upon XML markup language for the domain (e.g., a Modeling and Simulation Markup Language (MSML)). As we later argue later, 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).

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; hence, we more ahead in the Semantic Web.

2.3 Meta-Data Layer - Resource Description Framework (RDF)

The Web is made up documents (e.g., in HTML, XML, PDF, etc.) as well as services (e.g., servlets or Web servers). 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)). If we view the Web resources as data (which they primarily are), then these descriptions may be viewed as meta-data (or data about data). A key advantage of RDF is that it allows the descriptions to be processed and indexed, making the search for resources more precise. In addition, RDF

meta-data can be built using extraction techniques as demonstrated successfully by Semagix (Sheth, ZZZZ).

Ontologies typically include some amount of instance data, much of which could be RDF meta-data. Some of this could be enterred by hand, but more useful would be the use of extractors and crawlers to keep the instance data atomatically updated. We will discuss this in more detail in a later section.

2.4 Ontology Layer - Web Ontology Language (OWL)

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

Compared to RDF-S, OWL adds more feature for describing properties and classes such as relations between classes (e.g., disjointness), richer typing of properties, cardinality of properties, and characteristics of properties (e.g., symmetry, transitivity). As a revision of the DAML+OIL ontology language, OWL is built upon the lessons learned from the design and application of DAML+OIL (McGuinness and van Harmelen, 2003).

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

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

2.5 Logic Layer - Semantic Web Rule Language (SWRL)

There are some recent proposals for languages at the Logic Layer including the Semantic Web Rule Language (SWRL). SWRL is one proposed language for this layer of the Semantic Web. It extends OWL DL with the ability to write rules using a subset of RuleML. It permits Horn logic rules to be added to OWL descriptions. This allows one to build up more complex predicates which can be used for more precise definitions of concepts. Since the logic layer is in its preliminary stages of development, we plan to enhance DeMO using this layer in our future work.

3 Taxonomies for Discrete Event Modeling

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

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). Discrete-event simulation models are very popular for modeling many types of real-world systems such as banks, hospitals and transportations systems, so we will focus our attention on them.

Within discrete-event modeling there are multiple ways to construct a taxonomy.

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

For discrete-event simulation modeling there are three main world views (simulation modeling paradigms) that can be used: event-scheduling, activity-scanning and process-interaction. 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 five 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 a future event-list. Time is advanced by jumping to the timestamp of the earliest event (imminent event) on the list and processing that event.

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

Activity-Scanning (AS). 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 (ACD) are graphs with two types of nodes (bipartite graphs), activities and wait states, where the arcs connect either activities to wait states or wait states to activities (Tocker, 1963; Pidd, 1992). These diagrams depict the life-cycles of interacting entities flowing through a system.

- *Petri Nets (PN).* Petri Nets (PN) are graphs with two types of nodes (bipartite graphs), transitions and places, where the arcs connect either transitions to places or places to transitions (Petri, 1962; Peterson, 1977). A 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, 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 two other popular approaches: (i) state transition models (e.g., Markov models) and (ii) DEVS models (REF).

Most of the simulation taxonomies are based on execution characteristics of models – scheduling events, scanning for activities, interacting processes. Fishwick (Fishwick, 1995; Fishwick, 1996b) 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 (GSMP) subsume Semi-Markov Processes (SMP), which subsume Markov Chains (MC)). 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.

Surveying existing taxonomies is a useful way to begin the development of an ontology, hence we give a very brief review. Models can be classified based on various characteristics: static vs. dynamic, time-varying vs. time invariant, continuous state vs. discrete state, time-driven vs. event-driven, descriptive vs. prescriptive, analytic vs. numeric (Zeigler, 1976; Page, 1994). In particular, discrete event simulation models can be defined as abstract, dynamic, descriptive, numerical models (Page, 1994). (Cassandras and Lafortune, 1999) define discrete-event 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 model, probabilistic vs. stochastic, and autonomous (no input) vs. nonautonomous models (Page, 1994). (Nance, 1993) divides simulation models into three main classes: Monte-Carlo, continuous, and discrete event. The International Council on Systems Engineering maintains and updates a broad taxonomy for software tools including simulation tools (INCOSE, 2002). (Schruben and Roeder, 2003) propose a new way of organizing the highest levels a modeling taxonomy for discrete event simulation based on a dichotomy between resident entity cycle modeling and transient entity flow modeling as opposed to the classical three worldviews. As mentioned before, an ontology is more than a taxonomy, and indeed may implicitly capture multiple taxonomies.

In the next two sections, we will show and discuss some of the details of the Discrete-event Modeling Ontology (DeMO). DeMO is an ontology for discrete-event modeling (DEM) (system dynamics for discrete-event systems (DES)). In their essence, the models in the ontology involve how state evolves over time, although they may focus on other concepts such as entity, event or place. The state space is typically discrete (finite or countably infinite), 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 as well as facilitating meta-modeling and multi-modeling.

4 A Backbone Taxonomy for DeMO

As a first step in the development of the DeMO ontology, we start by defining a taxonomy of discrete-event modeling formalisms. We then use it as a backbone taxonomy on which the rest of the ontology is built. The goal here is to group the existing discrete event modeling formalisms into some kind of organized structure, and use it as a starting point for building an ontology. (A good review and evaluation of existing formal approaches in the field of discrete-event simulation is given in (Page, 1994)).

We define a series of top formalisms that differ from each other in their structural principles motivated by their word-views and modeling power. The assumption is that each of the top formalisms are general enough can serve as 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-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 DES models. Links to other Web resources can be established as well.

Model types are typically defined as n-tuples where the elements of a tuple are *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 usually defined and a collection of these elements (an n-tuple) serves as a formal definition of the model. In building the taxonomy we attempt to take a somewhat different approach: the elements are defined 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 DeMO ontology are defined. In addition to that, we try to move the concepts up the ontology's class hierarchy, so long as it makes sense to do so. 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 are 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 the 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.

We have factored out the definitions of three sets common to all of the individual models and moved them into the ontology's root class called DiscreteEventModel. These three sets are basic and fundamental, so all the types of models defined in DeMO will have characterizations (explicitly or implicitly) for these sets: State-space, Event-set and Time-set (S, E, 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 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 ($t \in Z^+$) for discrete-time models and real-valued ($t \in R^+$) for continuous time models (vector-valued time is also a possibility).

For other types of models (e.g., Petri Nets), multiple entities (or tokens) may be required. In these types of models, a token resides at a given place until it is involved in a transition.

Based on their structural characterization (i.e., the sets used to define the models structure), we show our first-level classification.

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

4.1 State-Oriented Models

In the DeMO ontology, state-oriented models are characterized by the three sets, S, E and T. These three fundamental sets establish the type of world that is being modeled. Worlds exist in space-time. The nature of the time in the world is characterized by a time set, T. At a given point in time, the configuration, location and properties of the things within the world space constitute the current state of the model and can be represented as a point in the state space, S. In discrete-event modeling, changes to the state of the model occur only at discrete points in time, called event occurrence times. The types of changes that can occur are referred to as events. The set of possible events that can occur are collected together in an event set, E.

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). Note, one could also consider static models (as opposed to dynamic models), in which T is the empty set (i.e., time plays no role in the model). However, such models usually have a sequence of inputs or outputs and one could consider the sequence number (e.g., the 5th input) to represent some form of time.

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 (Hass, 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. [1]

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. Instead, we extend the domains of c and d to accept *nil* arguments, upon which c gives the starting time and d gives the starting state.

4.1.1 Generalized Semi-Markov Processes

For simplicity, we 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.

[1] So far we have specified three of five "w" words: what, when and where. Who will come into play on introducing entities. Why is beyond the scope of this work.

Clock Function - $c : S \times E \rightarrow T$ according to F_c

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

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 time t. From the set of candidate events, let the state holding time t^* to be the minimum of all the event clocks.

$$t^* = min\{c(s_i, e)|a(s_i, e) \geq rn\}$$

where $rn \in (0, 1]$ is a random number (if a is deterministic simply replace rn with 1). Determine the set of imminent events to be those that will occur when the time t advances t^* units forward.

$$e^* = \{e|c(s_i, e) = t^*\}$$

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., $d : S \times 2^E -> S$ as done by (Hass and Shedler, 1987) in their definition of GSMP. The 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. Given a Glynn formulation of GSMP, this restriction produces an 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.

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

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). Rather than delve

into each possibility, we will now only consider the most common option in which a is deterministic, c is stochastic and d is stochastic.

Of course these definitions may be generalized to, for example, account for simultaneous events (Hass and Shedler, 1987). However, at this point, we would like to specialize our results. If from state s_i, we could determine the probability of event e occurring, a simpler state transition probability could be computed. The state transition probability may be computed by summing over all events e that could cause a transition from state s_i to state s_j weighted by the probability of event e occurring first giving the model is in state s_i.

$$Pr\{s_i, s_j\} = \sum_e Pr\{s_i, e, s_j\} Pr\{e|s_i\}$$

Unfortunately, unless old events are cancelled upon entering a new state, the conditional probability of event e occurring will depend on the history of the model, not just its current state.

4.1.2 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 (df) F_c needs to be specified for each state s_i.

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

which is the probability that upon entering state s_i, the models stays there for at least t 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.

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

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.

$$F_c(s_i; t) = 1 - \mathbf{P_d}[s_i, s_i]^t \text{ where } t \in Z^+$$

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.

$$Pr(t^* \geq t_0 + t | t^* \geq t_0) = Pr(t^* \geq t) = 1 - F_c(s_i; t)$$

This simplifies the mathematics describing the model dynamics, leading to straightforward transient and equilibrium solutions.

For DTMC models, the probability of the next state becoming s_j is the sum of the probability of being in state s_i times the probability of making a transition from state s_i to s_j.

$$Pr\{s(t+1) = s_j\} = \sum_{s_i \in S} Pr\{s(t) = s_i\}\mathbf{P_d}[s_i, s_j]$$

From this notion, equations can be developed yielding transient and equilibrium solutions are under appropriate conditions (Ross, 1983). Similar results are obtainable for CTMC models.

4.1.3 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,

$$c : E \rightarrow T \text{ according to } F_c$$
$$F_c(e; t) = Pr\{c(e) \leq t\}$$

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.

$$a : S \times E \rightarrow \{0, 1\}$$

$$d : S \times E \rightarrow S$$
$$p_d(s_i, e, s_j) = Pr\{d(s_i, e) = s_j\}.$$

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

$$\{e_{kl} | k \in E \wedge l \in Z^+\}$$

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:

$$t = \textbf{if } a(s_i, e_{kl}) \textbf{ then } c(e_{kl}) \textbf{ else } \text{throw exception}$$

$$s_j = d(s_i, e_{kl}) \text{ with probability } p_d(s_i, e_{kl}, s_j)$$

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

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

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 occurrence until event e reoccurs.

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

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). [2]

The tables 3 and 4 summarize this discussion for State-Oriented models.

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

Table 3: Sets and Functions for State-Oriented Models

Model Acronym	**Model Name**	**Citation**
GSMP+SE	Generalized Semi-Markov Process (w/ Simultaneous Events)	(Hass and Shedler, 1987)
GSMP-SE	Generalized Semi-Markov Process (wo/ Simultaneous Events)	(Glynn, 1983)
SMP	Semi-Markov Process	(Levy, 1954)
CTMC	Continuous Time Markov Chain	(Kolmogorov, 1938)
DTMC	Discrete Time Markov Chain	(Markov, 1913)
STA	Stochastic Timed Automata	(D'Argenio et al., 1997)
TA	Timed Automata	(Alur, 1990)
SA	State Automata	(ZZ, ZZZZ)
DFA	Deterministic Finite Automata	(Huffman, 1954)

Table 4: State-Oriented Models

4.2 Event-Oriented Models

It is often the case in state-oriented models that the size of 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

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

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 Graphs model (Ÿucesan and Schruben, 1992) provides a formal framework for Event Graphs. We use it as a top formalism for Event-Oriented models. [Give their formulation????]

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 Graphs events correspond to 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 to obtain the state of the system (as a combination of state variable values) at any point of simulation of the model, but not to specify the space space explicitly before running the model.

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

We still can talk about a clock function (when will events occur):

$$c : E \times S \to T$$

This corresponds to Ÿucesan and Schruben's set of edge delay times, e.g. if occurrence of event A schedules an event B than the edge connecting A and B may have time delay τ associated with it (i.e. B is scheduled to occur τ units from occurence of event A).

A transition function (how does the state change) can also be defined:

$$d : E \times S \to S$$

As mentioned above, we can avoid explicit specification of State-space, specifying instead a set of transitions (the state variables changes) associated with each event.

The activation function needs a slight modification. It corresponds to the set of edge conditions in Ÿucesan and Schruben's notation.

$$a : E \times S \to E$$

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. It can be defined as a priority ranking function:

$$r : S \times E \to \Re^+$$

The mechanics of the model may 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 Ÿucesan, 1989). This sort of duality is important to keep in mind when constructing an ontology.

Model Acronym	Model Name	Citation
EG	Event Graph	(Schruben, 1983)
SGM	Simulation Graph Models	(Ÿucesan and Schruben, 1992)

Table 5: Event-Oriented Models

4.3 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 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(s) occur to move them to other places. In this case, the state of model can be represented as a vector $\mathbf{n}$

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

where $n_p(t) \in Z^+$ 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 though 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 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).

4.3.1 Queuing Networks

Starting with Glynn's definition (Glynn, 1983) of GSMP in which transitions are produced by single events, looking at the modeling framework in the multi-entity context leads to the definition of Queuing Network (QN) models. In queuing networks, entities move from station to station, where each station represents a service center that includes one or more service units and usually a waiting queue.

In terms of our general framework, the service part of the station may be thought of as an activity, while a queue may be thought of as a place. Therefore, one could recast the activation, clock, and transition functions (a, c and d), so that they are evaluated at the end of an activity.

$$a : (Z^+)^{|P|} \times A \rightarrow 0, 1$$
$$c : (Z^+)^{|P|} \times A \rightarrow T \text{ according to } F_c$$
$$d : (Z^+)^{|P|} \times A \rightarrow (Z^+)^{|P|} \text{ with probability } p_d$$

Let us consider an example queuing network with two stations (or queues) that work in tandem. When an entity (e.g., a job) leaves station one, it goes to station two. Suppose e is a service completion event at station one. This event causes the following state transition:

$$d(n_1, n_2, e) = (n_1 - 1, n_2 + 1)$$

When the current state $\mathbf{n}(t) = (n_1, n_2)$ was entered, the activation and clock functions may enable additional events (e.g., a service completion event), which then compete with already enabled events (e.g., an arrival event) to become the next one to fire.

In general, Queuing Network models may be viewed as special cases of a GSMP-SE models. Under appropriate conditions, they are special cases of SMP models, in which case analytic solutions for equilibrium probabilities may be obtained. Unfortunately, since the state space is now a vector space, it tends to be quite large, so solving the equations is often intractable, unless additional criteria are met to yield product-form solutions (Chandy et al., 1975).

Although the above definitions of the driving functions maintain uniformity with the state-oriented models, they leave much to be desired from the point of view of the principle of parsimony. The functions may be of very high arity, depending on the number of places $|P|$. The model is treated as a Markov model of potentially high dimensions. However, only certain types of transitions are permitted, so it is possible to define a simplified set of driving functions. The key is to focus on what can happen at single station i: entities may arrive from another station j or from outside the models (station -1), or depart to another station k or to the outside (station -1). The rates or timings of internal arrivals and all departures are governed by service completion distributions. The rates or timings of external arrivals are governed by the external arrival distribution. The specifications is completed by giving the routing probabilities q_{ij} which is the probability of a departure from i going to j.

Product-Form Queuing Networks (PFQN) allow the model to be specified in terms of rates, thus simplifying the development. Flow balance equations based on rates and routing probabilities can be expressed and solved for an equilibrium (or steady-state) solution $p(\mathbf{n})$, the steady-state probability of being in state $\mathbf{n}$

$$p(\mathbf{n}) = G \prod_i p(n_i)$$

where G is a normalization constant and $p(n_i)$ is the equilibrium probability of having n_i entities at station i.

The local balance equations leading to a product-form solutions simply equate the rate of flow into station i, with the rate of flow out of station i (certain conditions allow simpler equations to be used, e.g., partial balance or detailed balance (Kant, 1992)).

Inflow into station i is the result of external arrivals, as well as, departures from any other station j, while outflow from station i is the result of departures wherever that may go, as well as, external arrivals to any other station j.

$$\lambda(N-1)q_{-1i}p(\mathbf{n} - \mathbf{1_i}) + \sum_{j \neq i} \mu_j(n_j + 1)q_{ji}p(\mathbf{n} - \mathbf{1_i} + \mathbf{1_j}) = [\lambda(N)q_{-1i} + \mu_i(n_i)]p(\mathbf{n})$$

Therefore, PFQN models may be specified by giving (i) the external arrival rate λ which is a function of the number of entities in the system N, (ii) the service rates at the stations/activities μ which is a function of the station/activity a_i and the number of entities n_i there, and (iii) the routing probability q which gives the probability of an entity going from an activity a_i and to an activity a_j.

$$\lambda : Z^+ \rightarrow R^+$$
$$\mu : A \times Z^+ \rightarrow R^+$$
$$q : A \times A \rightarrow [0, 1]$$

4.3.2 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 of 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 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.

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 correspods 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:

$$d_i : P \times A \rightarrow Z^+$$
$$d_o : A \times P \rightarrow Z^+$$

Here $d_i(p, e) = n$ means that a transition (activity) e can be enabled only if an input place p has n tokens, and $d_o(e, p) = m$ means that once an activity e 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.

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

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

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

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

Given the current marking (or state) $\mathbf{n}(t)$ and the transition to fire e_k, the new marking simply adds the product of the incidence matrix times the one vector, which indicates which transition is firing.

$$\mathbf{1}(predicate) = \textbf{if } predicate \textbf{ then } 1 \textbf{ else } 0$$

$$\mathbf{n}(t+1) = d(\mathbf{n}(t), e_k) = \mathbf{n}(t) + \mathbf{D1}(e = e_k)$$

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.

$$A_1 \subseteq A \text{ – timed transitions,}$$
$$A_2 \subset A \text{ – immediate transitions.}$$
$$A_1 \bigcap A_2 = \emptyset, A_1 \bigcup A_2 = A.$$

Stochastic features of Extended Stochastic Petri Nets (ESPN) include stochastic clocks given as arbitrary distributions as well as random selection from a simultaneously occurring set 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).

Orthogonal to these restrictions are those based on topology. These can be expressed as restrictions on the bipartite graph representing the net or on the incidence matrix $\mathbf{D}$. A Marked Graph (MG) is a Petri Net in which each column in $\mathbf{D}$ has at most one positive and one negative value. In the graph, each place would have a single incoming arc and a single outgoing arc. Marked Graphs are also called Decision-Free Nets (DFN), since there is only path for tokens to flow into (or out of) any given place (no decision). Because of the restriction on the number of arcs coming into and out of a place, the incoming arc, place and outgoing arc may be replaced by a single arc. The arcs now connect transition to transition. One may view the (typically timed) transitions as tasks to yield a Task Graph (TG) where the nodes represent tasks and the arcs represent dependencies between the tasks. Another 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).

For these topologically restricted Petri Nets, we can classify the models based on two capabilities: (1) Fork and Join and (2) Branch and Merge. Fork and Join allows multiple tasks to be enabled once a task completes (parallel tasks) and later joined back together. Branch and Merge allows one of multiple next tasks to be selected and merged back together.

- Fork and Join – AND-Split and AND-Join – Task Graph (TG), Marked Graph (MG), Decision-Free Net (DFN).
- Branch and Merge – OR-Split and OR-Join – Queuing Network (QN).
- Both – Both – Extended Task Graph (ETG), Free-Choice Net (FCN), Event Graph (EG).

An Event Graph (EG) may be mapped to an Extended Task Graph (ETG) by interpreting each event (node) as an end-event of a transition with the transition time on the incoming arc being merged with the event to form a task. The causality arcs in the event graph then correspond to the dependency arcs in the extended task graph. Note, (Schruben and Ÿucesan, 1994) provide a mapping procedure to convert any Marked Graph into a Event Graph.

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.

The tables 5 and 6 summarize this discussion 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 6: Sets and Functions for Activity-Oriented Models

The last entry in the table is that of Bounded Petri Net. It does not necessarily have any restrictions other than having a bound on the number of tokens that can be in any one place at a particular time (e.g., k-bounded implies that $n_i(t) \leq k$ and 1-bounded implies that $n_i(t) \leq 1$).

Note, to reduce complexity, we have yet to include Colored Petri Nets (Jensen, 1981) in our ontology. In Colored Petri Nets, the entities are classified into multiple classes distinguished by different colors. For Queuing Networks, this is the same as having multiple classes of customers.

4.4 Process-Oriented Models

Process-oriented models are naturally centered around the idea of a process. A process may be thought of as an active entity. Unlike the other three types of models, less work has been done to give process-oriented models a formal foundation.

A process may be viewed as a template consisting of a collection of activities and decisions with a purpose of achieving some goal, e.g., obtaining a loan. Entities that follow the Process Template (PT) are called active entities or process instances. Passive entities are called resources. A process-oriented model consists of one or more processes. At any given time, there may be several process instances for each process template (e.g., 4 loan_customers and 3 investment_customers).

Model Acronym	Model Name	Citation
ESPN	Extended Stochastic Petri Net	(Dugan et al., 1984)
GSPN	Generalized Stochastic Petri Net	(Marsan et al., 1984)
SPN	Stochastic Petri Net	(Molloy, 1982)
ACD	Activity Cycle Diagrams	(Hills and Poole, 1969))
TPN	Timed Petri Net	(Ramchandani, 1974)
PN	Petri Net	(Petri, 1962)
FCN	Free Choice Net	(Hack, 1972)
ETG	Extended Task Graph	(ZZ, ZZZZ)
QN	Queuing Network	(Jackson, 1963)
PFQN	Product-Form Queuing Network	(Chandy et al., 1975)
MG	Marked Graph	(ZZ, ZZZZ)
DFN	Decision Free Net	(ZZ, ZZZZ)
TG	Task Graph	(Baer, 1968)
BPN	Bounded Petri Net	(ZZ, ZZZZ)

Table 7: Activity-Oriented Models

Each process template may be depicted as a directed graph, where each node represents a reactivation or control point (Cota and Sargent, 1992) and each edge represents a transition to another control point. A process instance runs until reaching a control point where it will wait either for a known amount of time (unconditional wait) or for a condition to become true (conditional wait). The complete state s of a process at a control point p consists of the values of its local variables v and the shared resources r. A process-oriented model would keep track of the following information for each process template.

$$(P, S, T, a, c, d)$$

where P is the process's set of control points (somewhat analogous to places), S is the process's state set, T is the time set, $a : P \times S \times P \rightarrow \{0, 1\}$ is the activation function, $c : P \times S \rightarrow T$ is the clock function, and $d : P \times S \rightarrow P \times S$ is the transition function.

More recently, formalization efforts have been carried out by (Birtwistle and Tofts, 2001) to provide clear semantics for Discrete Event Modelling on Simula (DEMOS) simulation system (Birtwistle, 1979) as well as the corresponding modeling approach of using Activity Diagrams (AD). Activity diagrams are directed graphs with two types of nodes: activity nodes and resource nodes.

The semantics of DEMOS's structure and behavior are defined using both operational semantics and denotational semantics (Birtwistle and Tofts, 1996). Operational semantics defines structure and behavior to the level of detail that a simulator may be automatically generated. Internals such as event lists are explicitly represented. Denotational semantics provides a higher-level, more abstract approach. Denotational semantics do not prescribe in detail the implementation, but are better for analyzing properties of models.

4.5 Model Results

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 means

Model Acronym	Model Name	Citation
PT	Process Templates	(Cota and Sargent, 1992)
AD	Activity Diagrams	(Birtwistle, 1979)

Table 8: Process-Oriented Models

reject 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 machine in which outputs depend on both states and inputs (Mealy, 1955) Generally, we may utilize an output function of the following form:

$$f : S \times E \rightarrow E$$

The outputs of simulators based on either GSMP or GSPN can be analyzed to produce transient or steady-state results. One could establish criteria for acceptable results in order to find out which inputs (the values of the input parameters) produce acceptable results. This could be viewed as an input characterization problem akin to language recognition in automata. Stepping further, one could attempt to find the inputs that optimize the results. These results are based on running the model (usually many times). The models can also be statically analyzed, i.e., their behavior can be characterized without actually running the model. Properties such as reachability, deadlock, liveness, termination, ergodicity, equilibrium probabilities, etc. can be analyzed, although in many cases the complexity is prohibitive or even uncomputable (REF).

5 DeMO Ontology

The previous section developed a backbone taxonomy for discrete-event models to serve as a foundation for the development of the DeMO ontology (see Figure 1). 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 relations between these concepts. In fact, it is not 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 DE 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 DE modeling techniques included in the DeMO ontology or to capture all the nuisances of the DE 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 flush out potential benefits and pitfalls of the methods and principles used in its construction. Some of the decisions

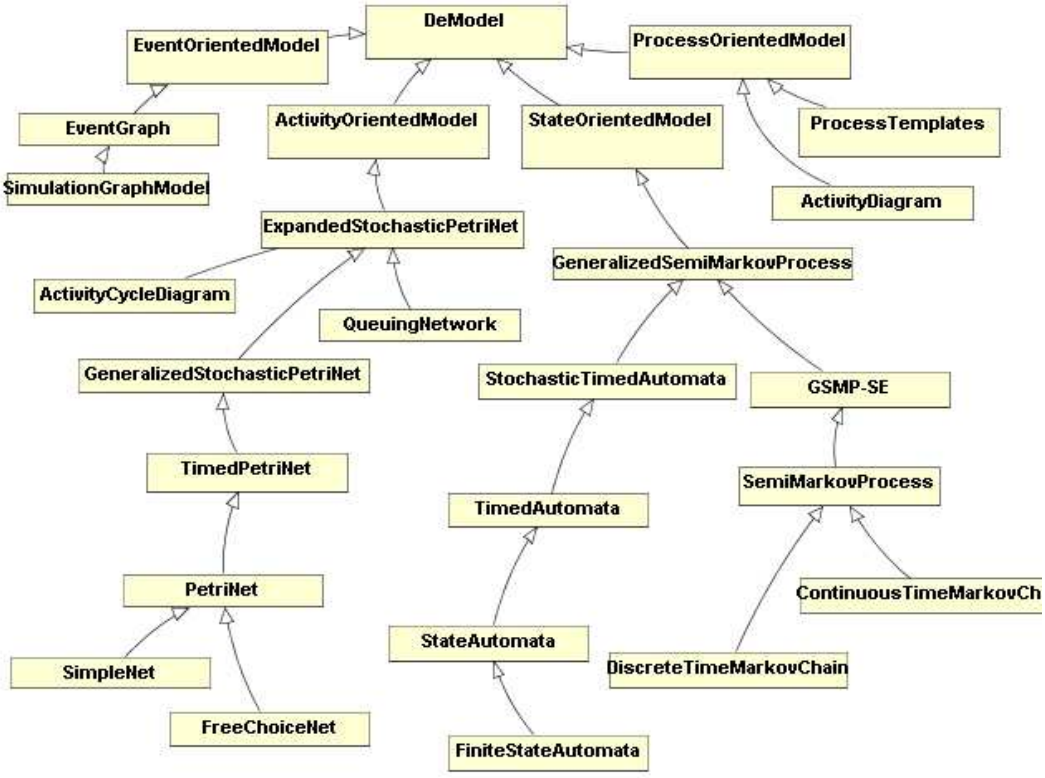

Figure 1: Portion of the DeMO's backbone taxonomy

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.

5.1 How Much Knowledge Should DeMO Have?

When building DeMO we do not have any specific usage of the ontology in mind (though some of the possible applications are discussed in section 5.5). It may seem therefore that our main goal at the moment is to capture as much knowledge about DE modeling domain as possible, i.e. provide machine-understandable formal specifications for all related concepts and their relations (at least within the scope defined by our backbone taxonomy). This, of course, may prove to be a fool's errand 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 choice 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 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. DeMO ontology, for example, may need some fundamental

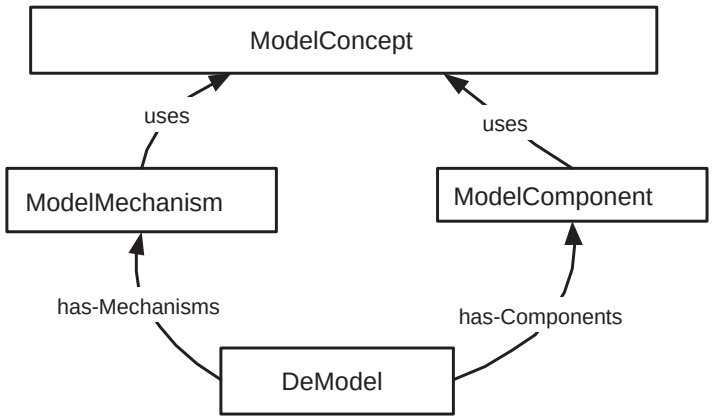

Figure 2: Schematic illustration of the rationale behind DeMO design

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 for example. This approach seems to be most promising and will probably be widely used in the future when numerous ontologies from different knowledge domains are finalized. However, since SUMO is still being refined (and to avoid unnecessary complications), we decided to do this 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.

5.2 DeMO Building Principles

As mentioned in section 2.3 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 well as some rules (mechanisms) providing specifics of how the models should run (syntax and semantics of the model). This is schematically illustrated in Figure 2.

Abstract classes Model, ModelComponent, and ModelMechanism are created. The subclasses of the class Model 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 correspond to the elements of the -tuples used in formal definitions of the models in section 4. The subclasses of the class ModelMechanism define how the components operate within the model.

In other words in order to define the Model within the ontology one should define the appropriate subclasses of ModelComponent, place them in the Model (i.e. relate the Model class with these components), define the appropriate subclasses of the ModelMechanism explaining how these components work, and relate the component classes to mechanisms. (This approach seems to be general and not-committing enough to be true for any modeling formalism, thus "keeping the door open" for upper-level formalisms not yet included in DeMO.)

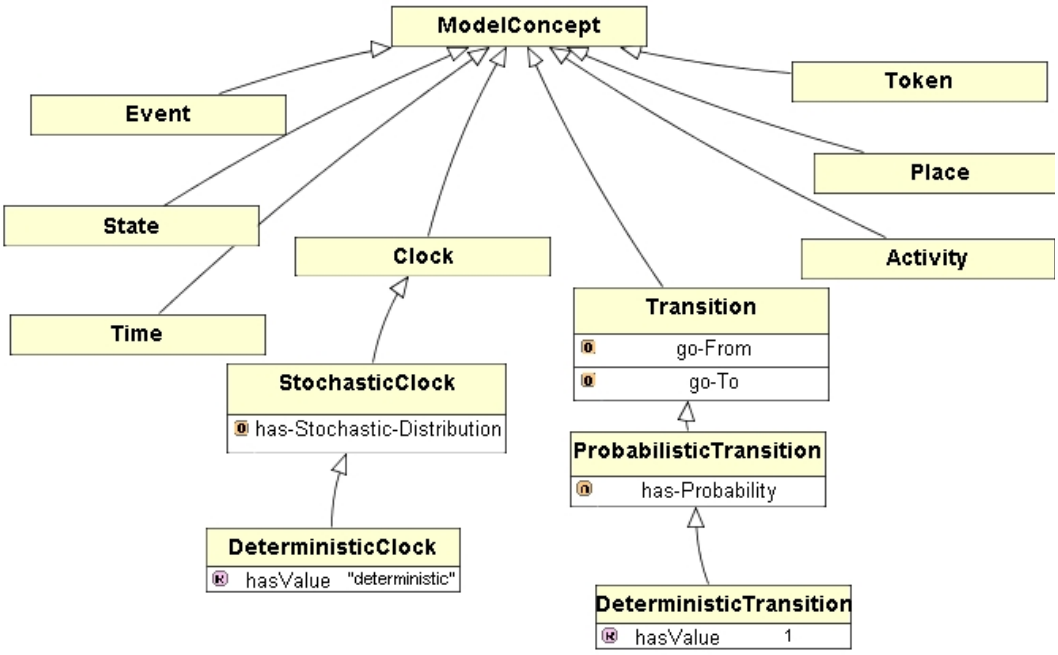

Figure 3: Portion of the ModelConcept ontology

The distinction between ModelComponents and ModelMechanisms seems to be arbitrary - some models may be reformulated in such a way that a mechanism can become a component and vice versa. However it 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 Model classes and give rise to natural taxonomies (hierarchies) within Model class different from our backbone taxonomy (?example?). 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-off` relations, it should still "emerge" as a natural hierarchy).

5.3 Defining the Base Terminology

ModelComponents are 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 are defined it 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 meta-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).

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 can not 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 the basic terms are "translated" to a machine the rest of the ontology becomes machine-understandable.

Within Protégé OWL environment we than 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 term 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 mechanism of moving the cut-off points of the ontology, based on some control parameters. This will create a kind of layered ontology, where layer 1, for example, has the class State as a basic concept, and level 2 has subclasses of StateVariable 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 `can-Occur-at`-instance-of-class-Time. 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 couple 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 is a couple of ways to implement these relations. One is to assign a property to each of the two ModelMechanism relating them to the class Transition. Say, `is-applied-to`-an-instance-of-class-Transition. Another is to assign two properties to the class Transition: `is-Enabled-by` and `is-Triggered-by`. 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 becomes 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 meta-ontology makes no sense, since it is now "dependant" on another ontology, and we loose the structural modularity offered by a meta-ontology. 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 usually the answer probably depends on the application.

5.4 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 the abstract class *DiscreteEventModel* (called here Model for short) splits into four first-level subclasses:

- *State-Oriented Model,*
- *Event-Oriented Model,*
- *Activity-Oriented Model,*
- *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 chapter 4. One way to implement it in OWL is to define the subclasses of ModelComponent and ModelMechanism: State-OrientedComponent, State-OrientedMechanism, Event-OrientedComponents etc. Then, obviously, any state-oriented modeling technique (any subclass of State-OrientedModel) is defined by putting restrictions on the properties defined by subclasses of State-OrientedComponent and State-OrientedMechanism.

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

`has-StateSpace`	Instance of	**DiscreteStateSpace**
`has-EventSet`	Instance of	**FiniteEventSet**
`has-TimeSet`	Instance of	**TimeSet**
`has-ActivationFunction`	Instance of	**ActivationFunction**
`has-TransitionFunction`	Instance of	**TransitionFunction**
`has-ClockFunction`	Instance of	**ClockFunction**
`has-InitialState`	Instance of	**InitialState**

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

For the State-OrientedModel the top subclass is a Generalized Semi-Markov Process (GSMP) model. For the Event-OrientedModel the top subclass is Simulation Event Graph (SEG). For Activity-OrientedModel this is Extended Stochastic Petri-Net (ESPN), and for Process-OrientedModel – Control Flow Graph (CFG).

Consider for example the Generalized Semi-Markov Process 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:

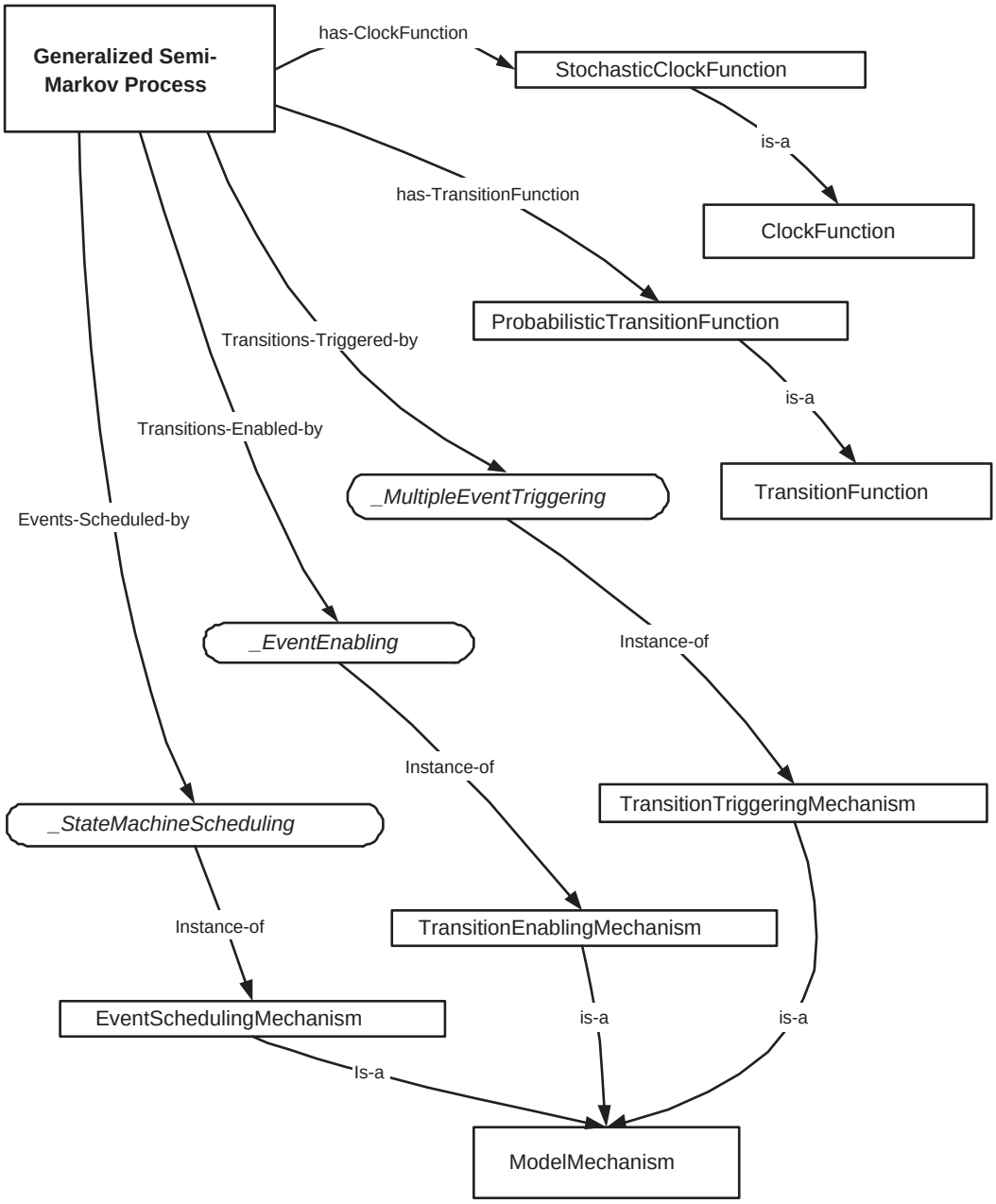

Figure 4: Graphical Representation of GSMP Portion of the DeMO Ontology

TransitionTrigering has value **_MultipleEventTriggering mechanism**
TransitionEnabling has value **_EventEnabling mechanism**
EventScheduling has value **_AutomataScheduling mechanism.**

Here the _MultipleEventTriggering mechanism is an instance of TransitionTrigering mechanism such that allows simultaneous event occurrence to trigger a transition, _EventEnabling mechanism means that the transitions are enabled together with events, and _AutomataScheduling mechanism determines the set of imminent events as described in 4.1.1.

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

TransitionTrigering 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. As described in 4.1.1 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 _MarkovianSheduling mechanism. On the other hand, effectively the same model will be produced by putting restrictions on the cardinality of the EventSet setting it equal to one.

(This ambiguity in derivation may not be desirable but may also turn out to be unavoidable. This issue has to be addressed with more attention.)

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 description.

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? We fill that the answers to these questions should be sought by consensus and/or converged to by the trial and error approach.

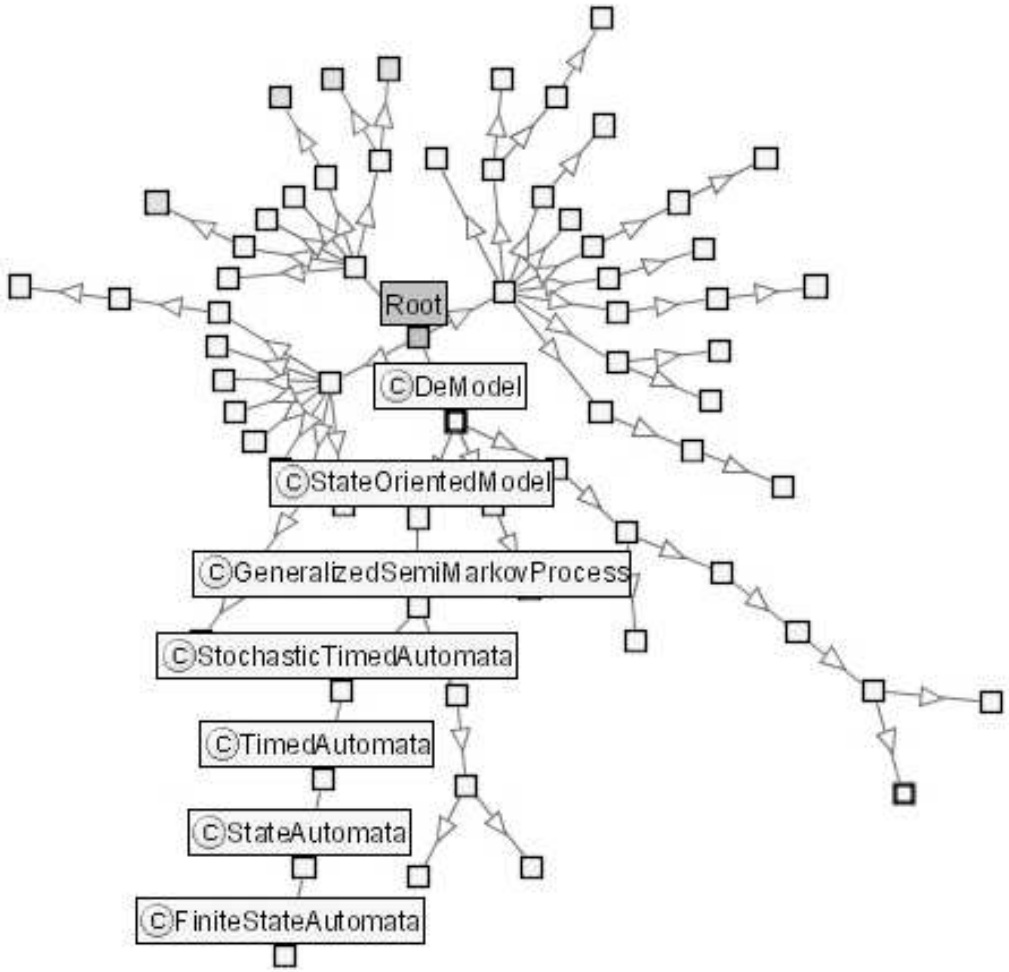

Figure 5: Portion of the DeMO ontology visualized using Jambalaya Protégé plugin

5.5 Benefits of Such Ontologies

An ontology for discrete-event modeling and simulation, such as DeMO, ultimately should be judged by the benefits it brings. We envisage several possible benefits or usages for such ontologies.

- *Browsing.* One could look at the concepts in the ontology and navigate to related concepts. This can be done with Protégé by expanding/contracting the class hierarchy tree. EzOWL and Jambalaya plug-ins provide GUI-oriented visualization of the same hierarchy. (Figure 5, for example, shows a screen-shot of a part of DeMO ontology using Jambalaya tool. Each node represents ontology class, and edges symbolize a subclass relation. Other relations can also be visualized in a similar manner.) Finally, there is some initial work on making OWL tools interoperate UML tools.

- *Querying.* Query languages are being developed to provide powerful querying capabilities for one or more ontologies. Such languages and tools will augment today's search engines. There are many such

languages under development (e.g., RQL, DQL, OWL-QL) and more work is required as the Semantic Web stack gets fleshed out. Current work will soon establish the next layer, the logic layer (e.g., SWRL) and 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 the use of ontologies in Web service discovery, please see (Verma et al., 2003). For using Web services in simulation modeling, see (Chandrasekaran et al., 2002).

- *Components.* By providing an ontology, we create a Web-based infrastructure for storing and retrieving executable simulation model components. These components can facilitate reuse. For example, consider `ProbabilisticTransitionFunction` in Figure 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.

- *Hypothesis Testing.* The LSDIS Lab is currently carrying out funded research to allow hypothesis testing to be carried out using the Semantic Web (Sheth et al., 2002). 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 (Sheth el al., zzz).

- *Mark-up Language Anchor.* It is popular today to develop domain-specific XML-based mark-up languages. This allows interfaces to software or descriptions of software to be presented in platform and machine-independent ways. In the simulation community such work has begun. For example, ¡*talk about RUBE-XML*¿. This enhances the interoperability of simulation models. The tags used in the markup language should be anchored in a domain ontology.

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

- *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)). Such consistency checking will become even more useful when rules are specified (e.g., using SWRL).

6 Conclusions and Future Work

This paper represents an early attempt to create an ontology for modeling and simulation. Such an ontology could contribute to increasing collaborative work in these areas as well as enhance the reusability of artifacts. DeMO is intended to be a starting point for an eventual well-accepted ontology for Modeling and Simulation. One way for this to happen quickly is to start with a core set of papers on the subject and then form 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 (or 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.
- Breadth of the Ontology. If the domain ontology is too broad it may become too complex and disjointed. Ambiguities may be quite difficult to resolve. On the other hand, if it is too narrow, it is of limited use.
- 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 Diagrams (ND) models for simulators such as SLAM, SIMAN and GPSS into our ontology. Both types of diagrams are similar in that they divide entities into two classes: those requiring full histories which they called entities, and those that do not which they called resources (Birtwistle and Tofts, 2001). Furthermore, there is little work on formalization of Network of Network models. Although one could map the concepts/constructs in these models to our existing framework, their richness would be lost, so a minimal extension to our framework is likely to be the best approach.
- Incorporate or link to Discrete-EVent Simulation (DEVS) models which could be viewed as a sub-taxonomy or sub-ontology of their own right (Sarjoughian et al., 2001).
- Semantics in OWL ontologies relies on the use of natural language to supply meaning where OWL constructs cannot. Use of 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.

References

(Baer, 1968) J. L. Baer, "Graph Models of Computations in Computer Systems," PhD thesis, Dept. of Elect. Engg., UCLA, Los Angeles, CA.

(Berners-Lee et al., 2001) T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web," *Scientific American*, Vol. 284, No. 5, pp. 34-43.

(Birtwistle, 1979) G.M. Birtwistle, *Discrete Event Modeling in SIMULA*. MacMillian, London, England.

(Birtwistle and Tofts, 1996). G. Birtwistle and C. Tofts, "Relating Operational and Denotational Descriptions of πDemos,"

(Birtwistle and Tofts, 2001) G. Birtwistle and C. Tofts, DEMOS 2000 – A Semantically Justified Simulation Language

(Cassandras and Lafortune, 1999) C.G. Cassandras and S. Lafortune, S., *Introduction to Discrete Event Systems*, Kluwer.

(Chandrasekaran et al., 2002) S. Chandrasekaran, G. Silver, J.A. Miller, J. Cardoso and A.P. Sheth, "Web Service Technologies and their Synergy with Simulation," *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, pp. 606-615.

(Cota and Sargent, 1990) Cota, B.A. and Sargent, R.G. (1990). "Simulation Algorithms for Control Flow Graph Models," CASE Center Technical Report 9023, Sryacuse University, Syracuse, NY, November.

(Cota and Sargent, 1992) 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), pp. 109-129, April.

(Denny, 2002) M. Denny, "Ontology Building: A Survey of Editing Tools," *www.xml.com/pub/a/2002/11/06/ontologies.html.*

(Dugan et al., 1984) J.B. Dugan, K.S. Trivedi, R.M. Geist, V.F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis. *Performance '84–Models of Computer System Performance*, E.Gelenbe, ed., Elsevier Sci. Publ. Co., Inc., pp. 507-519.

(Fishman, 1973) G.S. Fishman, *Concepts and Methods in Discrete Event Digital Simulation*, Wiley, New York, NY.

(Fishwick, 1995) P.A. Fishwick, *Simulation Model Design and Execution: Building Digital Worlds*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

(Fishwick, 1996) P.A. Fishwick, "Web-Based Simulation: Some Personal Observations," *Proceedings of the 1996 Winter Simulation Conference*, San Diego, CA, pp. 772-779.

(Fishwick, 1996b) P.A. Fishwick, "A Taxonomy for Simulation Modeling Based on Programming Language Principles," *IIE Transactions on IE Research*, Vol. 30, pp. 811-820.

(Fishwick, 2002) P.A. Fishwick, "Using XML for Simulation Modeling," *Proceedings of the 2002 Winter Simulation Conference* San Diego, CA, pp. 616-622.

(The Gene Ontology Consortium, 2000) The Gene Ontology Consortium, "Gene Ontology: Tool for the Unification of Biology," *Natural Genetics*, Vol. 25 pp. 25-29.

(Gruber, 1995) Gruber, T.R. "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *Int. Journal of Human-Computer Studies*, Vol. 43, pp. 907-928.

(Haarslev,V. and Möller,2001) Haarslev,V. and Möller, R. "Description of the RACER System and its Applications". *To appear in: Proceedubgs International Workshop on Description Logics (DL-2001), Stanford, USA, August 1-3, 2001.*

(Hills and Poole, 1969) Hills, B.R. and Poole, T.G. "A Method for Simplifying the Production of Computer Simulation Models," TIMS Tenth American Meeting, Atlanta, GA, October 1-3, 1996.

(INCOSE, 2002) INCOSE, "SE Tools Taxonomy - Simulation Tools," www.incose.org/tools/tooltax/simulation_tools.html.

(Kim et al., 2002) T. Kim, J. Lee, and P.A. Fishwick. "A Two-Stage Modeling and Simulation Process for Web-Based Modeling and Simulation", ACM Transactions on Modeling and Computer Simulation, Vol. 12 no. 3, pp. 230-248.

(Laclaustra, 1990) J.C. Laclaustra, "Performance Bounds for Synchronized Queueing Networks," Ph.D. Thesis, Departamento de Ingenier'ia El'ectrica e Inform'atica, Universidad de Zaragoza, Spain, December 1990, Research Report GISI-RR-90-20.

(Lacy, 2001) L. Lacy, "Semantic Web Applications for Modeling and Simulation," www.daml.org/2001/07/dmso-applications/semantic-web-071101.ppt.

(McGuinness and van Harmelen, 2003) D.L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," www.w3.org/TR/owl-features.

(Miller et al., 1997) J.A. Miller, A.P. Sheth and K.J. Kochut, "Perspectives in Modeling: Simulation, Database and Workflow," Proceedings of the 16th International Conference on Conceptual Modeling: Preconference Symposium, Los Angles, CA.

(Miller et al., 2001) J.A. Miller, P.A. Fishwick, S.J.E. Taylor, P. Benjamin and B. Szymanski, "Research and Commercial Opportunities in Web-Based Simulation," Simulation Practice and Theory, Special Issue on Web-Based Simulation, Vol. 9, No. 1-2, October 2001, pp. 55-72

(Miller et al, 2004) Miller, J.A., Baramidze, G.T., Fishwick P.A. and Sheth, A.P. 2004, "Investigating Ontologies for Simulation Modeling", Proceedings of the 37th Annual Simulation Symposium, Arlington, VA, April 2004, pp. 55-71.

(Nair et al., 1996) R. Nair, J.A. Miller and Z. Zhang, "A Java-Based Query Driven Simulation Environment," Proceedings of the 1996 Winter Simulation Conference, Coronado, CA, December 1996, pp. 786-793.

(Nance, 1993) R.E. Nance, "A History of Discrete Event Simulation Programming Languages," Proceedings of the Second ACM SIGPLAN History of Programming Languages Conference, Cambridge, MA, Reprinted in ACM SIGPLAN Notices, Vol. 28, No. 3, pp. 149-175.

(Page, 1994) E.H. Page, Jr., "Simulation Modeling Methodology: Principles and Etiology of Decision Support", Ph.D. Dissertation, Virginia Tech. www.thesimguy.com/ernie/papers/unref/dissert/dw.html

(Page et al., 2000) E.H. Page, Jr., P.A. Fishwick, K.J. Healy, R.E. Nance, and R.J. Paul, "Web-Based Simulations: Revolution or Evolution," ACM Transactions on Modeling and Computer Simulation, Vol. 10, No. 1, pp. 3-17.

(Pegden, 1990) C.D. Pegden, R.E. Shannon and R.P. Sadowski, Introduction to Simulation Using SIMAN McGraw-Hill, NY.

(Peterson, 1977) J.L. Peterson, "Petri Nets," ACM Computing Surveys, Vol. 9, pp. 223–252.

(Petri, 1962) C.A. Petri, "Fundamentals of a Theory of Asynchronous Information Flow," Proceedings of IFIP Congress 62, pp. 386-390.

(Pidd, 1992) M. Pidd, Computer Simulation in Management Science, Wiley, Chichester, England, 3rd edition.

(Pritsker, 1979) A.A.B. Pritsker, Introduction to Simulation with SLAM, Wiley, New York, NY.

(Sarjoughian et al.,, 2001) H.S. Sarjoughian, F.E. Cellier and B.P. Zeigler (Editors), "Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and Ai-Based Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler," Springer Verlag, Berlin-Heidelberg.

(Schriber, 1974) T.J. Schriber, Simulation Using GPSS, Wiley, NY.

(Schruben, 1983) L. Schruben, "Simulation Modeling with Event Graphs," Communications of the ACM, Vol. 26, pp. 957-963.

(Schruben and Roeder, 2003) L. Schruben and T. Roeder, "Fast Simulations of Large-Scale Highly-Congested Systems," Simulation: Transactions of the Society for Modeling and Simulation International.

(Sheth et al., 2002) A.P. Sheth, S. Thacker and S. Patel, "Complex Relationship and Knowledge Discovery Support in the InfoQuilt System," VLDB Journal, Vol. , No. , pp. -.

(Tocher, K.D. 1963) Tocher, K.D. (1963). The Art of Simulation, Van Nostrand Company, Princeton, NJ.

(Verma et al., 2003) K. Verma, K. Sivashanmugam, A.P. Sheth, A. Patil, S. Oundhakar and J.A. Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," Information Technology and Management, Vol. , No. , pp. -.

(Zeigler, 1976) B.P. Zeigler, Theory of Modelling and Simulation, John Wiley and Sons, New York, NY.

(Zeigler, 1982) B.P. Zeigler, "Subject Plan for an Encyclopedia: A Taxonomy for Modelling and Simulation," ACM Simuletter, Vol. 13, No. 1-4, pp. 55-62.