Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.

SUPPORTING INTEROPERABILITY USING THE DISCRETE-EVENT MODELING ONTOLOGY (DeMO)

Gregory A. Silver
College of Business
Anderson University
Anderson, SC 29621, USA

Kushel Rai Bellipady
John A. Miller
Krys J. Kochut
Dept. of Computer Science
University of Georgia
Athens, GA 30602, USA

William York
Complex Carbohydrate Research Center
University of Georgia
Athens, GA 30602, USA

ABSTRACT

In modeling and simulation, the need for interoperability can be between simulation models or, more broadly, within simulation environments. For example, simulation of biochemical pathways for glycan biosynthesis will need access to glycomics knowledge bases such as the GlycO, EnzyO and ReactO ontologies and bioinformatics resource/databases. Traditionally, developers have studied these information sources and written custom simulation code with hardlinks into, for example, databases. Our research explores a technique which allows developers to create a conceptual model using domain ontologies, and then use alignment and mapping information between the domain ontologies and the Discrete-event Modeling Ontology (DeMO) to create DeMO instances which represent a model that conforms to a particular simulation world view. Once the DeMO instances have been created, a code generator can be used to produce an executable simulation model. This paper discusses several situations in which DeMO can support interoperability but focuses primarily on interoperability between domain ontologies and DeMO.

1 INTRODUCTION

Every software system has implied meaning associated with the concepts it uses during execution. The meaning of these concepts is typically assumed by the systems that use them, however, since each system places its own meaning on concepts, problems may occur when systems are expected to interoperate with one another. Assumptions about the meaning of concepts are usually designed into systems by those who develop them. One way to address this problem of disparate meanings is to develop a standard, external to the interoperating systems, that encodes the meaning of the concepts. Any such standard must be structured in such a way that it can be understood by both machines and humans. An increasingly accepted way of standardizing the meaning of concepts within particular knowledge domains is through the use of ontologies (Gruber 1993). An ontology is a structured definition used to describe and categorize concepts and the relationships among concepts in a particular knowledge domain, and they can be constructed so that they are readable by both machines and humans.

While technical interoperability, the ability of systems to exchange data and services using a common protocol, has been achieved using interoperability-protocols such as the High Level Architecture (HLA), semantic interoperability, the ability of systems to exchange data with an unambiguous shared meaning, is difficult to achieve without the use of a reference model or ontology which defines the meaning of the data. The use of ontologies in modeling and simulation has recently emerged as a growing area of research interest as evidenced by the creation of the Discrete Even Modeling Ontology (DeMO) (Miller et al. 2004), the evaluation of the Command and Control Information Exchange Data Model (C2IEDM) as an interoperability-enabling ontology (Tolk 2005), the development of the Process Interaction Modeling Ontology for Discrete Event Simulations (PIMODES) (Lacy 2006), the development of the Component Simulation and Modeling Ontology (COSMO) (Teo and Szabo 2008), and the use of domain ontologies in agent-supported interoperability of simulations (Yilmaz and Paspuleti 2005). Ontologies can be utilized in several ways, but are particularly relevant to simulation interoperability. The main reason is that what is needed for interoperability is a basis or agreed upon standard on which two separately developed components can rely. Such a standard could take many forms, e.g., it is popular to express them as XML documents. More recently, the advantages of enforcing logical consistency in such specifications, which can become large, have been noticed. Hence, several modeling and simulation ontologies such as DeMO, PIMODES, and COSMO have been developed using the Web Ontology Language (OWL), which is based on description logic, allowing ontologies to be programmatically checked for consistency. In OWL, classes are used to specify concepts and properties are used to specify both relationships between classes as well as values.

While both technical and semantic simulation interoperability typically focus on runtime interchange of information, the methodology presented in this paper focuses primarily on the interchange of information between models rather than executing simulations. We address interoperability in three different areas:

- Interoperability between conceptual models built using domain ontologies and discrete-event models built using the DeMO ontology
- Interoperability between discrete-event models built using DeMO and discrete-event simulators
- Interoperability between models using different simulation worldviews or model formalisms

Our methodology takes into account that nowadays much application and domain knowledge is captured in ontologies, and we use this knowledge to aid and expedite the development of simulations. The methodology allows developers to create conceptual models using domain ontology instances, then transform the conceptual model into a discrete-event model represented as DeMO instances and finally generate an executable simulation model from the DeMO instances.

The remainder of this paper is organized as follows: Section 2 discusses ontology-based semantic interoperability and its role in modeling and simulation. Section 3 describes a method by which ontologies can be used to drive the development of simulation models via information interchange between conceptual models. In Section 4, we present the DeMOForge tool, which has been developed to support ontology-driven simulation (ODS), and Section 5 provides our conclusions and recommendations for future work.

2 ONTOLOGY-BASED SEMANTIC INTEROPERABILITY

Software systems that have been developed independently of one another often need to exchange data. If these systems are able to exchange data and cooperate in a way that produces meaningful results, we say that the systems can interoperate with one another. Interoperability of simulation software is an important topic within modeling and simulation. In general, we might say that two simulation components exhibit technical or syntactic interoperability if they can exchange data or services via some agreed upon protocol. This type of interoperability does not guarantee that cooperating simulations will produce meaningful results. In order to ensure that meaningful results are produced, the interoperating components must each interpret the information that is being exchanged based on some unambiguous, agreed upon, pre-established meaning. When simulations interoperate in this way, we may refer to them as having substantive, meaningful, or semantic interoperability (Dunhmann et al. 1999, Heiler 1995, Petty and Weisel 2003). We will refer to this type of interoperability as semantic interoperability throughout the remainder of this paper.

Ontologies, within the field of computer science, are formal descriptions of concepts and the relationships among them within a particular domain, and as such can be used to categorize the data exchanged between interoperating systems into classes that represent concepts. When data fits within a particular concept/class, defined in an ontology, the meaning of the data may be made evident to interoperating systems by referencing the ontology. Thus, semantic interoperability can be facilitated by allowing interoperating simulations to reference a common ontology. Tolk (2005) discusses the idea that ontologies can be used to support semantic interoperability. He makes the important point that the term interoperability generally refers to technical interoperability, but that systems must be able to do more than exchange information, they must be able to understand one another. This further level of interoperability, which he calls operational interoperability and we call semantic interoperability, requires substantial effort to realize when there are multiple systems, each with their own understanding of the concepts that they use. Tolk puts forth the idea that establishing a central reference model that conforms to a core ontology is one way to achieve this type of interoperability.

With the advent of languages such as the Resource Definition Language (RDF) and OWL, which provide support for ontology development on the Semantic Web (Berners-Lee et al. 2001), the idea of grounding the concepts used by applications in a formal ontology has become more popular (Albertsen and Bomqvist 2007). However, many different organizations develop systems which must exchange information with one another, and it is not realistic to expect all of these systems to conform to the same ontology. Since the various systems may rely on different ontologies to describe the concepts that they use, it is probable that similar concepts will be described in different ways in the various ontologies. If the systems that rely on these ontologies are to interoperate, the differences between the ontologies must be reconciled. The process of reconciling these differences is known as ontology mediation (Bruijn et al. 2006). Ontology mediation establishes relationships between various ontologies in order to reconcile these differences. There are two different ways of approaching ontology mediation, (1) ontology mapping and (2) ontology merging. In ontology mapping, correspondences between ontologies are created and maintained, and in ontology merging, multiple ontologies are merged to create a new ontology which is the union of the merged ontologies. The goal of ontology mediation is to allow an application which conforms to a specific ontology to correctly interpret information it receives from another application which conforms to a different ontology. Given that different applications may conform to different ontologies, it is obvious that something like ontology mediation is needed to achieve semantic interoperability.

Interoperability is important to the discipline of modeling and simulation on more than one level. We have seen that it is important that simulations be able to exchange meaningful information at run-time, but in order for this to occur, the conceptual models which represent the system must also take interoperability into account. Tolk (2004) addresses the issue of interoperability of conceptual models by proposing the Levels of Conceptual Interoperability Model (LCIM). Level 3 of the LCIM is the semantic level, where *"*the unambiguous meaning of data is defined by common reference models" (Tolk 2004). It seems evident that in creating these reference models one may need to deal with multiple ontologies and that ontology mediation could be used to reconcile differences among related concepts in the various ontologies. The ontology-driven simulation methodology presented in this paper attempts to achieve the type of interoperability described by level 3 of the LCIM in order to use the domain knowledge resident in ontologies to aid and expedite the development of simulations. In doing this it makes use of ontology mediation but in a slightly different way than it has been used in the past.

2.1 Interoperability Using DeMO

The objective of a developer building a model in the biochemical domain may be to create a static model that accurately represents a biochemical pathway. This model may be built by a researcher who is familiar with the biochemical domain but has limited knowledge of modeling and simulation. The model will naturally make use of biochemical concepts to represent the qualitative (structural/visual) and quantitative (mathematical) components of the pathway. In order for others working within the biochemical domain to have a clear understanding of the model, it is important that it be built using components whose meaning and format has been agreed upon by domain authorities and expressed in an unambiguous way. Ontologies are an accepted way of capturing such domain knowledge in a format that is readable by both humans and machines.

The components (classes and properties) that make up an ontology are used to represent domain concepts and relationships. These classes and properties can be instantiated to represent particular occurrences of domain concepts. Developers may actually create models by instantiating domain concepts. For example, a biochemist may create a model of a particular glycan pathway by creating instances of selected molecule, reaction and enzyme classes found in the Reaction Ontology (ReactO) `<http://ccrc.uga.edu/ontologies>`. The pathway model may then be read by other researchers within the domain or by machines with access to ReactO. We propose a methodology that uses a conceptual model, like the pathway model described earlier, along with knowledge resident in domain and modeling ontologies, to derive the creation of an executable simulation model. The DeMOForge tool allows a domain expert to develop a model by creating instances of concepts found in domain ontologies. It then uses mappings between the domain ontologies and the DeMO ontology to transform the model into DeMO instances which represent the model using a particular discrete-event simulation (DES) world view or model formalism. The DeMO instances, representing the model, can then be used to generate an executable simulation model.

The ReactO ontology may be considered a conceptual model of the biochemical reaction domain, and the DeMO ontology can be considered a conceptual model of the discrete-event modeling domain. These ontologies can be considered conceptual models of their perspective domains because they are implementation independent models which describe the domains. We can also say that a particular pathway model, built using ReactO instances, is a conceptual model of a biochemical pathway since it is an implementation independent representation of that pathway.

The DeMOForge tool allows concepts in domain ontologies to be mapped to concepts in the DeMO ontology. Once these mappings are complete, a domain expert can create a conceptual model using instances of concepts in the domain ontologies. DeMOForge may then read the mappings between the ontologies and use the instances, which represent the previously created conceptual model, to create DeMO instances which represent a conceptual model based on a supported formalism. Thus, DeMOForge allows the knowledge resident in the domain ontologies to interoperate with the knowledge in DeMO enabling the model developer to compose a simulation model using well defined concepts within a particular domain without getting bogged down in low-level implementation details of the simulation model. The developer may be a domain expert working only at the conceptual level, but the previously defined mappings (and rules) allow the conceptual domain model to be transformed into a semantically meaningful discrete-event model.

2.2 Types of Interoperability Supported by DeMO

As mentioned earlier, the primary focus of the methodology presented in this paper is ontology-driven simulation. The methodology facilitates ontology-driven simulation through the interoperation of three types of systems, (1) domain ontologies, (2) a DES modeling ontology and (3) simulation software packages. The interoperation of these systems is achieved through the use of ontology mediation. Mappings are established between concepts (classes and properties) in domain ontologies and the DeMO ontology, then the mappings are used to support data interchange between the domain ontologies and DeMO.

Specifically, instances from populated domain ontologies are transformed into DeMO instances using previously established mappings. Taken together these DeMO instances can be thought of as a conceptual model of a discrete-event simulation. If, for example, we are transforming ReactO instances that represent a biochemical pathway into DeMO instances that represent the same pathway as a Hybrid Functional Petri Net (HFPN), we can view the transformation process as supporting interoperability between a conceptual model in the biochemical domain and a conceptual model in the discrete-event modeling domain.

The next type of interoperability that is encountered during the ontology-driven simulation process is between the DeMO ontology and simulation software packages. Translation software that conforms to the DeMO ontology has been written to translate DeMO instances specifying a discrete-event model into a model that can be executed by specific simulation software. The DeMO instances that specify a conceptual DES model represent most of what is needed to create an executable model. These instances specify invariant aspects of a conceptual DES model, capturing the essential knowledge, but the variant aspects must also be specified in order execute models. In particular, the two key variant aspects, scenario management and layout management, are addressed later in the ODS process. For scenario management, our ODS process is designed to allow for data interchange with external information sources, such as ontologies, databases or spreadsheets, in order to retrieve input parameters during the translation of DeMO instances into an executable simulation model. For layout management, the fundamentals, such as the topology of the model and the components that represent it, are taken into account to calculate the coordinates of the model components to be displayed on a canvas.

The DeMO ontology can also be used to support interoperability between model types (or simulation worldviews) in the form of model morphisms. As an example, suppose that we have a discrete-event model which follows the event scheduling worldview and it is represented using instances associated with the DeMO SimulationEventGraph class. It is possible to transform this model into a model which follows the activity scanning worldview by transforming it into instances associated with one of the subclasses of the DeMO ActivityOrientedModel. The Semantic Web Rule Language (SWRL) allows logic to be expressed in relation to concepts defined using OWL. Since DeMO is written in OWL, SWRL rules can be used to morph components (instances) from one model type into components of another model type. The morphisms are be defined as SWRL rules which are maintained with the ontology.

3 ONTOLOGY-DRIVEN SIMULATION

The concepts and relationships defined in an ontology are important because they allow knowledge to be shared between various entities within an application domain in an unambiguous way. However, the knowledge resident in an ontology consists of more than just the definitions of concepts and relationships. The definition of concepts (classes) and relationships (properties) in an ontology make up the ontology schema. While the schema is important because it allows the meaning of concepts and relationships to be defined in a structured way, an ontology can be much more useful if it is populated with instances. For example, the schema of the DeMO ontology represents the various worldviews of the discrete-event modeling domain, but when DeMO is populated with instances, the ontology also contains conceptual DES models which conform to the various worldviews.

ODS uses populated ontologies to drive the creation of executable simulation models. The methodology that we propose views ODS as having three phases, (1) the mapping phase, (2) the transformation phase and (3) the generation phase. The details of each of these phases are discussed in sections 4.1 and 4.2 of the paper.

3.1 The DeMO Ontology

DeMO plays a pivotal role in supporting ODS, as instances of DeMO classes are used to represent conceptual DES models (Silver et al. 2007). In this section, we look at how DeMO uses the constructs of OWL to encode knowledge of the discrete-event modeling domain. All DES models have some basic components from which they are built and some mechanisms which specify how the models should run. Under our approach, DeMO has three top level classes: DeModel, ModelComponent and ModelMechanism as shown in Figure 1. The subclasses of ModelComponent define the building blocks of DES models, such as state, event, activity, etc., while the subclasses of ModelMechanism define how components work within the model. The DeModel class splits into four first-level subclasses: StateOrientedModel, EventOrientedModel, ActivityOrientedModel and ProcessOrientedModel. Each of these classes defines a top level DES formalism, and the subclasses of these classes represent existing modeling techniques, such as Petri Nets, Markov Chains, Event Graphs, etc., for existing DES modeling formalisms. The subclasses are created by defining appropriate subclasses of ModelComponent which are associated with DeModel via OWL properties, then subclasses of ModelMechanism are defined explaining how the components work, finally, the mechanisms are associated with components via properties. Figure 2 illustrates the structure of several levels of the DeMO DeModel class hierarchy.

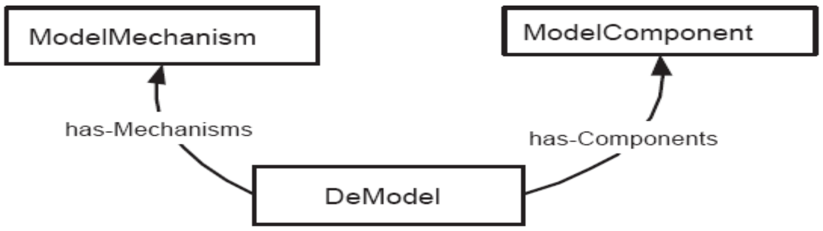


Figure 1: DeMO DeModel, ModelComponent and ModelMechanism classes

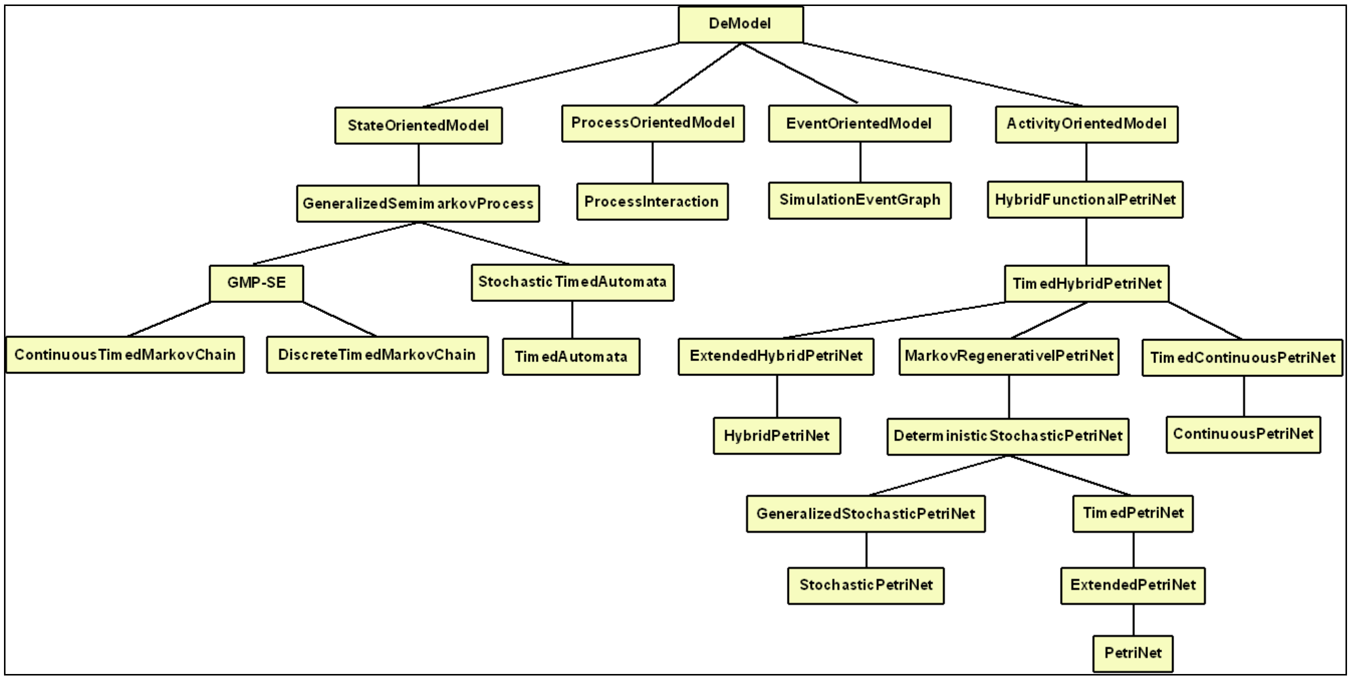


Figure 2: DeMO DeModel class hierarchy

3.2 Phases of Ontology-driven Simulation

In order to allow ontologies to drive the creation of DES models, we must be able to transform selected knowledge resident in domain ontologies into knowledge that can be used by an executable DES model. This means that the ontologies that contain the domain knowledge and the simulation software package that executes the simulation should be interconnected. Our method for enabling this involves having the domain ontologies and the simulation software package logically linked via a common DES modeling ontology (DeMO). During the first phase of ODS (the mapping phase), correspondences between concepts in the domain ontologies and the DeMO ontology are created by mapping domain ontology concepts (classes and properties) to DES modeling concepts in DeMO. The purpose of the mappings is to provide a bridge over which knowledge can be transferred from domain ontologies to DeMO in a form that is syntactically compatible and conceptually meaningful. The mappings established during the first phase are used by the second phase (transformation) to create instances of DeMO classes that correspond to instances of domain ontology classes (or class/property relationships). Before transformation can take place, it is assumed that someone has created relevant instances in the domain ontology. For example, a researcher working in the biochemical domain would have instances of ReactO ontology classes to build a model of a biochemical pathway. During the transformation phase these instances are transformed into DeMO instances, which represent the system as a conceptual model that conforms to a particular DES worldview (i.e., activity oriented) and modeling technique (i.e., Petri Nets). The next phase (generation) uses the DeMO instances, which represent the conceptual DES model, to generate an executable simulation model. As can be ascertained from the discussion above, the domain ontologies are linked to the DeMO ontology through the use of mappings, however, since it is difficult to establish explicit mappings between simulation software packages and DeMO, we developed category specific code generators based on the DeMO ontology (e.g. currently there are code generators for Petri net and process interaction models). The generators are able to use DeMO instances that represent a specific category of DES model and generate an executable model for selected simulation software packages.

3.3 Ontology-Driven Simulation of Biochemical Pathways

The ODS examples in this paper refer to the domain of biochemistry, specifically the simulation of biochemical pathways. The simulation of these pathways has historically been achieved using differential equations (Heinrich and Schuster 1998). Differential equations are able to capture the quantitative aspects of pathways, but the qualitative aspects (structure) do not lend themselves to representation via differential equations. Reddy et al. (1993) proposed that Petri Nets (PN) (Petri 1966) be used to create qualitative pathway models. A PN is a bipartite graph with place nodes, represented as circles, and transition nodes, represented as rectangles. The nodes in the graph are connected via arcs. Each arc is directed and connects either a place to a transition or a transition to a place. When an arc connects a place to a transition, the place is considered to be input to the transition, and when an arc connects a transition to a place, the place is considered output of the transition. The content of a place is known as the place's marking, and the marking is typically represented as black dots called tokens. Tokens are moved from input places to output places by the firing of transitions. In order for a transition to fire, it must be enabled by having the number of tokens in each of its input places meet or exceed a weight value assigned to the arcs that connect the places to the transition. Graphical representations of biochemical pathways, such as those associated with the Kyoto Encyclopedia of Genes and Genomes (KEGG) (Kanehisa and Goto 2000), typically consist of a graph with nodes which represent molecules or enzymes, and edges which represent reactions between them. Such a graph can be represented as a PN for the purpose of simulating biochemical pathways.

When a PN is used to model a biochemical pathway, places are used to represent molecules and enzymes, and transitions are used to represent reactions. The marking of a PN place must be discrete, and since the modeling of biochemical reactions rely on equations that use real values, traditional PNs are limited in their ability to model pathways. The introduction of the Hybrid Petri Net (HPN) (Alla and David 1998) extended PNs to include place and transition nodes that were able to handle real values. The new place nodes (continuous places) could contain real valued markings, and the new transitions (continuous transitions) could be assigned a firing rate to determine the speed at which they were to consume the markings of their input places. Since concentrations of molecules and enzymes could now be modeled as real values using continuous places, and the rate at which molecule concentrations were consumed by reactions could now be modeled as a real valued firing rate, modelers could more accurately represent the dynamics of a pathway.

The HFPN, introduced by Matsuno et al. (2001) extended the HPN for the purpose of more accurately modeling biochemical pathways. One significant enhancement was the addition of the test arc. A normal PN consumes the markings of its input places when it fires, but if the input place is connected to the transition via a test arc, the markings of the input places are not consumed when the transition fires. Since reactions do not actually consume enzymes, this extension enabled model developers to more accurately model the relationship between an enzyme and a reaction.

Several ontologies, such as the Glycomics ontology (GlycO) (Thomas et al. 2006), which focuses on the glycoproteomics domain, the Enzyme ontology (EnzyO), which describes enzymes, and the Reaction ontology (ReactO), which describes enzyme-catalyzed reactions, have been developed for portions of the domain of biochemistry. ReactO, which references concepts in both GlycO and EnzyO, along with DeMO was used extensively during the development of the ODS methodology presented in this paper. ReactO can be populated with instances which can be configured to represent conceptual models of metabolic pathways. Using our ODS methodology, we take these conceptual pathway models and transform them into instances of DeMO classes which are then configured to represent a conceptual HFPN model of the pathways. Once created, these DeMO based conceptual HFPN models can be translated into executable simulation models using a code generator.

4 THE ONTOLOGY-DRIVEN SIMULATION TOOL (DeMOForge)

In this section, we provide an overview of the DeMOForge ontology-driven simulation tool and give an example of its use in the biochemical domain. In our version of ontology-driven simulation (Benjamin et al. 2006), we use concepts and relationships resident in both domain ontologies and modeling and simulation ontologies to drive the creation of simulation models (Silver et al. 2007). DeMOForge provides three modules each of which supports a specific phase of ontology-driven simulation. The DeMOForge Mapping Module supports the mapping phase in which relationships between domain ontologies and the DeMO ontology are established by mapping concepts from domain ontologies to corresponding concepts in DeMO. The Transformation Module supports the transformation phase in which a conceptual model represented as domain ontology instances is transformed into a discrete-event model represented as DeMO instances, and the Generation Module supports the generation phase in which DeMO instances representing a discrete-event model are translated into an executable simulation model. Figure 3 gives a graphical depiction of the DeMOForge architecture.

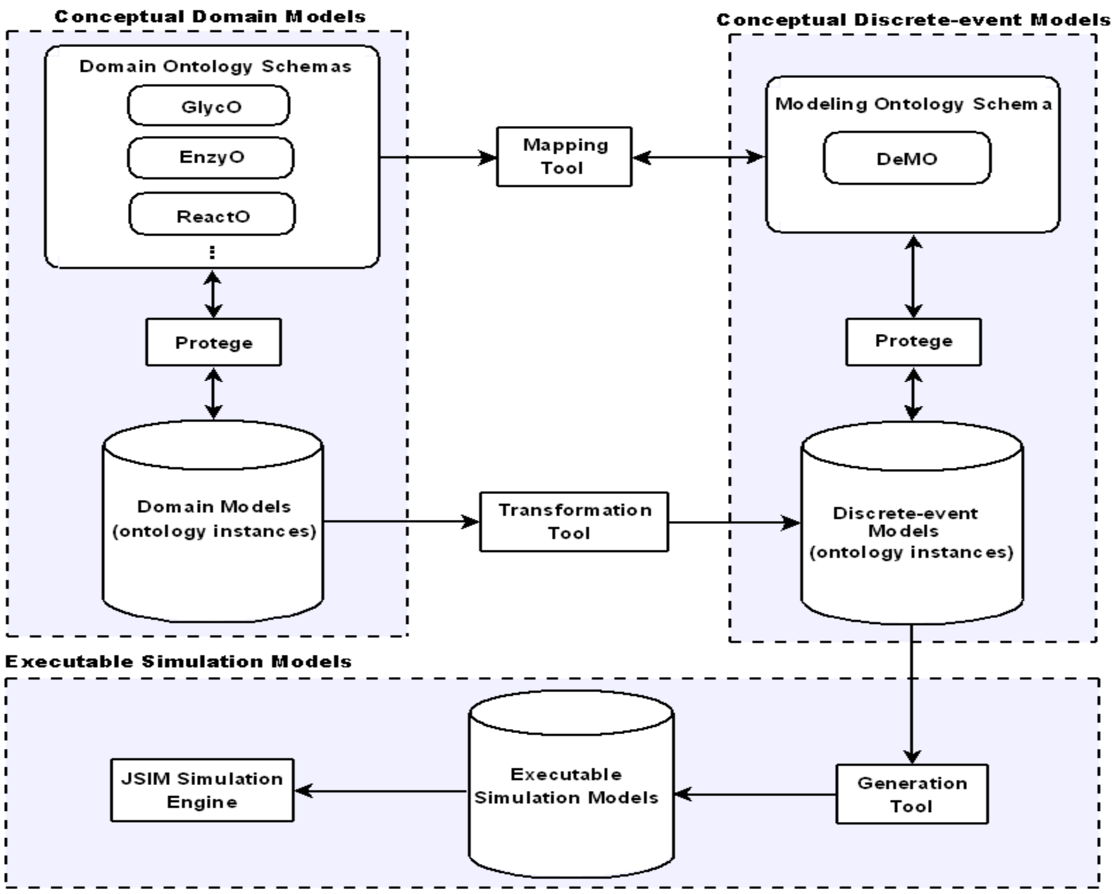


Figure 3: DeMOForge architecture

4.1 Mapping Phase

DeMOForge supports the mapping phase of ontology-driven simulation by providing a facility for creating mappings between domain ontologies and the DeMO ontology. While these mappings are, on a technical level, similar to the mappings that often occur between domain ontologies, their purpose is different. The mappings that occur between domain ontologies are typically used to establish relationships (such as equivalence and subsumption) between two or more concepts. For example, one domain ontology may use the term 'automobile' and another may use the term 'car' to describe the same concept. Machines, or humans, which make use of both of these ontologies will need to understand that both terms refer to the same concept, and the accepted way of facilitating this understanding is to create a mapping between the two terms. In other words, the mapping is specifying that 'automobile' and 'car' are equivalent. DeMOForge creates mappings, not between two domain ontologies, but rather between domain ontologies and a modeling ontology, and the concepts being mapped do not necessarily have obvious relatedness. For example, in the biochemical domain, the Reaction class of ReactO is used to represent a reaction within a biochemical pathway. In a HFPN, reactions are often represented by continuous transitions. Given this information, a developer may create a mapping between the Reaction class of ReactO and the ContinuousTransition class of DeMO. This mapping specifies that a biochemical reaction corresponds to a continuous transition for the purpose of discrete-event simulation. In this case, relatedness will often not produce equivalences or subsumptions but rather connects concepts as analogs. These mappings are not typically created during model composition but are created as a collaborative effort between domain experts and discrete-event simulation experts prior to the development of models. While the primary purpose of these mappings is to facilitate ODS, they also have the side benefit of providing formalized documentation of the model development process.

In ontology alignment, several methods for finding entities in two or more ontologies that are semantically equivalent (Ehrig et al. 2005) have been developed. Automated ontology alignment tools typically use concepts such as semantic similarity, taxonomical similarity, and lexical analysis to find semantically equivalent entities. Automated ontology alignment can be useful when attempting to align ontologies that contain obviously similar concepts, but when attempting to align two ontologies from different knowledge domains, such as an ontology for biochemistry and the DeMO ontology, automated ontology alignment tends to produce less than satisfactory results. For this reason the DeMOForge tool provides a facility allowing domain experts to create ontology mappings and relies less on ontology alignment tools to produce mappings. DeMOForge should be thought of more as an ontology mediation tool rather than as an ontology alignment tool as the mappings are typically more general than equivalence or subsumption and are therefore hard to establish in an automated fashion. Even though mappings are established manually using the tool, they are maintained by the system and can be reused.

A developer wishing to create ontology mappings using DeMOForge is presented with a graphical user interface that allows them to select a domain ontology. Once selected, the classes of this ontology along with those of the DeMO ontology will be displayed side-by-side as hierarchical trees. The developer will then select a class from each ontology to be mapped and direct the tool to create a mapping between the two classes. Mappings are not necessarily one-to-one, since single class in one ontology may map to multiple classes in another. Once a mapping for a particular DeMO class has been created, it will be stored in the classes' has-DomainMapping property. Each subclass of the DeMO ModelComponent and DeModel classes inherits a copy of this property, enabling them to store mapping rules and version information describing mappings between the DeMO class and domain ontology classes. For example, the HybridFunctionalPetriNet class of DeMO may be mapped to the Pathway class of ReactO indicating that a pathway in the biochemical domain is to be modeled as a HFPN in the DES modeling domain. Since DeMO's HybridFunctionPetriNet class is a subclass of DeModel, it will have a has-DomainMapping property into which the mapping information will be placed.

In addition to mapping classes between ontologies, DeMOForge is also capable of mapping properties to classes. Tables 1 and 2 below show the mappings between ReactO and DeMO that were created in order to facilitate the simulation of biochemical pathway models. Table 1 shows the class to class mappings, while Table 2 shows the property to class mappings.

Table 1: Mappings between corresponding ReactO and DeMO classes

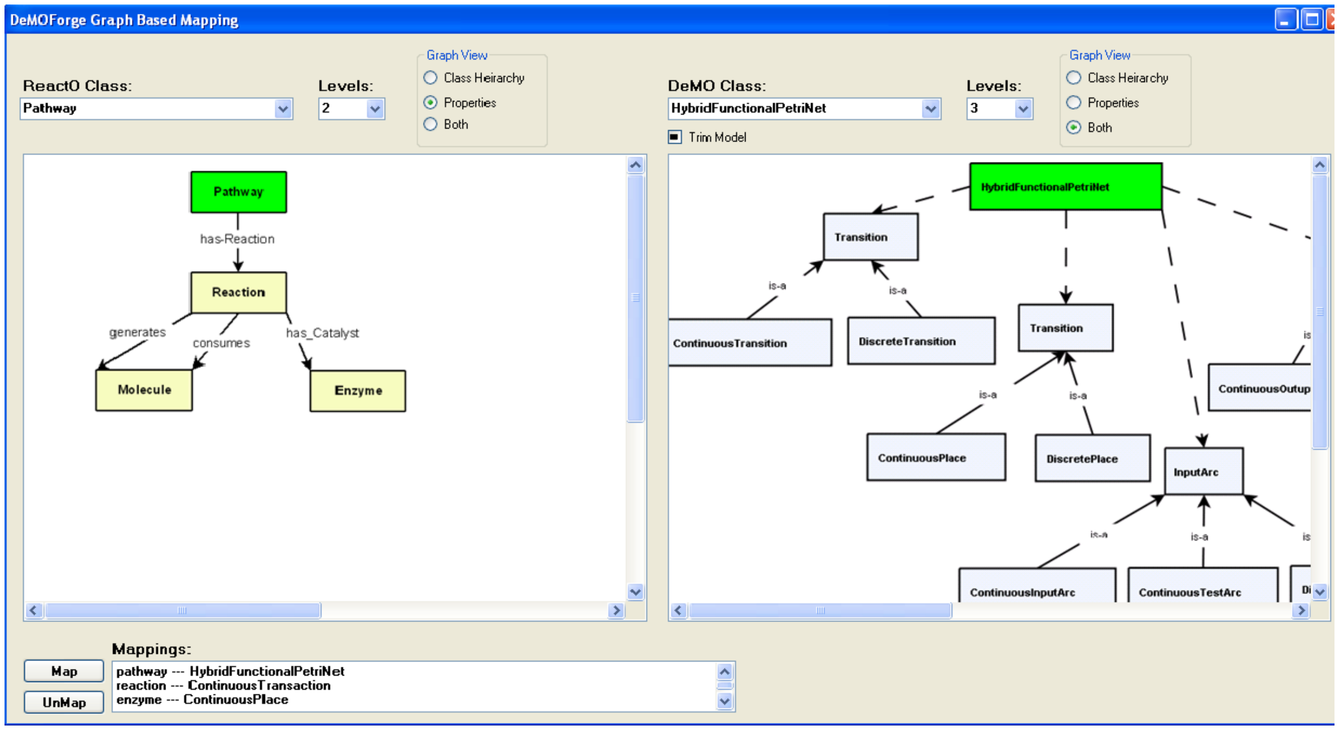
ReactO Ontology class	DeMO Ontology class
Pathway	HybridFunctionalPetriNet
Reaction	ContinuousTransition
Molecule	ContinuousPlace
Enzyme	ContinuousPlace

Table 2: Mappings between corresponding ReactO properties and DeMO classes

ReactO Ontology property	DeMO Ontology class
Reaction **consumes** Molecule	ContinuousInputArc
Reaction **generates** Molecule	ContinuousOutputArc
Reaction **… has-Catalyst** Enzyme	ContinuousTestArc

In order to create the mappings necessary to support the development of simulation models, users need to have some knowledge of both the domain ontologies and DeMO. Given that modelers developing simulations for a particular application domain may not have an in-depth knowledge of DES modeling techniques, DeMOForge provides features to assist the user in determining which classes need to be mapped. We will use the mapping of ReactO classes and properties used to represent a biochemical pathway to DeMO classes used to represent an HFPN to illustrate some of these features. Since it is more intuitive to represent an ontology as a graph than as a series of classes and properties, the DeMOforge mapping module can display ontologies as directed graphs. The user may select the classes or properties that are to be mapped by clicking on graph nodes or edges and directing the tool to map the selected components of the two ontologies. DeMO describes modeling techniques in a formal way that is recognizable to those acquainted with DES formalisms, and while this is necessary in order to provide a through and meaningful description of modeling techniques, it can make it difficult for those not well versed in DES formalisms to create accurate mappings. Because of this, the mapping module by default trims the ontology to display only those classes and properties necessary to give the user an intuitive view of a selected modeling technique. For example, the formal definition a PN is described as a tuple consisting of several sets, but the modeler will most likely think of a PN as a graph with two types of nodes (places and transitions) connected by arcs. Viewing the trimmed version of the DeMO HFPN graph in Figure 4, the user can more easily recognize the transitions, places and arcs associated with the intuitive definition of an HFPN. If the user desires to view the ontology untrimmed, the tool gives them the option to do so.

Once a few key mappings have been created, the tool can generate suggestions for additional mappings based on current mappings. For example, if the ReactO Molecule class has been mapped to the DeMO ContinuousPlace class and the ReactO Reaction class has been mapped to the ContinuousTransition class, then the tool can determine that the ReactO 'consumes' property should be mapped to either the ContinuousInputArc class or the ContinuousOutputArc class. The suggested mappings are presented to the user by highlighting the appropriate classes and properties in the graphs, as seen in Figure 5.

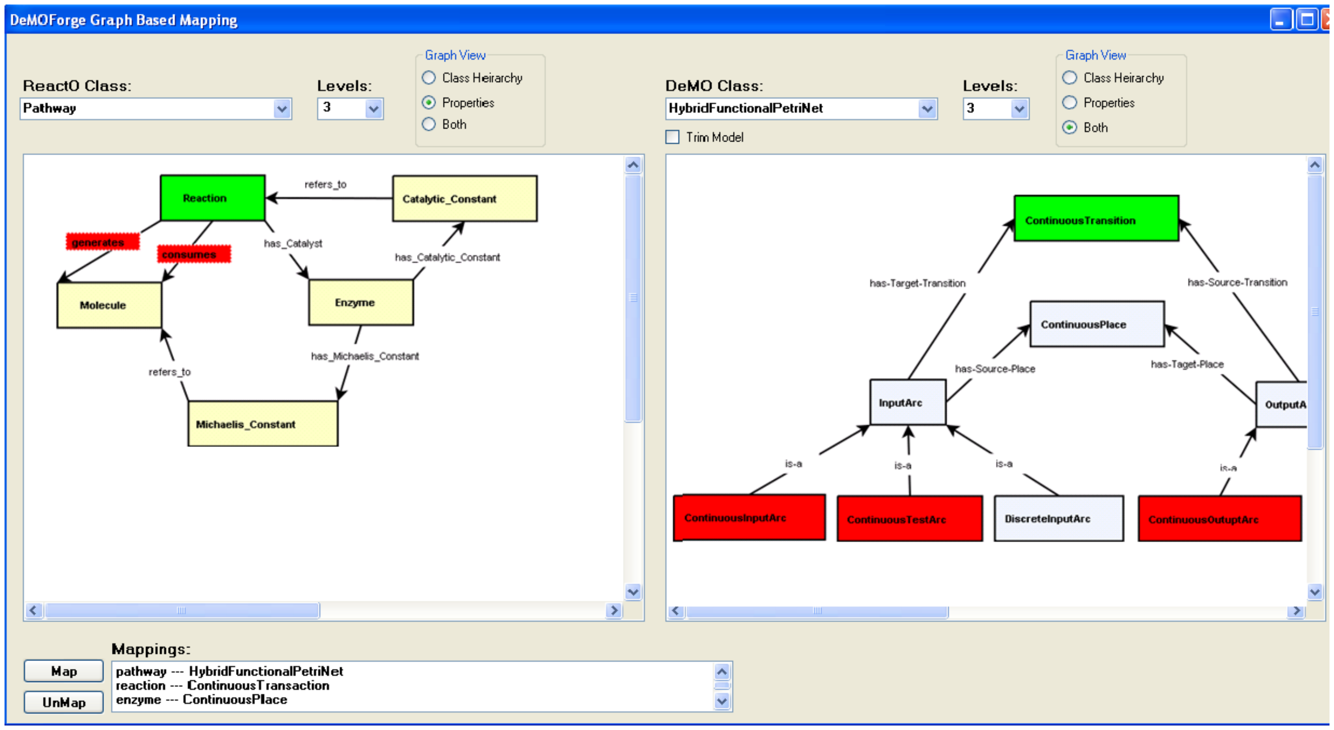

DeMOForge Graph Based Mapping

ReactO Class: Pathway
Levels: 2
Graph View: Class Heirarchy [✓ Properties] Both

DeMO Class: HybridFunctionalPetriNet
Levels: 3
Graph View: Class Heirarchy Properties [✓ Both]
[✓ Trim Model]

Mappings:
```
pathway --- HybridFunctionalPetriNet
reaction --- ContinuousTransaction
enzyme --- ContinuousPlace
```

Figure 4: DeMOForge Graph Based Mapping tool, showing ReactO Pathway (left) and HybridFunctionalPetriNet (right)


DeMOForge Graph Based Mapping

ReactO Class: Pathway
Levels: 3
Graph View: Class Heirarchy [✓ Properties] Both

DeMO Class: HybridFunctionalPetriNet
Levels: 3
Graph View: Class Heirarchy Properties [✓ Both]
Trim Model

Mappings:
```
pathway --- HybridFunctionalPetriNet
reaction --- ContinuousTransaction
enzyme --- ContinuousPlace
```

Figure 5: The classes and properties highlighted in red indicate mapping suggestions made by the DeMOForge

4.2 Transformation and Generation Phases

The Transformation Module of the DeMOForge tool translates instances from domain ontologies into DeMO instances that are used to represent a discrete-event simulation model. In order to illustrate this, we will use a particular example in which a portion of an N-Glycan biosyntheses pathway as represented via ReactO instances is transformed into DeMO instances which are used to represent the pathway as a HFPN. The pathway is represented using instances of ReactO classes shown in Figure 4.

For example, a developer may create instances of ReactO classes to represent an N-Glycan pathway. The DeMOForge transformation tool can use these instances along with the mappings shown in Table 1 and Table 2 above to create instances of DeMO classes which taken together can be viewed as a conceptual HFPN model of the same pathway. The tool makes use of SWRL rules to accomplish the transformation. The steps below give a brief outline of the process.

1. SWRL queries will be generated and used to retrieve the ReactO instances that represent the pathway model.
2. Previously defined mappings are used to create DeMO instances which correspond to the ReactO instances retrieved in step 1.
3. The structure of the DeMO HFPN formalism is captured, and SWRL rules are generated according to the formalism.
4. The SWRL rules created in step 3 are used to create relationships between the DeMO instances so that they form a conceptual HFPN model of the pathway.

After the transformation is complete, the conceptual HFPN model can be used generate an executable simulation model.

The generation module of DeMOForge uses a category specific code generator to translate the DeMO instances created during the transformation phase into an executable simulation model. In the case of the N-Glycan pathway example discussed earlier, an executable HFPN model for the JSIM simulation environment (Nair et al. 1996) was produced by a generator using the following steps: (1) The OWL 2.2.0 API (Horridge et al. 2007) was used to read through the DeMO instances and populate a data structure with a graph based representation of the model. (2) The scenario management component of the generator determined that additional information required for the particular scenario under which the simulation was to be executed, and it prompted the user for the location of sources from which data was to be retrieved. (3) The information collected in the previous two steps was used to produce a JSIM based HFPN model. The runtime animation of the model is shown in Figure 6.

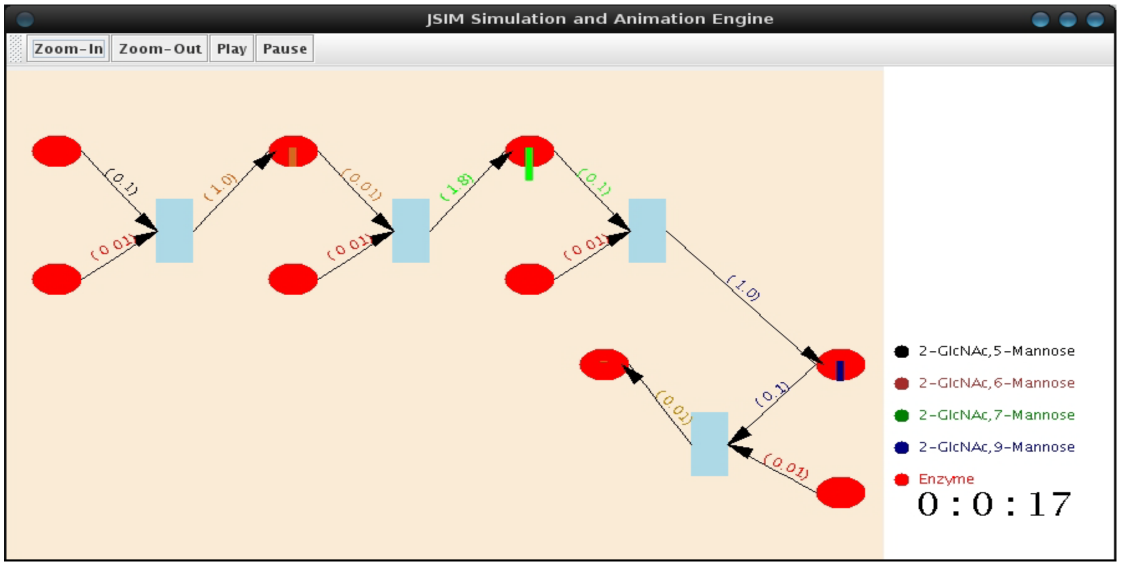


Figure 6: JSIM simulation environment’s animation of N-Glycan pathway simulation

5 CONCLUSIONS

This paper overviews ways in which the DeMO ontology may be used to support interoperability, and it presents an ODS methodology that makes practical use of the interoperability. The methodology uses the DeMOForge tool to accomplish end-to-end ODS and which can expedite the development of simulation models via the reuse of application knowledge as well as prior related simulation models. It does so by using domain ontology instances, which taken together can be thought of as a conceptual model of a system, and transforms them into DeMO instances associated with a specific DES modeling technique. These instances are then translated into a category specific executable simulation model. In accomplishing end-to-end ODS, the methodology supports interoperability between domain ontologies and a DES modeling ontology via ontology mediation, and it uses a rule language (SWRL) to accomplish the actual data interchange between ontologies. The interchange of data between the ontologies requires that mappings between corresponding concepts in the ontologies be established. The methodology allows the mappings to be created visually using directed graphs to represent the ontologies. It also trims the graphs, where possible, to make mapping selections more straight forward, and uses previously defined mappings to make suggestions for future mappings. A side benefit includes the ability to use the mapping instances stored in DeMO as a basis for documenting the model development process. In particular, our ODS methodology provides a thorough view of how the system under study is modeled.

ACKNOWLEDGEMENTS

This research is supported in part by the Integrated Technology Resource for Biomedical Glycomics (5 P41 RR18502-02) funded by the National Institutes of Health - National Center for Research Resources.

REFERENCES

Albertsen, T. and E. Blomqvist. 2007. Describing Ontology Applications. In *Proceedings of the 4th European conference on The Semantic Web: Research and Applications*, 549-563. Berlin, Germany: Springer-Verlag.

Alla, H. and R. David. 1998. Continuous and Hybrid Petri Nets. In *Journal of Circuits Systems Computers*, 8(1): 156-188.

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. The Semantic Web, In *Scientific American*, 284(5): 35-43.

Bruijn, J., M. Ehrig, and C. Feier. 2006. Ontology mediation, merging and aligning. In *Semantic Web Technologies: Trends Research and Ontology-based Systems*. Chichester, UK: John Wiley and Son's.

Benjamin, P. and M. Graul. 2006. A Framework for Adaptive Modeling and Ontology-driven Simulation. In *Proceedings of the SPIE, Enabling Technologies for Simulation Science X*, 6227: 5-5.

Dahmann, J., M. Salisbury, P. Barry, and P. Blemberg. 1999. HLA and beyond: Interoperability challenges. In *Simulation Interoperability Workshop*. Orlando, Florida.

Ehrig M., S. Staab, and Y. Sure. 2005. *Framework for Ontology Alignment and Mapping*. Available via <http://www.uni-koblenz.de/~staab/Research/Publications/2006/alignmentJournal-submitted.pdf>. [accessed June 4, 2009].

Gruber, G. 1993. A Translation Approach to Portable Ontology Specifications. In *Knowledge Acquisition.* 5(2): 199-200. London, UK: Academic Press Ltd.

Heiler, S. 1995. Semantic Interoperability. In ACM *Computing Surveys*, 27(2). 271-273. New York, New York: ACM.

Heinrich, R. and S. Schuster. 1998. The modeling of metabolic systems. Structure, control and optimality. In *BioSystems*, 47(1-2): 61-77.

Horridge, M., S. Bechhofer, and O. Noppens. 2007. Igniting the OWL 1.1 Touch Paper: The OWL API. In *CEUR Proceedings of OWL Experiences and Directions Workshop 2007*, 278: 1-9.

Horrocks, I., P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. 2004. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Available via <http://www.w3.org/Submission/SWRL>. [accessed June 2, 2009]

Kanehish, M., and S. Goto. 2000. KEGG: Kyoto Encyclopedia of Genes and Genomes. In *Nucleic Acids Research*, 28(1): 27-30. Oxford, UK: University of Oxford Press.

Lacy, L. W. 2006. *Interchanging Discrete-Event Simulation Process-Interaction Models using the Web Ontology Language – OWL*. Ph.D. Dissertation, Department of Industrial Engineering, University of Central Florida, Orlando, Florida.

Matsuno, H., A. Doi, Y. Hirata, and S. Miyano. 2001. XML documentation of Biopathways and their simulations in Genomic Object Net. In *Genome Informatics*, 12: 54-62.

Miller, J., G. Baramidze, and P. Fishwick. 2004. Investigating Ontologies for Simulation and Modeling. In *Proceedings of the 37th Annual Simulation Symposium*: 55-71. Arlington, Virginia.

Nair, R., J. A. Miller, and Z. Zhang. 1996. A Java-Based Query Driven Simulation Environment. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 786-793. Piscataway, New Jersey: IEEE, Inc.

Page, E. H. and R. Briggs. 2004. Toward a Family of Maturity Models for the Simulation Interconnection Problem. In *Proceedings of the 2004 Spring Interoperability Workshop*. IEEE CS Press.

Petri, C. 1966. Communication with Automata. Technical Report No. RADC-TR-65-377, Griffiss Air Force Base, Rome, New York.

Petty, M., and E. Weisel. 2003. A Composability Lexicon. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*. IEEE CS Press.

Reddy, V., M. Mavrovouniotis, and M. Liebman. 1993. Petri net representations in metabolic pathways. In *Proceedings of the First International Conference on Intelligent Systems*, 328-336. Menlo Park, California: IAAA Press.

Silver, G., O. Hassan, J. Miller. 2007. From Domain Ontologies to Modeling Ontologies to Executable Simulation Models. In *Proceedings of the 2007 Winter Simulation Conference*. December 2007, ed. S. G. Henderson, B. Biller, M. H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1108-1117. Piscataway, New Jersey: IEEE, Inc.

TEO, Y. and C. SZABO. 2008. CODES: An Integrated Approach to Composable Modeling and Simulation. In *Proceedings of the 41st Annual Simulation Symposium*, 103-110. Washington, DC: IEEE Computer Society.

Thomas, C., A. Sheth and W. York. 2006. Modular Ontology Design Using Canonical Building Blocks in the Biochemistry Domain. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*. IOS Press.

Tolk, A., 2003. The Levels of Conceptual Interoperability Model. In *Proceedings of the 2003 Simulation Interoperability Workshop*, 14-19. IEEE CS Press.

Tolk, A., 2004. Composable Mission Spaces and M&S Repositories – Applicability of Open Standards. In *Proceedings of the 2004 Spring Simulation Interoperability Workshop.* IEEE CS Press..

Tolk, A., 2005. Evaluation of the C2IEDM as an Interoperability-Enabling Ontology. In *European Simulation Interoperability Workshop*. ACM Press.

Yilmaz, L. and S. Paspuleti. 2005. Toward a Meta_level Framework for Agent-Supported Interoperation of Defense Simulations. In *The Journal of Defense Modeling and Simulation*. 2(3): 161-175. San Deigo, California: SCS.

AUTHOR BIOGRAPHIES

GREGORY A. SILVER is an Assistant Professor of Computer Information Systems at Anderson University and a PhD Candidate in Computer Science at the University of Georgia. Mr. Silver received his M.S. degree in Computer Information System from Georgia State University in 1996. His research interests include simulation, Web services. His email address is <gsilver@andersonuniversity.edu>.

KUSHEL RAI BELLIPADY is a Master's student in the Department of Computer Science at The University of Georgia. His research interests include ontology alignment and mediation and the use of ontologies for automatic web service discovery and composition. His email address is <kushel@cs.uga.edu>.

JOHN A. MILLER is a Professor of Computer Science at the University of Georgia and has also been the Graduate Coordinator for the department for 9 years. His research interests include database systems, simulation, Web services and bioinformatics. Dr. Miller received a B.S. in Applied Mathematics from Northwestern University in 1980 and an M.S. and Ph.D. in Information and Computer Science from the Georgia Institute of Technology in 1982 and 1986, respectively. During his undergraduate education, he worked as a programmer at the Princeton Plasma Physics Laboratory. Dr. Miller is the author of over 140 publications covering all areas of his research interests. His email address is <jam@cs.uga.edu>.

KRYS J. KOCHUT is a Professor and former Head of Computer Science at the University of Georgia. He received a Ph.D. in Computer Science from Louisiana State University in 1987. His research interests include workflow, Web services, Semantic Web, databases and bioinformatics. His is the author of numerous research papers on these topics including recent publications in major Semantic Web conferences. In particular, his recent work on Brahms and SPARQLer that provide a very fast implementation of an extended SPARQL Semantic Web query language are becoming well known. His email address is <kochut@cs.uga.edu>.

WILLIAM YORK is Associate Professor of Biochemistry and Molecular Biology, and Adjunct Associate Professor of Computer Science and Plant Biology at the University of Georgia. He received his B.A. in molecular, cellular and developmental biology in 1978 at the University of Colorado and his PhD. in biochemistry and molecular biology in 1996 from the University of Georgia. He was senior research chemist at the Complex Carbohydrate Research Center at the University of Georgia from 1985 to 1996. Dr. York's diverse research interests include the development and application of spectroscopic and computational methods for the analysis of complex carbohydrates, the development of bioinformatics tools to study the roles of carbohydrates in living systems, and the use of these tools to develop realistic models describing the assembly and morphogenesis of the "primary cell walls" of higher plants. His email address is <will@ccrc.uga.edu>.