Ontology–based Representation of Simulation Models

Katarina Grolinger, Miriam A. M. Capretz
Department of Electrical and Computer Engineering,
Faculty of Engineering
The University of Western Ontario
London, ON, Canada N6A 5B9
{kgroling, mcapretz}@uwo.ca

José R. Marti, Krishan D. Srivastava
Department of Electrical and Computer Engineering,
Faculty of Applied Science
The University of British Columbia
Vancouver, BC, Canada V6T 1Z4
jrms@ece.ubc.ca, kd@interchange.ubc.ca

***Abstract*—Ontologies have been used in a variety of domains for multiple purposes such as establishing common terminology, organizing domain knowledge and describing domain in a machine-readable form. Moreover, ontologies are the foundation of the Semantic Web and often semantic integration is achieved using ontology. Even though simulation demonstrates a number of similar characteristics to Semantic Web or semantic integration, including heterogeneity in the simulation domain, representation and semantics, the application of ontology in the simulation domain is still in its infancy. This paper proposes an ontology-based representation of simulation models. The goal of this research is to facilitate comparison among simulation models, querying, making inferences and reuse of existing simulation models. Specifically, such models represented in the domain simulation engine environment serve as an information source for their representation as instances of an ontology. Therefore, the ontology-based representation is created from existing simulation models in their proprietary file formats, consequently eliminating the need to perform the simulation modeling directly in the ontology. The proposed approach is evaluated on a case study involving the I2Sim interdependency simulator.**

Keywords-Ontology; Simulation Model; Ontology-based Model; Semantic Integration

I. INTRODUCTION

Ontologies are frequently associated with the Semantic Web where computers are capable of analyzing the content, meaning and semantics of the data and performing the reasoning upon the content. Other ontology applications include data integration, application integration and interoperability, knowledge management, machine learning, information extraction, information browsing and navigation.

Simulation domain exhibits a number of similar characteristics to those fields including heterogeneity in the simulation domain, vocabulary, representation and semantics. However, the application of ontology to the field of simulation is still in its infancy and primarily contained within the research community.

The simulation heterogeneity is largely caused by its application in a variety of different domains including critical infrastructures, medicine, learning and chemical engineering. Consequently, a number of software simulation packages or simulation engines exist for the support of computer simulations in those domains [1]. Commonly, simulation packages are application-oriented, designed for the use in a specific domain, hence they apply diverse modeling approaches, different technologies, domain specific terminologies and store simulation models and results in a variety of formats. This diversity of application-oriented simulation engines presents a challenge for comparing simulation models and results, reusing and sharing existing models, as well as querying and making inferences.

The objective of this work is to address the following challenges of the application-oriented simulation approach:

- The extraction of specific information from model files or from simulation results is not straightforward. Simulation packages may provide basic information, nevertheless, the extraction of more detailed or specific summary information becomes demanding.
- The comparison between models of a single simulation engine or different engines is difficult. Typically the comparison relies on the simulation engine to provide the means for comparing specific pairs of model files.
- The comparison between results of different simulation runs of the same simulation engine or different engines is a challenging endeavor. Simulation packages focus on providing performance measures for a single simulation run while the comparison between simulation runs often requires external tools and a significant manual effort.

As a solution, this paper proposes the representation of domain simulation models as instances of Simulators' Ontologies. By using the same formalism to represent various simulation models, we place them on the same platform, thus enabling a simplified comparison. Moreover, ontology-based representation allows for inquiries with ontology querying languages and inferences with ontology reasoners. The proposed approach uses existing models in the simulation engine proprietary file formats as the foundation for the creation of its ontology-based representation.

The remainder of the paper is organized as follows: Section II reviews related works, the proposed system is portrayed in Section III, while Section IV depicts a case study. Finally, the conclusions and future work are presented in Section V.

II. RELATED WORKS

Ontology can be described as an abstract, machine-readable model of a phenomenon that identifies the relevant concepts of that phenomenon as well as the relations among them. Furthermore, ontologies represent a way of establishing common terminology, organizing domain knowledge and representing this information in a machine-readable form. The potential use of ontologies in simulation and modeling is explored by Lacy and Gerber [2]. From the perspective of these authors, ontologies are beneficial in simulation and modeling through the formalization of semantics, the ability to query and inference, and the sharing and reuse of developed models.

Studies that are especially relevant to our research are related to the use of ontologies to represent real world scenarios for the simulation purposes such as Tofani et al. [3], Miller et al. [4] and Silver et al. [5].

Tofani et al. [3] use the ontology framework to model the interdependencies among critical infrastructures (CI). Their proposed framework consists of three ontologies: WONT (World ONTology) contains concepts and relations that are common across CI domains; IONT (Infrastructure ONTology) extends WONT to represent the knowledge of specific CIs and FONT (Federation ONTology) enables modeling relations among different infrastructures. The CI network is modeled twice: as instances of the ontology and in the simulation language of the domain. The mapping between ontology representations and simulation models is established manually.

Miller et al. [4] investigate the development requirements and benefits of ontologies in discrete event simulation (DES), and consequently, these authors present the Discrete-event Modeling Ontology (DeMO). The proposed DeMO consists of four main classes: DeModel, ModelComponent, ModelMechanism and ModelConcept. DeModel is composed of ModelComponents and activated by the ModelMechanism, while the ModelConcepts serve as a terminology upon which other classes are built. The main challenges in building DeMO, or a similar ontology for simulation and modeling, are twofold [6]; firstly, it needs to be domain-independent, as a DES can model any domain. Secondly, since simulation formalisms are founded in mathematics and statistics, the DES ontology should be based upon the ontologies of those domains.

Silver et al. [5] represent simulation models as instances of the extended DeMO PIModel (Process Interaction Model). In the proposed approach, reality is first represented as instances of the DeMo PIModel ontology. Subsequently, these DeMo PIModel instances are transformed to XPIM (Extensible Process Interaction Markup) instances, which are then translated to a JSIM (Java-based SIMulation) model.

Benjamin and Akella [7] use ontologies to facilitate semantic interoperability and information exchange between simulation applications. The ontology models for each simulation application domain are extracted from textual data sources, such as requirements and design documents. Subsequently, the established ontology mappings represent the translation rules for the ontology-driven translator, which facilitates information sharing between simulation applications.

III. Simulation Models as Instances of an Ontology

This section describes the fundamentals of the proposed system: system architecture, relations and model hierarchies definition, querying ontology-based models and the process of creating ontology-based simulation models.

A. System Architecture

The ontology-based representation of simulation models has a layered architecture, as described in Fig. 1.

The top layer consists of the upper ontology, which contains generic concepts that are common for all simulation engines.

The next architecture layer, the Simulators' Ontologies layer, consists of ontologies that are specific to the actual simulators. Thus, the terminology matches that of the simulators, facilitating domain experts' understanding of ontologies as well as enabling the creation of the ontological representations from the simulators' models.

The third layer, the ontology-based simulation model layer, contains ontology-based simulation models that are represented as instances of Simulators' Ontologies. More specifically, each simulation model, usually contained in a simulation engine proprietary file format, is represented as an ontology-based model consisting of interconnected instances of the Simulator's Ontology. Different simulation models contained in distinct proprietary files correspond to the various models in this layer.

The bottom architecture layer, or rule layer, is optional and contains a rule engine. In particular, this layer is intended for situations when ontology-based specifications are not sufficient and additional expressiveness is required. Furthermore, this layer expresses design rules and guidelines to which the simulation model should conform.

B. Defining Relations Between Simulation Model Entities

In the ontology-based simulation model, entities are represented as instances of the Simulator's Ontology, while the relations among them are established by means of object properties. The description of the object properties is found in the upper ontology and the Simulators' Ontologies.

Fig. 2 presents a fragment of the upper ontology in RDF/XML syntax. In this ontology the *cell* indicates an entity that performs a function transforming inputs into outputs, *channel* is a mean of transporting entities between *cells* and/or *controls*, while *controls* are entities responsible for distributing the flow among channels. The two object properties, *hasInput* and its inverse *hasEndNode*, are included in the ontology fragment shown in Fig. 2. The domain of the *hasEndNode*

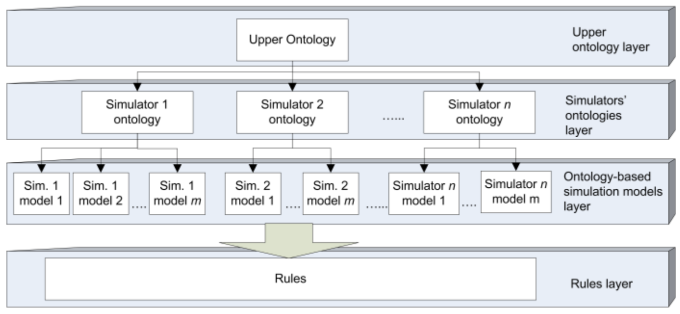
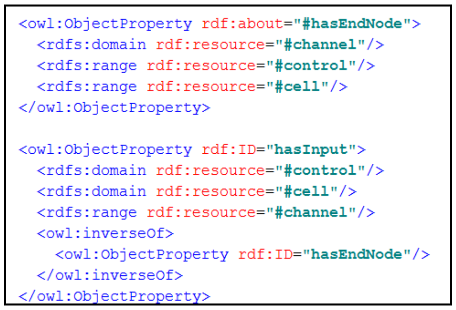


Figure 1. Architecture layers



```
<owl:ObjectProperty rdf:about="#hasEndNode">
  <rdfs:domain rdf:resource="#channel"/>
  <rdfs:range rdf:resource="#control"/>
  <rdfs:range rdf:resource="#cell"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasInput">
  <rdfs:domain rdf:resource="#control"/>
  <rdfs:domain rdf:resource="#cell"/>
  <rdfs:range rdf:resource="#channel"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasEndNode"/>
  </owl:inverseOf>
</owl:ObjectProperty>
```

Figure 2. Upper Ontology fragment

property is the *channel* while the range includes the *cell* and the *control*. Since the direction of object properties runs from the domain to the range, the inverse property enables the expression of relations in both directions. For instance, the *hasEndNode* property will express relations of the form, ‘channel 10 *hasEndNode* cell 24’, while its inverse, *hasInput*, will express the same relation as ‘cell 24 *hasInput* channel 10’. The direction that is used will be influenced by the manner in which the relation is expressed in the simulator’s model file.

C. *Defining Simulation Model Hierarchies*

Frequently, to facilitate modeling of complex systems, simulation packages provide the ability to divide models into hierarchies of sub-models [8] as illustrated in Fig. 3. To represent the model hierarchies in ontology, the proposed approach uses the *parentSystem* object property. For each child model, the *parentSystem* property links the model to its direct parent. The set of assigned *parentSystem* properties establishes the model hierarchy. A fragment of a hierarchy depicted in Fig. 3 is represented as: *modelE.parentSystem(modelB), modelB.parentSystem(modelA).* Since the sub-model entities do not belong to any of the Simulator’s ontologies classes, we establish a new class *parentSystem* to contain entities that serve as containers for the other entities.

The elements from the parent and the child models are interconnected since they form a single simulation model. The relation between elements from different hierarchy levels is established in the same way as the relation between entities of a single non-hierarchical model as described in Section III.B.

We considered creating a separate ontology for each sub-model which would import ontologies of all its child models. This would establish the ontology hierarchy matching with its equivalent simulation model hierarchy. Simulation sub-models can be as simple as two linked entities that would not warrant the formation of a separate ontology. Nevertheless, simulation models could contain a large number of sub-models resulting in a large number of ontologies for a single simulation model and thus causing maintenance challenges. Therefore, at this stage of our research, we use one ontology to represent one simulation model with all its sub-models.

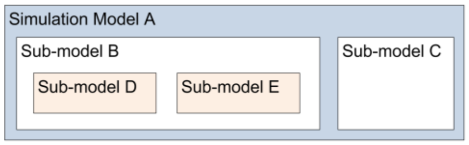


Figure 3. Simulation model hierarchy

D. *Querying Ontology-based simulation models*

Simulation models represented as instances of Simulator’s ontologies can be queried using different querying languages. We explore two different querying language styles: the RDF querying language, SPARQL [9], and the ontology querying language, SQWRL [10].

Since SPARQL is the W3C recommendation for querying RDFs [9] and OWL can be serialized as an RDF, SPARQL can be used to query ontology-based simulation models. However, as SPARQL is not an ontology querying language, it ignores inferences imposing limitations on querying, as will be shown in the case study. This drawback can be overcome by using a genuine ontology querying language such as SQWRL (Semantic Query-Enhanced Web Rule Language), which is a SWRL-based (Semantic Web Rule Language) language for querying ontologies. Accordingly, in the presented scenario, we use both the SPARQL and SQWRL approaches, identifying their advantages and disadvantages in regards to querying ontology-based simulation models.

E. *Creating Ontology-based Simulation Models*

In the proposed approach, the simulation models represented in the domain simulation engine environment serve as an information source for the representation of models as instances of an ontology. While the approaches proposed by Tofani et al. [3] and Silver et al. [5] also describe simulation models as instances of ontologies, these approaches perform modeling directly in the ontology, which is then mapped or transformed to a different representation. In contrast, our approach uses existing domain simulation models as an origin for the creation of its own ontology-based representation. The advantages of this approach include:

- The use of existing, domain-specific models.
- The ability of domain experts to create new models in the simulation engine to which he/she is accustomed rather than creating models directly as an ontology.
- The use of proven domain simulators for simulation execution.
- There is no need for manual mapping between simulators and ontology models.

Fig. 4 portrays our approach for the creation of an ontology-based simulation model representation. Specifically, the Transformation Engine inputs consist of the Simulator’s Ontology and the simulator’s model in the domain simulation engine representation. The Simulator’s Ontology is simulator-specific, while the simulator’s models are model-specific, as each model is stored in a separate file.

The Simulator’s Ontology is read by the Ontology Reader, which is independent of the simulation engine. While ontologies are stimulator-specific, they are always represented using the same ontology language, thus allowing for a simulator-independent reader. In particular, the Ontology Reader is responsible for acquiring information about simulator’s classes and their properties. Classes are relevant concepts from a specific domain, such as *channel* and *cell*; they can be perceived as sets of individuals, which include actual objects from the domain, such as a set of all individual

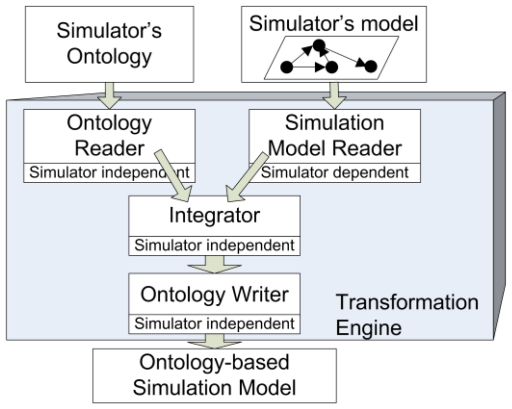


Figure 4. Ontology-based model creation from simulator's model

channels in a distribution network. Although the Simulators' Ontology does not contain individuals, they will be extracted by the Transformation Engine. Moreover, the Ontology Reader is responsible for reading properties, including data properties and object properties. Data properties connect individuals with literals; an example of a data property is the capacity of a specific storage cell. Conversely, object properties connect pairs of individuals such as the *hasInput* property, which links the cell with the channel in the statement, 'Cell *x hasInput* Channel *y*'.

The second transformation source, the simulator's model, is read by the Simulation Model Reader. Since the format of the simulator's model depends on the specific simulator, a separate Simulation Model Reader has to be created for each simulator whose model requires transformation to an ontology-based representation. However, once a Simulation Model Reader is created for a specific simulator, the reader can be used to transform any model represented in that format. The architecture of the Simulation Model Reader depends on the model being read. For instance, the reader can utilize the simulator's API interface, directly read the model file or use external model readers.

The Integrator uses the data received from the Ontology Reader and the Simulation Model Reader for creating the ontology-based model representation. Specifically, the Integrator receives information about the simulator's classes from the Ontology Reader. For each class, the Integrator obtains knowledge about its individuals from the Simulation Model Reader. When an Integrator identifies individuals, it also obtains values for their data properties. After acquiring information about all individuals of all classes and their data properties, the Integrator proceeds to determine the object properties. Since object properties connect individuals of the same or different classes, all individuals must be determined before the object properties are defined.

Subsequently, the Integrator sends information about classes, individuals, data properties and object properties to the Ontology Writer, which writes an ontology-based simulation model representation. Rather than recreating classes, the output ontology imports the Simulators' Ontology to acquire domain-relevant concepts and properties. Then, individuals and property values are created from the information received via the Integrator, and the output is recorded in an ontology language such as OWL. Thus, the Ontology Writer is simulator-independent, as its purpose is to write ontologies from the Integrator's information.

Consequently, the Simulation Model reader is the only Transformation Engine component that is simulator-dependent. However, this reader can be replaced with a different simulator's reader in order to represent that specific simulator's model in an ontology-based representation.

IV. Case Study

This work is part of the CANARIE-sponsored Disaster Response Network Enabled Platform (DR-NEP) project [11]. The project aims to improve the capability to prepare for and respond to large disasters. In particular, disaster modeling and simulation play a major role in the project, with a special focus on critical infrastructure (CI) interdependency simulation. Therefore, the proposed ontology-based representation of simulation models is evaluated using I2Sim [12] infrastructure interdependencies simulator.

The proposed approach is generic, as it is independent of any simulation engine; however, its implementation requires the creation of engine-specific Simulator Ontologies and the Simulation Model Reader. Therefore, the I2Sim ontology is created; the ontology design and the mapping to the upper ontology are presented in [13]. Since I2Sim is based on MATLAB's Simulink engine, the Transformation Engine inputs include the I2Sim ontology and the I2SIm model, which is stored in the Simulink style .mdl file. The Transformation Engine was implemented using the following technologies:

- The Ontology Reader and the Ontology Writer are implemented using Protégé OWL API [14] and Java 1.6.
- OWL is used for the representation of the ontology-based simulation models.
- The Integrator is implemented using Java 1.6.
- The I2Sim Simulation Model Reader uses the Simulink Java library from Technische Universität München [15].

A. *Ontology-based Representation of Simulation Models*

To explore ontology-based simulation models we used the I2Sim model developed as part of the DR-NEP project for the investigation of infrastructure interdependencies.

MATLAB's Simulink engine [8], upon which I2Sim is built, is an environment for multi-domain simulation and for dynamic and embedded systems. Simulink provides block libraries which can be customized to conform to a specific simulation domain. Complex models are managed by dividing models into hierarchies of sub-models. Accordingly, I2Sim builds upon Simulink by customizing Simulink blocks and providing entities specific for infrastructure interdependency simulation.

The I2Sim model that we used in this case study consists of several hierarchy levels. However, before transforming it to the ontology-based model we were not aware of the number of layers, the number of blocks or types of blocks used.

First, the I2Sim ontology was created [13] containing only I2Sim blocks as illustrated in Fig. 5(a). Subsequently, the

I2Sim simulation model was transformed to an ontology-based representation, which is depicted from the perspective of the Protégé ontology editor in Fig. 5(b). Specifically, the left part of the screen shows I2Sim classes, such as *i2sim_source*, and *production_cell*. As the *channel* class is selected, the middle part of the screen displays all of the individual channels from the I2Sim model. On the right side are displayed the object and data properties for the selected channel, *from_feed_water_pump_1_to_steam_house_3*. As the channels are not named in I2Sim, we have chosen to use *from_sourceNode_port_to_targetNode_port* as a channel naming pattern. The object properties *hasStartNode* and *hasEndNode* indicate that the selected channel starts from the *feed_water_pump* and ends at the *steam_house*. In the I2Sim ontology, *hasStartNode* and *hasEndNode* are asserted properties, as the I2Sim model specifies the channel start and end. The inverse properties, *hasInput* and *hasOutput*, are inferred, allowing for ontology querying in both directions.

Another significant I2Sim modeling concept for establishing network topology is the port concept. I2Sim entities such as the production cells can have several input and output ports. Each channel connects to a specific input and output port as identified by the port number. Therefore, in the ontology-based representation each channel has *hasInPort* and *hasOutPort* data properties, as illustrated in Fig. 5(b). The channel selected in the figure connects to the first *feed_water_pump* port and the third *steam_house* port.

Observing the classes from I2Sim ontology, Fig. 5(a), and from the ontology-based simulation model, Fig. 5(b), it can be noticed that the ontology-based model contains additional entries such as *minmax*, *product* and *fcn*. Initially, we expected the I2Sim model to have only I2Sim blocks. However, when the model was transformed to its ontology-based representation, many entities belonged to the *other* class.

Subsequently, we analyzed those entities and identified that they are Simulink blocks. Since I2Sim is founded on Simulink by customizing and extending Simulink blocks, it allows for the use of Simulink blocks in conjunction with the I2Sim blocks. Accordingly, our case study of the I2Sim model actually contained I2Sim and Simulink blocks. Therefore, we created the *non_i2sim* class and in the transformation process we allowed for the creation of *non_i2sim* sub-classes representing Simulink blocks categories used in the observed I2Sim model.

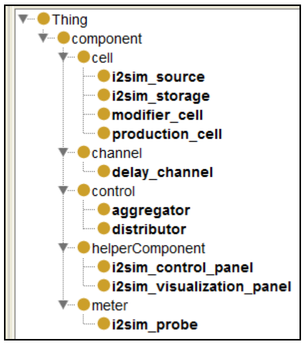
The ontology-based representation of the I2Sim model hierarchy is portrayed in Fig. 6. On the left part of the screen the *parentsystem* class is selected, while the segment to the right shows all entities that act as a container for the other elements. The selected entity *water_pump_1* is a child of the *water_pump_control* as indicated with the *parentSystem* object property. To simplify the illustration of hierarchies we have chosen to include the hierarchy chain in the entity name separating the hierarchy levels with a dash as in *steam-house-water_pump_control-water_pump_1*. The maximum number of hierarchy levels in the observed I2Sim model was three.

B. *Querying ontology-based representation*

A simulator's model that is represented as ontology instances can be queried using querying languages such as SPARQL or SQWRL. The use of these languages enables the extraction of specific information from the model. Since the number of entities in a system is one of the complexity indicators, after representing I2Sim models as instances of an ontology, we first wanted to obtain the number of instances in the ontology-based representation. However, the standard SPARQL included with Protégé does not support aggregate functions such as *count* and *avg*. Nevertheless, different implementations that support aggregates exist, such as ARQ [16]. Instead, we used SQWRL to identify the number of I2Sim and Simulink entities. I2Sim components are instances of the *component* class and its subclasses, while Simulink entities are instances of *non-i2sim* class and its subclasses:

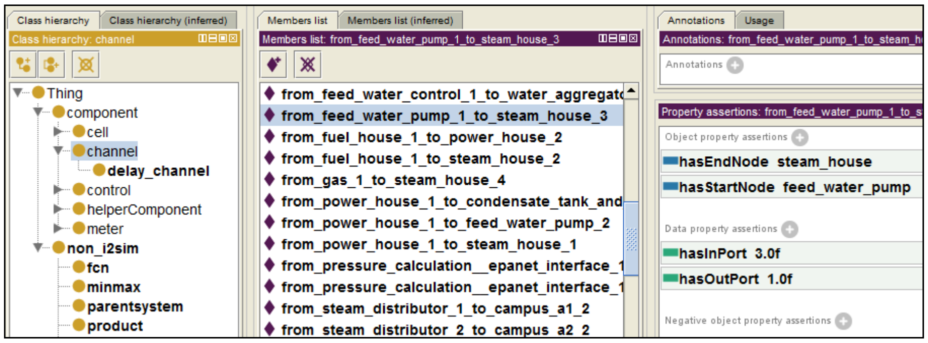
```
simupper:component(?c)→sqwrl:count(?c)
non_i2sim(?c)→sqwrl:count(?c)
```

Those two queries identified that the observed I2Sim model consists of 449 I2Sim components and 230 Simulink



```
▼ Thing
  ▼ component
    ▼ cell
        i2sim_source
        i2sim_storage
        modifier_cell
        production_cell
    ▼ channel
        delay_channel
    ▼ control
        aggregator
        distributor
    ▼ helperComponent
        i2sim_control_panel
        i2sim_visualization_panel
    ▼ meter
        i2sim_probe
```

a) I2Sim Ontology



Class hierarchy | Class hierarchy (inferred)

Class hierarchy: channel

```
▼ Thing
  ▼ component
    ► cell
    ▼ channel
        delay_channel
    ► control
    ► helperComponent
    ► meter
  ▼ non_i2sim
      fcn
      minmax
      parentsystem
      product
```

Members list | Members list (inferred)

Members list: from_feed_water_pump_1_to_steam_house_3

```
from_feed_water_control_1_to_water_aggregat
from_feed_water_pump_1_to_steam_house_3
from_fuel_house_1_to_power_house_2
from_fuel_house_1_to_steam_house_2
from_gas_1_to_steam_house_4
from_power_house_1_to_condensate_tank_and
from_power_house_1_to_feed_water_pump_2
from_power_house_1_to_steam_house_1
from_pressure_calculation__epanet_interface_1
from_pressure_calculation__epanet_interface_1
from_steam_distributor_1_to_campus_a1_2
from_steam_distributor_2_to_campus_a2_2
```

Annotations | Usage

Annotations: from_feed_water_pump_1_to_steam_h

Annotations

Property assertions: from_feed_water_pump_1_to_s

Object property assertions

```
hasEndNode  steam_house
hasStartNode  feed_water_pump
```

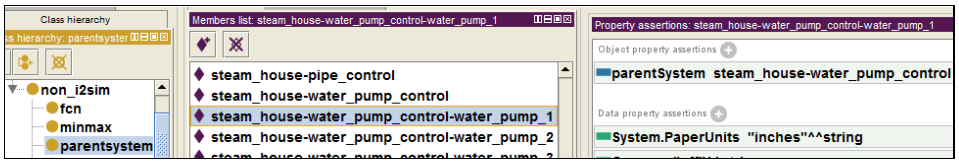
Data property assertions

```
hasInPort  3.0f
hasOutPort  1.0f
```

Negative object property assertions

b) Ontology-based representation of the I2Sim model

Figure 5. Ontologies from the Protégé editor



Class hierarchy

s hierarchy: parentsyster

```
▼ non_i2sim
    fcn
    minmax
    parentsystem
```

Members list: steam_house-water_pump_control-water_pump_1

```
steam_house-pipe_control
steam_house-water_pump_control
steam_house-water_pump_control-water_pump_1
steam_house-water_pump_control-water_pump_2
```

Property assertions: steam_house-water_pump_control-water_pump_1

Object property assertions

```
parentSystem  steam_house-water_pump_control
```

Data property assertions

```
System.PaperUnits  "inches"^^string
```

Figure 6. The hierarchy levels of the I2Sim model

components. Subsequently, since I2Sim model extensively uses hierarchical modeling, we identified each sub-system together with the number of elements in each sub-system using the following query:

```
owl:Thing(?c) ∧
i2sim:parentSystem(?c,?subsystem) ∧
sqwrl:makeSet(?s, ?c) ∧
sqwrl:groupBy(?s, ?subsystem) ∧
sqwrl:size(?count, ?s) →
sqwrl:select(?subsystem, ?count) ∧
sqwrl:orderByDescending(?count)
```

The first few rows from this query results with our ontology-based representation of the I2Sim model are displayed in Fig. 7. Since the sorting is performed on the number of entities, the first few rows indicate sub-models with the highest number of components.

To illustrate the difference of querying ontology-based simulation models using SPARQL and SQWRL we used a simple example of finding channels in the *steam_house-boiler_1* sub-model. In SPARQL this query is written as:

```
SELECT *
WHERE { ?c rdf:type :i2sim.channel.
?c i2sim:parentSystem :steam_house-boiler_1}
```

While in SQWRL the same query is expressed as:

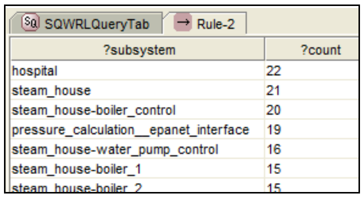
```
simupper:channel(?c) ∧
i2sim:parentSystem(?c, steam_house-boiler_1) →
sqwrl:select(?c)
```

The SPARQL query did not find any entities, while the SQWRL found the *steam_house-boiler_1-water_tank_tube*. This entity belongs to the *delay_channel* class. However, although the *delay_channel* is defined as *subClassOf channel*, SPARQL could not infer that *delay_channel* is also a *channel*, and thus, SPARQL did not identity this entity. Since SQWRL is an ontology querying language, it inferred that *delay_channel* is also a *channel* and therefore properly identified the entity.

V. Conclusions

Application-oriented simulation packages vary greatly in modeling approaches, technologies and vocabularies. However, this heterogeneity imposes several challenges, including the difficulty of comparison among simulation models of one or more simulation engines and the inability to query simulation models. This work proposes to solve those issues using ontology-based representations of simulation models where domain simulation models are represented as instances of Simulators' Ontologies. Existing domain simulation models in proprietary file formats are a foundation for this ontology-based representation.

The main benefits of using ontology-based representation



?subsystem	?count
hospital	22
steam_house	21
steam_house-boiler_control	20
pressure_calculation__epanet_interface	19
steam_house-water_pump_control	16
steam_house-boiler_1	15
steam_house-boiler_2	15

Figure 7. SQWRL query output from Protégé

for simulation models include: models from different simulation platforms are represented in a common manner, the models can be queried using ontology querying languages and inferences can be performed using ontology reasoners.

While we have focused on representing the simulation model as contained in a model file, this model is only a static part of the simulation. Accordingly, we plan to explore the possibility of using ontologies for representing the dynamic part of the running simulation. Furthermore, we will investigate the use of proposed ontology-based simulation models to facilitate information sharing among simulators of different domains.

Acknowledgment

Support for this work has been provided by Canada's Advanced Research and Innovation Network (CANARIE) and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] E. Abu-Taieh and A. El Sheikh. , "Commercial Simulation Packages: A Comparative Study", *International Journal of Simulation*, vol. 8, no. 2, pp. 66-76, 2007.

[2] L. Lacy and W. Gerber, "Potential Modeling and Simulation Applications of the Web Ontology Language - OWL", *Proc. Winter Simulation Conference*, vol. 1, pp. 265-270, 2004.

[3] A. Tofani, E. Castorinia, P. Palazzaria, A. Usovb, C. Beyelb, E. Romeb and P. Servilloc, "Using Ontologies for the Federated Simulation of Critical Infrastructures", *Proc. International Conference on Computational Science*, vol. 1, no. 1, pp. 2301-2309, 2010.

[4] J.A. Miller, G.T. Baramidze, A.P. Sheth and P.A. Fishwick, "Investigating Ontologies for Simulation Modeling", *Proc. 37th Annual Simulation Symposium*, pp. 55-63, 2004.

[5] G.A. Silver, L.W. Lacy and J.A. Miller, "Ontology Based Representations of Simulation Models Following the Process Interaction World View", *Proc. Winter Simulation Conference*, pp. 1168-1176, 2006.

[6] J.A. Miller and G. Baramidze, "Simulation and the Semantic Web", *Proc. Winter Simulation Conference*, pp. 2371-2377, 2005.

[7] P. Benjamin and K. Akella. , "Towards Ontology-Driven Interoperability for Simulation-Based Applications", *Proc. Winter Simulation Conference*, pp. 1375-1386, 2009.

[8] "Simulink - Simulation and Model-Based Design", http://www.mathworks.com/products/simulink/, 2011.

[9] E. Prud'hommeaux and A. Seaborne. , "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query/, 2008.

[10] M.J. O'Connor and A. Das. , "SQWRL: A Query Language for OWL", *OWL Experiences and Directions, 6th International Workshop*, 2009.

[11] "DR-NEP (Disaster Response Network Enabled Platform) project", http://drnep.ece.ubc.ca/index.html, 2011.

[12] H.A. Rahman, M. Armstrong, D. Mao and J.R. Marti, "I2Sim: A Matrix-Partition Based Framework for Critical Infrastructure Interdependencies Simulation", *Proc. Electric Power Conference*, pp. 1-8, 2008.

[13] K. Grolinger, M.A.M. Capretz, A. Shypanski and G.S. Gill, "Federated Critical Infrastructure Simulators: Towards Ontologies for Support of Collaboration", *Proc. IEEE Canadian Conference on Electrical and Computer Engineering, Workshop on Connecting Engineering Applications and Disaster Management*, 2011.

[14] "Protégé OWL API", http://protege.stanford.edu/plugins/owl/api/, 2011.

[15] "Simulink Library, Technische Universität München", http://conqat.in.tum.de/index.php/Simulink_Library, 2011.

[16] "ARQ - A SPARQL Processor for Jena", http://incubator.apache.org/jena/documentation/query/.