ONTOLOGY DRIVEN SIMULATION USING ONTOLOGY MEDIATION TECHNIQUES

by

KUSHEL RAI BELLIPADY

(Under the Direction of John A. Miller)

ABSTRACT

The need for interoperability arises in the discipline of simulation and modeling, particularly when the components of a simulation environment need to communicate with each other or communicate with outside entities including other simulation environments. Ontologies can be used to facilitate this interoperability between the different entities in the real world. This thesis describes ways to semi-automatically facilitate the interoperation between two ontologies and between a modeling ontology and specific simulation software packages. In the first stage, the interoperation is between any domain ontology that may contain a conceptual model describable in Discrete Event Simulation (DES) modeling terms, and the Discrete Event Modeling Ontology (DeMO). This is achieved through ontology mediation. In the second stage, the DeMO ontology is made to interoperate with specific simulation software packages in order to produce executable code that conforms to the software. The paper discusses methods to enable interoperation in this context.

INDEX WORDS: Interoperation, ontology mediation, simulation, DeMO, JSIM

ONTOLOGY DRIVEN SIMULATION USING ONTOLOGY MEDIATION TECHNIQUES

by

KUSHEL RAI BELLIPADY

B.E. Vishweshwaraiah Technological University, India, 2006

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2009

© 2009

KUSHEL RAI BELLIPADY

All Rights Reserved

ONTOLOGY DRIVEN SIMULATION USING ONTOLOGY MEDIATION TECHNIQUES

by

KUSHEL RAI BELLIPADY

Major Professor:	John A. Miller
Committee:	Krzysztof J. Kochut
	William York

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2009

ACKNOWLEDGEMENTS

I sincerely wish to acknowledge Dr. John A. Miller, the head of my thesis committee, for his constant support and approval. This thesis was possible solely due to his encouragement and technical advice. His Advanced Databases course helped me a great deal in propelling this thesis forward at a faster rate. I also wish to thank Dr. Krzysztof J. Kochut for his support and advice and for introducing me to quite a few interesting topics related to the semantic Web, Dr. Ismailcem Budak Arpinar and Dr. Prashant Doshi who introduced me to the world of semantic Web through their Database Systems and Enterprise Integration courses respectively. I am also very grateful to Dr. William York who agreed to be on my thesis committee on very short notice and for his valuable advice on the domain ontology that I am using in this thesis as a case study.

I also wish to thank Gregory Silver who helped me get a great start on this thesis, his constant advice on how to make this thesis better and for creating images of certain components of this thesis which I have used in this paper. Finally I would like to thank Sonu Swaika and Justin Martin, my classmates for helping me research on solutions to my thesis topic.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Software systems and services cater to specific needs and are inflexible in terms of collaborating with other systems to share and exchange knowledge. This is because each of these systems uses a vocabulary with an intended meaning known and interpretable only to them. However, if these systems used a shared vocabulary, then interoperation between them can be achieved and this could lead to many advantages such as collaboration and composability. One of the best and proven ways to enable interoperation between systems and agents is by using ontologies. An ontology represents knowledge in a formal and structured way and is also machine-readable, thereby making it easy for systems to use as a shared vocabulary. However, simply developing and maintaining ontologies are not sufficient to enable interoperation between these systems. Since there is no centralized authority overseeing the development of ontologies, there exist many different ontologies describing the same domain in different terms and their structural frameworks could be different too. Due to this, though systems ground their semantics within ontologies, it is still hard for them to interoperate. A way to overcome this roadblock is through ontology mediation. Since ontologies are written in a formal language and since many of them are decidable, it is possible to perform mediation between two ontologies that correspond to each other on a semantic level. Through ontology mediation, the knowledge of two or more systems can be combined, shared and utilized between them, thus enabling interoperation.

In recent times, the use of ontologies in modeling and simulation is being considered to enable interoperation between the components of simulation systems. Most simulation software packages today allow the user to create simulation models using graphical drag and drop schemes. For more flexibility, many tools such as MATLAB/Simulink [Ong, 1998], AutoMod [Roher, 2003] (which includes a simulation software package, a tool for experimentation and analysis and one for making AVI movies of built-in 3-D animation), Micro Saint [Bloechle and Schunk 2003] (supports discrete-event models and used to simulate real life processes) and Arena [Bapat and Sturrock, 2003] (can be used to simulate both discrete and continuous systems), allow users to develop simulation models using their specific simulation languages. However, this requires the domain experts to have sufficient knowledge about modeling and simulation. If there was a way for the domain ontologies, capturing the essential knowledge about conceptual domain models, to interoperate with the simulation software packages, then the domain experts will be relieved of the requirement for them to have in-depth knowledge about modeling and simulation. However, this would require that the simulation software package be able to interoperate with almost every domain ontology and this poses a very complex problem to the simulation software developer. If the domain ontology were to interoperate with a common modeling ontology and the modeling ontology with the simulation software, then it significantly drops the complexity on both sides, i.e., from the domain expert's point of view and from that of the simulation software. This interoperation between the two ontologies can be enabled through ontology mediation.

This thesis discusses ontology driven simulation aided by ontology mediation techniques. We have developed a system, DeMOForge, that uses the Discrete-event Modeling Ontology (DeMO)

as its modeling ontology, and it interoperates with the JSIM simulation software package to produce executable simulation code. Domain experts wishing to simulate the conceptual models from their domain ontologies can use DeMOForge to transform their models into discrete-event models in DeMO and run the simulation. DeMOForge provides a mapping tool and a transformation tool that enables interoperation between the domain ontologies and DeMO through ontology mediation. It also provides a code generator that interoperates with the DeMO ontology and the simulation software, JSIM, to produce executable simulation code that can be run using JSIM.

The rest of this thesis is organized as follows: Chapter 2 discusses background information related to this thesis, Chapter 3 discusses the problem statement and solution strategies explored and undertaken, Chapter 4 discusses the implementation of the DeMOForge tool and Chapter 5 concludes the thesis and discusses future work.

CHAPTER 2

BACKGROUND

2.1 Ontology

One of the shortest yet informative definitions of an ontology states that it "is an explicit specification of a conceptualization" [Gruber, 1993]. Conceptualization in this context refers to a particular view of the world chosen for representation. The representation is knowledge structured in the form of concepts belonging to a particular domain linked together with relationships. In other words, an ontology can be defined as a formal specification of this knowledge representation. Today, ontologies are used extensively all over the world to structure and represent knowledge in order for people and machines alike to use and reuse this knowledge for various purposes. Ontologies, in the view of modern science, consists of concepts or classes which are typically nouns describing tangible or intangible entities, relationships or properties which link these concepts to one another and other such entities that help in specifying, structuring and formalizing knowledge in a machine readable way. Ontologies have to be machine readable as they play a very important role in the interoperation of different systems, and they do this by sharing the knowledge that they represent in a standard way. Ontologies also help such systems and agents in grounding their beliefs and the actions that they perform [Singh and Huhns, 2005].

Ontologies are used in the fields of artificial intelligence, the Semantic Web, software engineering, biomedical informatics and information architecture to name a few. The most

common languages used to build, represent and express ontologies are the Resource Description Framework (RDF) [Miller and Schloss, 1997] and the Web Ontology Language (OWL) [McGuiness and Harmelen, 2002]. While RDF is restricted to defining resources, it is often used with the constructs (classes and subclasses, property domains and ranges) of RDF-Schema (RDFS) [Brickley and Guha, 1999] which help in structuring these resources. Ontology editors may be used to develop and maintain ontologies and Protégé [Noy et al., 2001] is one of the most popular ontology editors. In this thesis, we use ontologies written in OWL and developed in Protégé.

2.2 OWL

The Web Ontology Language (OWL) is a language for knowledge representation, recommended by the World Wide Web Consortium, primarily used to structure knowledge and write them in the form of ontologies. OWL is written in XML and it is the successor of Darpa Agent Markup Language and Ontology Inference Layer (DAML+OIL) [Horrocks et al., 2001]. OWL is a family of three sub-languages based on the expressiveness that each allows: OWL-Lite which is the least expressive, OWL-DL which is based on Description Logic and OWL-Full which is the most expressive of the three. OWL Lite is used when only a classification of concepts along with a few simple constraints are required, OWL-DL is for users who need maximum expressiveness while still being decidable [McGuiness and Harmelen, 2002] and OWL-Full though, being the most expressive, but is not decidable. Hence OWL-DL is the most popular sub-language and because it is based on Description Logic, it allows for consistency checks and making inferences on the knowledge. Popular Application Programmer Interfaces (API) used to work with OWL

are the Protégé-OWL API [Knublauch et al., 2004], the Jena API [Verzuilli, 2001] and the Manchester-OWL API [Bechhofer et al., 2003].

2.3 Modeling and Simulation and Discrete-Event Simulation (DES)

Modeling is the process of generating abstract, conceptual, graphical and/or mathematical models of real world systems. Modeling is used when it is infeasible to test the real system due to reasons such as cost and time. Simulation, in computer science, refers to the execution of such models that represent a conceptual framework describing a system, by a computer program in order to investigate the processes and behaviors of the system [Arsham, 1995].

Discrete-event simulation is used in scenarios where the processes and events within a system tend to occur as a chronological sequence, more specifically when significant changes which may affect state of the system, take place at discrete time instances [Park and Leemis, 2001]. The components of DES include clocks, events lists, random-number generators, statistics and ending conditions.

2.4 Discrete-Event Modeling Ontology (DeMO)

The Discrete Event Modeling Ontology (DeMO) is an ontology built and maintained by the Large Scale Distributed Information Systems (LSDIS) lab at the University of Georgia. DeMO is an ontology that encompasses the knowledge of the discrete-event simulation domain. The models in DeMO focus on how discrete states evolve over time. DeMO is useful to researchers who are involved in modeling and simulation [Miller et al., 2004]. DeMO is one of the primary components of Ontology Driven Simulation (ODS). DeMO is built using OWL-DL and its constructs. Below is a description of DeMO:

"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 and process, 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 the first few levels of the DeMO DeModel class hierarchy." [Silver et al., 2009]

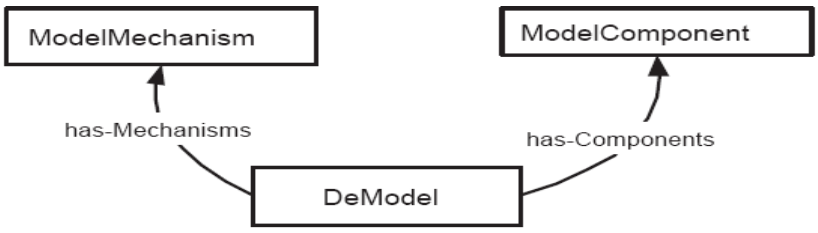

Figure 1: DeMO DeModel, ModelComponent and ModelMechanism classes

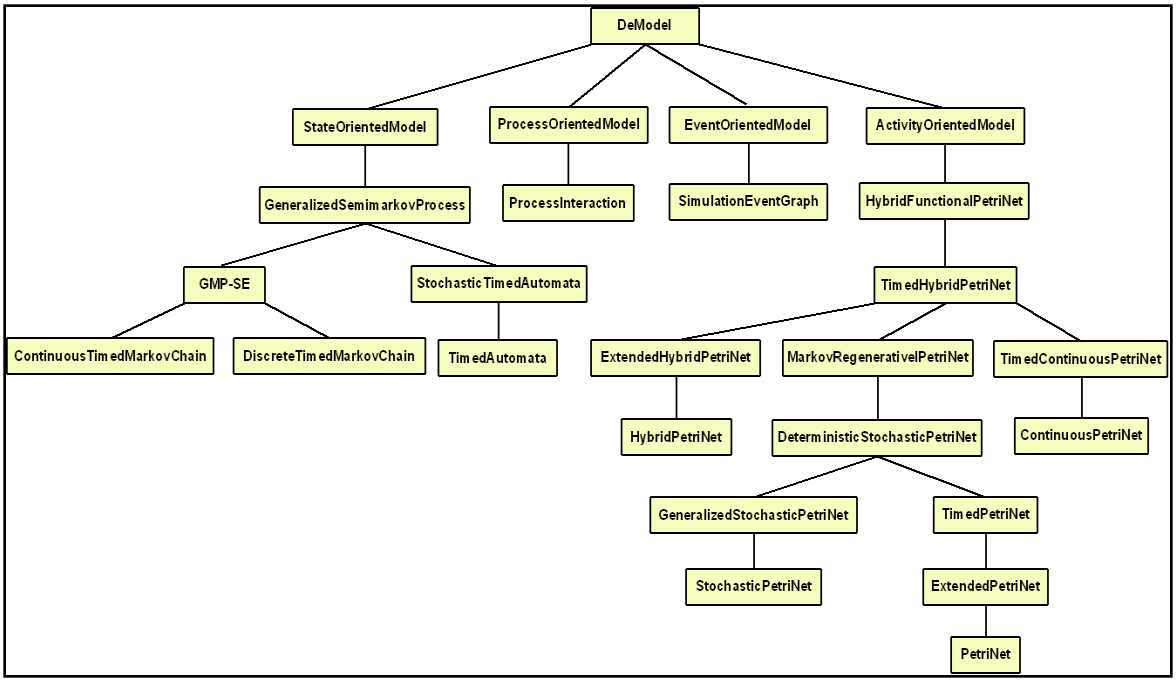

Figure 2: DeMO DeModel class hierarchy

2.5 JSIM

JSIM is a tool built at the Large Scale Distributed Information Systems Lab, and it is a Java based simulation and animation environment [Nair and Miller, 1996]. Initially, JSIM provided a Java-based simulation language as well as a graphical user interface for the user to create a simulation model. Our work built code generators to produce simulation code by reading conceptual models instantiated in the DeMO ontology. Figure 3 shows the simulation and animation of a process interaction model.

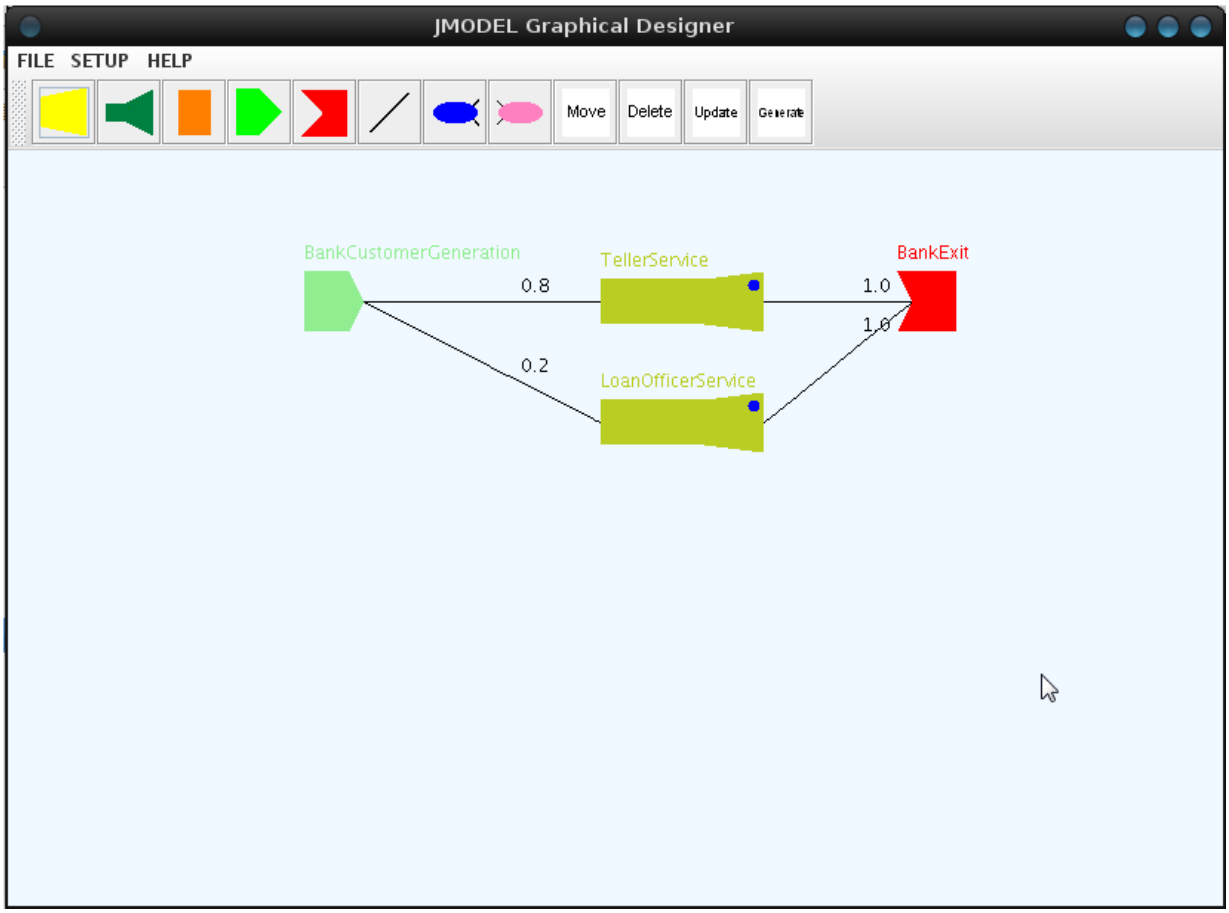

Figure 3: Simulation of a Process Interaction Model in JSIM

2.6 The Domain Ontology in Study: ReactO

For the purpose of this thesis, the domain ontology that we are using which interoperates with DeMO is the Reaction Ontology (ReactO). ReactO is an ontology built and maintained by the Complex Carbohydrate Research Center (CCRC) at the University of Georgia. ReactO imports three other ontologies, namely GlycO, EnzyO and RO. The Glycomics Ontology, GlycO, focuses on modeling the structure and functions of glycans. The Enzyme ontology, EnzyO, contains knowledge about the enzymes that are involved in the biosynthesis and modifications of the glycans from the GlycO ontology. The Relation Ontology, RO, contains logical relations shared across different Open Biomedical Ontologies(OBO) [Smith et al., 2005]. ReactO is an

ontology that encompasses this knowledge along with the knowledge of reactions and biochemical pathways [Appendix A] based on the glycans and enzymes. The conceptual framework of the pathway as described in ReactO is shown in figure 4. The pathway class may have one or more reactions and each reaction may consume a source molecule which is catalyzed by an enzyme to generate an altered molecule. In our project, we will be talking about a specific biochemical pathway that is a part of the overall the N-Glycan Biosynthesis Pathway.

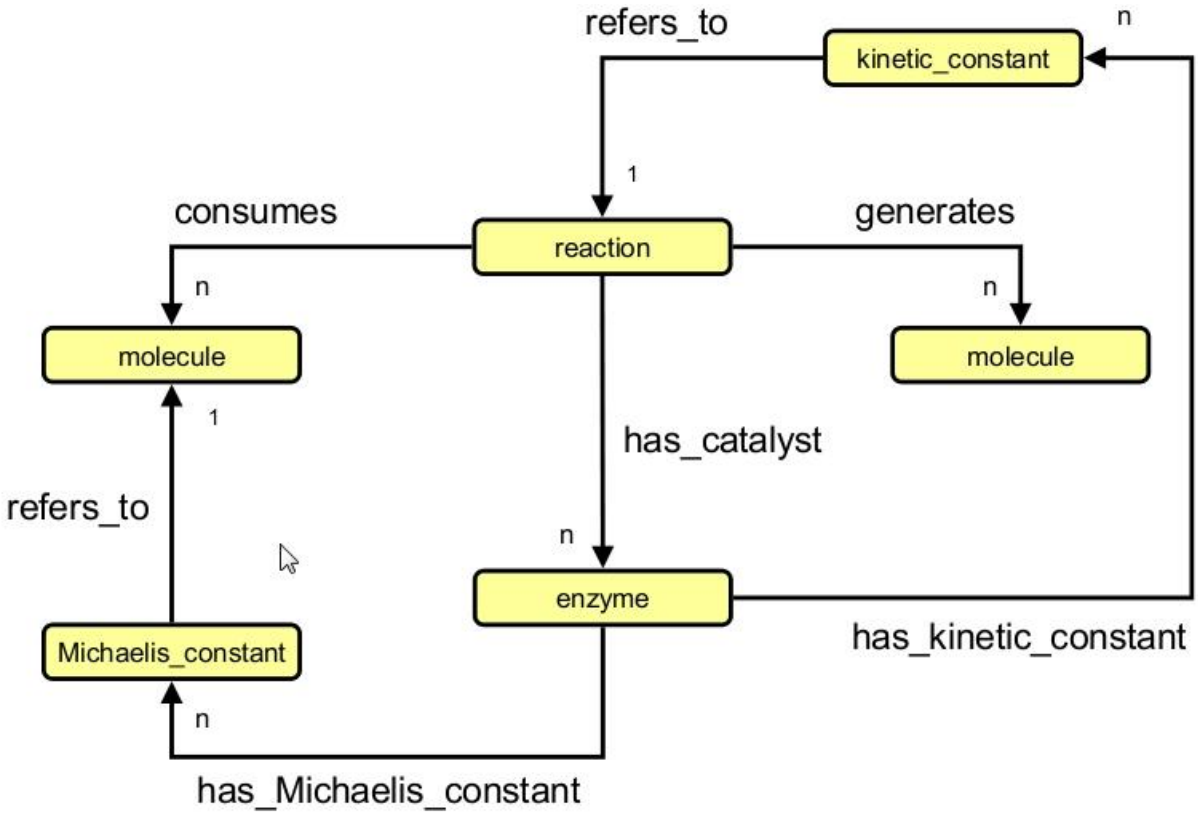

Figure 4: The structure of the 'reaction' class: the main component of the 'pathway' class [York et al., 2009]

2.7 Ontology Mediation

Ontology mediation is an area of research which aims at finding correspondences between a set of ontologies in order to facilitate reuse and sharing of knowledge and also the interoperation

between two or more agents. It is often used to facilitate interoperation between systems that root their semantics within ontologies. Ontologies are being developed all over the world in homes, labs, schools and industries. Since there is no central authority to standardize the development of these ontologies, there exists several ontologies describing the same domain with the same or similar entities (classes, relationships and instances). Because of this, systems which employ these ontologies, either as their foundation or as an integral part, are not able to communicate and collaborate with each other to achieve a common or consolidated goal. Ontology mediation aims at minimizing if not eliminating these roadblocks and differences in order to allow these systems to interoperate with each other.

Ontology mediation can be subdivided into three areas: ontology alignment, ontology mapping and ontology merging [Bruijn et al., 2006]. Ontology alignment is the process of comparing two ontologies and attempting to find similarities (semantic and syntactic) between every pair of corresponding entities (classes, relationships and instances), thus bringing the two ontologies 'in line' with one another. Ontology mapping is the process of representing these similarities or correspondences between the entities of the two ontologies in and storing them in a structured manner. Ontology merging is the process of merging the two ontologies based on their similarities, thereby combining the relevant knowledge and eliminating redundancy.

In recent times, ontology mediation has been used in the Web Service Modeling Execution Environment (WSMX) [Mocan, 2004], to increase the efficiency of search and retrieval systems by using multi-agent systems [Rahimi et al., 2008], in multi-representation ontology-based information systems to facilitate interoperation [Benslimane, 2005], to bridge the gap between formal and informal knowledge [Marc and Ralf, 2005], for collaboration between heterogeneous venue services [Tan et al., 2003] and a proposal has been made to utilize ontology mediation

techniques in the Service Oriented Architecture (SOA) to improve Web service discovery and Web service composition [Polleres, 2006].

2.8 Semantic Web Rule Language (SWRL)

The Semantic Web Rule Language (SWRL) is based on a combination of the OWL-DL with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language [Horrocks, 2004]. SWRL provides the user with the capability of expressing rules which may be used to infer new knowledge or establish certain facts. Every rule is an implication between an antecedent (body) and consequent (head), i.e., "whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold". Below is an example related to the ReactO ontology:

(reaction(?r) ^ consumes (?r,?m) ^ molecule (?m))

^ (reaction(?r) ^ hasCatalyst(?r,?e) ^ enzyme(?e) ^ hasMichaelisConstant(?e, ?mc))

→ MichaelisConstant(?mc) ^ refersTo(?mc, ?m)

In the above example, reaction, molecule, enzyme and MichaelisConstant are classes while consumes, hasCatalyst, hasMichaelisConstant and refersTo are properties within the ReactO ontology. The variables that substitute the instances of the classes are preceded by question marks (if the value of a particular variable is known then it can be used directly). The above rule simply establishes the fact that if a particular reaction consumes a molecule and is catalyzed by an enzyme which has a particular Michaelis Constant, then this Michaelis constant has to refer to the same molecule consumed by the reaction (this rule is used to ascertain the fact that a

particular Michaelis Constant exists between a molecule-enzyme pair participating in a biochemical reaction). Note, in more complex situations we would need a ternary predicate, such as rate_of (?m,?e,?mc), so that given a substrate molecule (?m) and an enzyme (?e), the Michaelis Constant ?mc is uniquely determined.

Though SWRL is primarily a rule language, we can use it as a querying language as well and retrieve information by executing SWRL rules as queries. In this thesis, we use the rule language characteristics of SWRL to expedite the mapping process and the query language characteristics to retrieve the instances from the domain ontology in order to transform them into DeMO instances.

CHAPTER 3

PROBLEM STATEMENT AND SOLUTION STRATEGIES

Ontologies have the capability to capture and maintain domain knowledge in a structured fashion and this makes it easy to share knowledge across systems in the way in which it is intended, without loss of semantics or meaning. Ontologies are being developed in almost every domain these days and it is not surprising that the simulation experts began to analyze the use of ontologies in their discipline [Miller et al., 2004]. Since then, the development and use of ontologies in the field of modeling and simulation have surfaced and have become more popular as indicated by the following:

- Discrete Even Modeling Ontology (DeMO)
- The evaluation of the Command and Control Information Exchange Data Model (C2IEDM) [Tolk, 2005], as an interoperability-enabling ontology
- The Process Interaction Modeling Ontology for Discrete Event Simulations (PIMODES) [Lacy, 2006] and the proposed Component Simulation and Modeling Ontology (COSMO) [Teo and Szabo, 2006]
- The development of an ontology to represent ports for automated model composition – "ports define the locations of interaction at the boundaries of components or sub-systems" [Liang and Paredis, 2003]

With the use of ontologies, many doors open in the world of modeling and simulation with respect to factors such as interoperation, automation and knowledge representation. For systems

to communicate with each other, the way they represent knowledge must be standardized. Ontologies facilitate this exact requirement and they do so in a formal and structured fashion. Also, in many cases, inferences can be made from ontologies which further improve the interoperability capabilities between systems.

As discussed earlier, there is a need for the knowledge encapsulated in domain ontologies, in the form of conceptual models, to be simulated and this would entail the domain expert to have in-depth knowledge about simulation and simulation software packages to interoperate with every other domain ontology. This poses a complex situation to both the domain expert and the simulation software developer. We propose to solve this by having the domain ontologies interoperate with a common modeling ontology and the modeling ontology with the simulation software. This approach reduces the complexity on both sides of the modeling ontology.

The DeMO ontology has been developed at the LSDIS lab and this ontology contains templates to capture knowledge about discrete event models such as activity oriented models, event oriented models, state oriented models and process oriented models. Instances from the ModelComponent and ModelMechanism classes may then be used to create instances of the DeModel class. Using DeMO as our common modeling ontology, we can have the domain ontologies interoperate with it and transform the knowledge to it. Then, the simulation software can interoperate with DeMO and read the models from it and perform the simulation.

The interoperation between the domain ontology and the DeMO ontology will be facilitated by ontology mediation techniques. The alignment and mapping components of ontology mediation

will be used to achieve this. A portion of the structure of DeMO and ReactO and their respective entities can be seen in section 2. Our initial thoughts were to automate or semi-automate the alignment and mapping of DeMO and ReactO. In order to see if the alignment and mapping process can be automated, we tried a few existing ontology alignment/mapping tools. First we tried PROMPT [Noy et al., 2003] to see if its 'merge' component provided any intuitive suggestions while trying to merge DeMO and ReactO. The intention was not to merge the two, but if there was any kind of similarity, for example that between the Pathway class in ReactO and HybridFunctionalPetriNet class in DeMO, then PROMPT would have suggested merging these two classes together and we could have used the suggestions to perform an automated mapping; however since, PROMPT primarily identifies linguistic similarities, it could not provide any accurate suggestions. Then we tried FOAM [Ehrig et al., 2005] which is a tool that semi-automatically aligns two or more ontologies. It is based on similarity heuristics and concentrates on labels and structures. However FOAM, was of no avail either. Finally we looked at the results of the Ontology Alignment Evaluation Initiative, 2008 [Caracciolo et al. 2008] and tried one of its top contenders, RiMOM. RiMOM is a tool for ontology alignment that combines different strategies such as Edit-distance based strategy, WordNet based strategy, Vector-similarity based strategy and few other strategies [Zhang et al., 2008] to find the optimal alignment results. Though RiMOM is far more advanced than PROMPT and FOAM, it failed to give any satisfactory mapping results. We did foresee this happening; however we decided to try it, to see if there was any scope for automatic or at least semi-automatic alignment/mapping between the two ontologies. The reason for this inaccuracy and void of automatic alignment and mapping is because there was no taxonomical similarities between the two ontologies, no lexical similarities between the entity labels (ex: between reaction and ContinuousTransition or between

molecule and ContinuousPlace) and no other significant semantic similarities. It might be unrealistic to expect such similarities from any domain ontology when compared to DeMO; however if the domain ontologist annotates his ontology with correspondence to DeMO, then newer ontology alignment and mapping tools which consider the annotations may be able to detect a few correspondences. However, due to the stated reasons, the mapping process will not rely on these automated alignment tools in order to establish the correspondences between the domain ontologies and the DeMO ontology. The mapping process will be fairly simple and will expect the domain expert to have only limited knowledge about DeMO. There are ways in which the mapping process will be user friendly and also assist the user in the process and this is discussed further in chapter 4. Further, use of common base ontologies such as RO or Basic Formal Ontology (BFO), could allow fine anchor points to be established in the ontology that could bootstrap further alignment.

Once the correspondences between the domain ontology and the DeMO ontology have been established, the conceptual model from the domain ontology must be translated and transformed into the DeMO ontology in the form of DeMO instances. This will be an automated process. SWRL queries will be used in this process to retrieve instances from the domain ontology and SWRL rules will be used to formulate the structure of the model in DeMO and perform the transformation. One of the other popular query languages for ontologies is SPARQL (SPARQL Protocol and RDF Query Language) [Prud'hommeaux and Seaborne, 2004]. However, some of the advantages of SWRL over SPARQL are as follows:

1. SWRL can be used as a rule language and a query language unlike SPARQL, which is solely a query language.
2. SWRL rules and queries can be stored and maintained within an ontology itself, thus allowing their reuse.

A code generator should be developed which reads simulation models from DeMO and produces executable code based on these models. The code generator should also handle scenario management in the context of retrieving additional simulation input parameters from spreadsheets, databases or ontologies and it should have a layout manager which should aid in generating the graphical animation component of the executable simulation code.

To summarize, DeMOForge should have three modules: a mapping module that establishes correspondences between the domain ontology and the DeMO ontology, a transformation module that transforms the conceptual model from ReactO to DeMO and a code generator module that reads the model from DeMO and generate executable code to perform the simulation.

CHAPTER 4

DEMOFORGE - IMPLEMENTATION

DeMOForge is the tool that we developed to solve the problem that is stated in the previous section. DeMOForge allows users to transform the conceptual models stored in their respective domain ontologies to DES models and then produces executable code conforming to specific simulation software packages, which can be used to simulate these models. DeMOForge is divided into three phases: The mapping phase, the transformation phase and the code generation phase. The mapping phase helps the user in aligning the domain ontology with DeMO and creating the correspondences between them, the transformation phase converts the conceptual model from the domain ontology to one in the DeMO ontology and the code generation phase generates the executable code which simulates the model in a specific simulation package. The use of DeMOForge is illustrated by converting a portion of a biochemical pathway from the ReactO ontology to a Hybrid Functional Petri Net model in the DeMO ontology, with each of the three phases explained in detail. The architecture of DeMOForge is shown in figure 5.

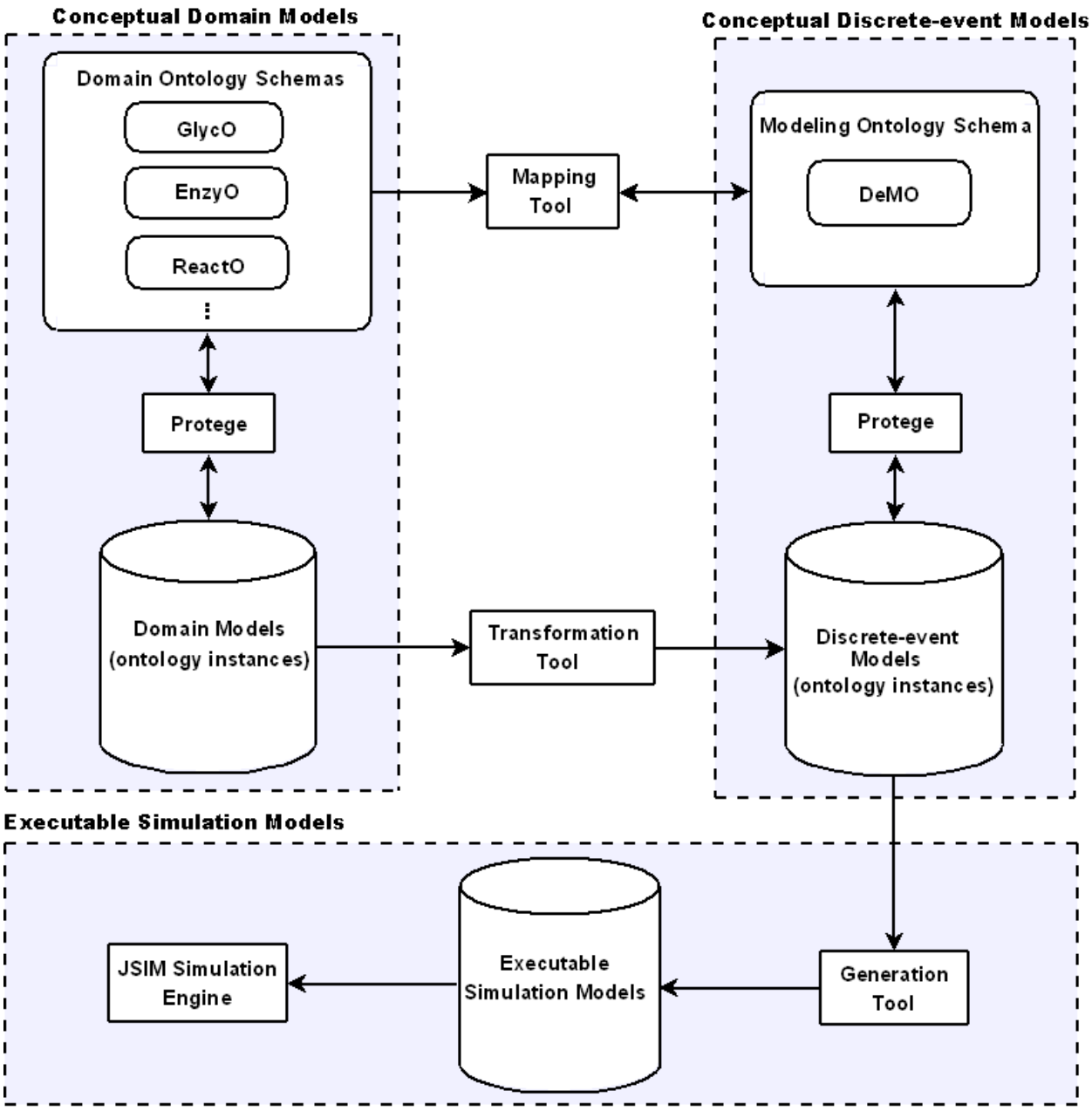

Figure 5: Architecture of DeMOForge [Silver et al., 2009]

4.1 The Mapping Phase

In this phase, the domain expert loads his ontology in the opening window and specifies the name of the conceptual model, within the domain ontology, that he wishes to simulate. The name

of the conceptual model is required because the ontology may contain many models, but the user may want to simulate only one of these, for example the ReactO ontology may have many biochemical pathways instantiated in it, but the user may wish to simulate only one, in this case, we consider a portion of the N-Glycan Biosynthesis Pathway. DeMOForge then loads the classes of the domain ontology and the DeMO ontology as hierarchical trees, beside each other. As discussed earlier, automated ontology alignment is not possible in this context as the entities between the two ontologies do not hold any correspondences with each other syntactically or semantically. Therefore, the mapping process is computer-aided: however the domain expert is assisted in a host of ways to make this process as intuitive as possible. There are search boxes for each of the ontologies which the domain expert can use to find a particular class. Also, the choices are narrowed down when the domain expert clicks on a particular model in the DeMO ontology and it basically trims the class hierarchy and shows only those classes that are relevant to the particular model. A few more intuitive suggestions are shown to the user in the latter part of the mapping phase which will be discussed shortly.

First the domain expert has to map the relevant roots of the conceptual models from both the ontologies and in this case he has to map the class "pathway" from the ReactO ontology to the class "HybridFunctionalPetriNet". As soon as he does this, the tool trims the ontologies and displays only those classes that are relevant to the pathway class in ReactO and HybridFunctionalPetriNet in DeMO. The user now sees a very few classes from which he has to continue setting up the mappings. He chooses to go the next level in the ReactO pathway which is the reaction class and he maps this to a ContinuousTransition and finally maps the Enzyme class to a ContinuousPlace and the Molecule class to the ContinuousPlace, as well. So, as can be seen, the mappings are not one to one. A single class in the domain ontology can be mapped to

multiple classes in the DeMO ontology and vice versa. If the user wishes to, he can save the mappings at any point by clicking the Save2Onto button. This saves the mapping information within DeMO itself. This serves two purposes. One is that these mappings can be reused and the next time the domain expert uses this tool, he does not have to go through the whole mapping process again, he has to just click Load4mOnto button which will load the mappings that were made with the current domain ontology and DeMO. The other use is discussed in chapter 5.

Once the user is done creating the mappings, he clicks the 'Done Mapping' button. As soon as he does this, DeMOForge computes what information is required from the user in order to perform the transformation and the user is alerted of these requirements as shown in figure 6.

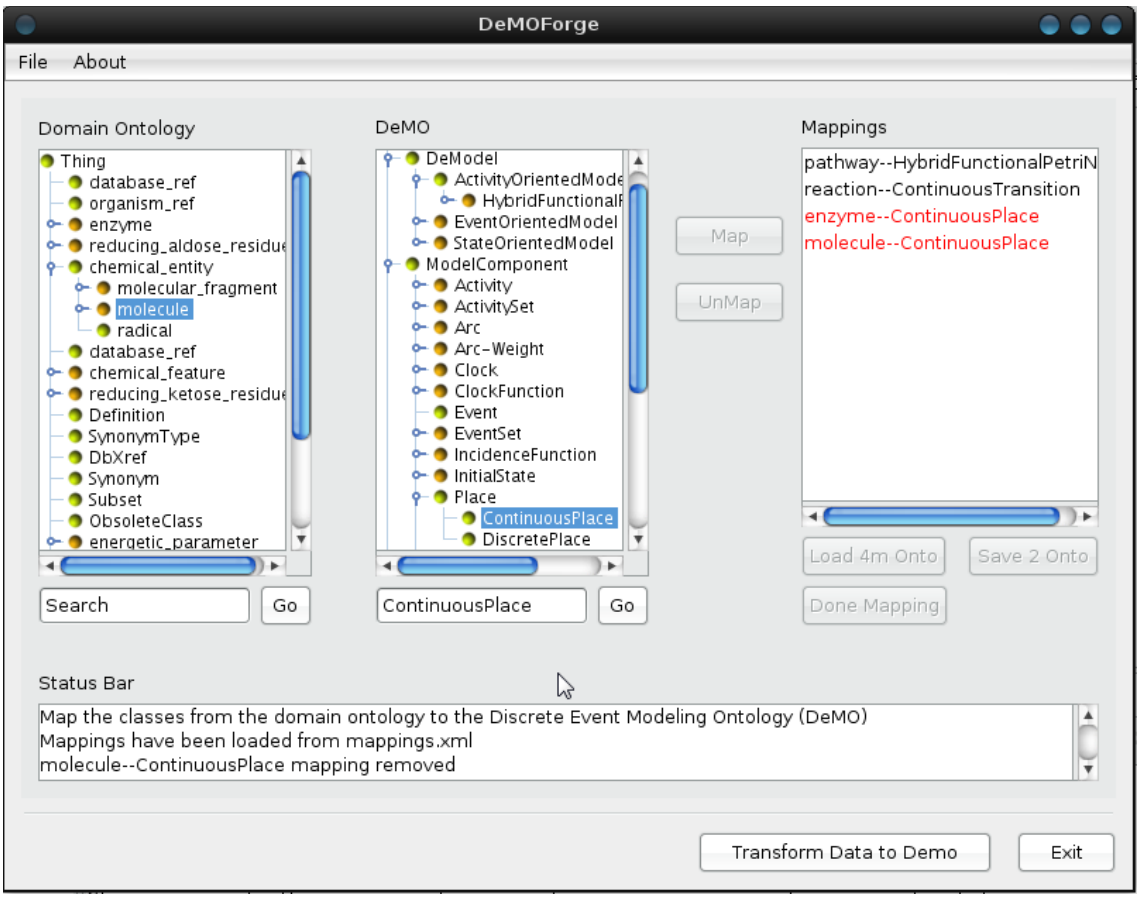

Figure 6: The user being alerted (mappings in red) for extra information

For example, in the case of a Petri Net, information about the arcs is required which connects the instances of the Places to Transitions or vice versa. Without this information, a topology of the conceptual model cannot be built and thus the sequence of events will not be known. Also, there may be more than one way of retrieving the instances for a particular class. In such a case the user is alerted and the user will be shown all computed ways or paths of retrieving instances for a particular class and the user may choose a subset of these paths, for example, as shown in figure 7, there are two ways of retrieving the instances of the Molecule class: Reaction-consumes-Molecule or Reaction-generates-Molecule.

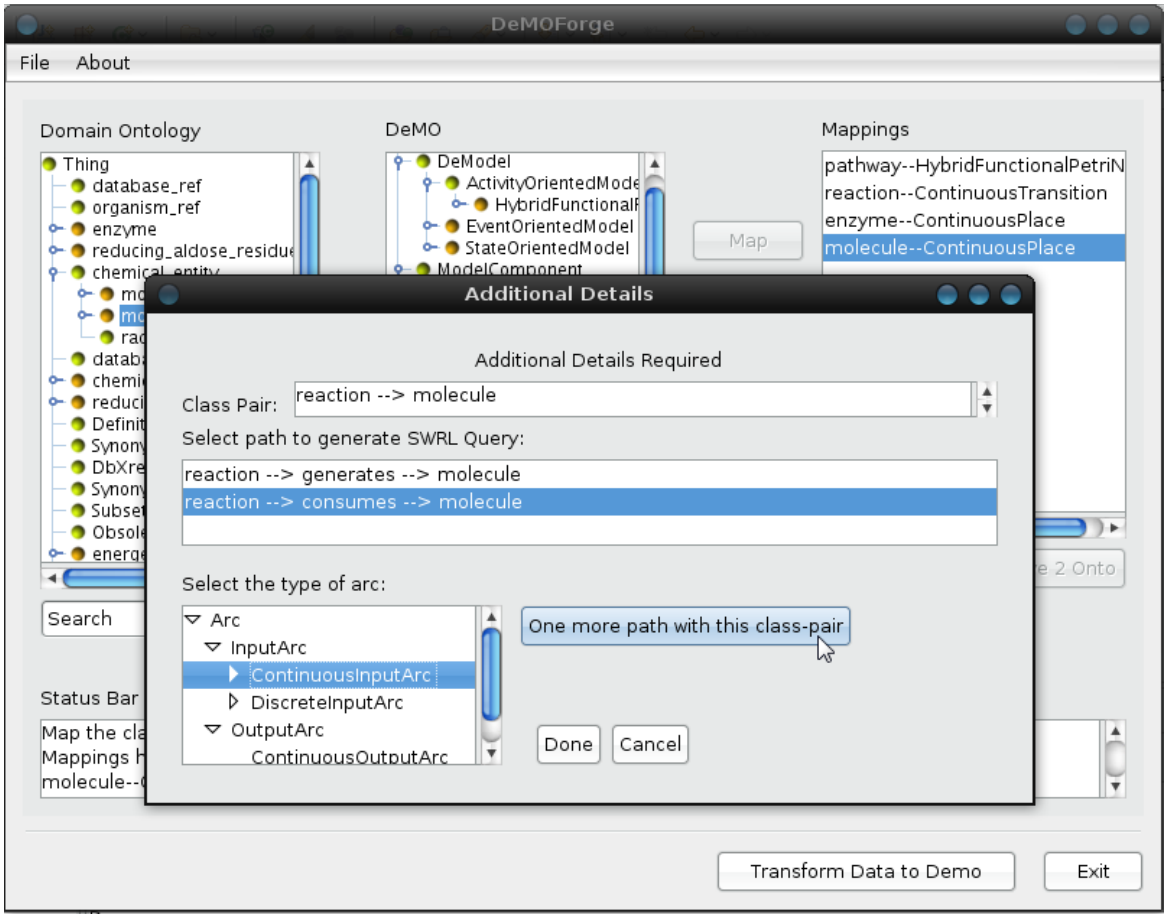

Figure 7: Multiple ways to retrieve instances of a class

Both these paths will produce a different set of instances. Since both these are relevant in this context, the user selects both of them, and also specifies how exactly the molecule class is linked to the reaction class from the perspective of the DeMO Arcs. The Arc types shown in figure 8 are the subclasses of the Arc class, however in ReactO, the arcs that connect the reactions to molecules (consumes and generates) or reactions to enzymes (has_catalyst) are properties; hence, unlike the class to class mapping this would be a property to class mapping. Table 1 shows the class to class mappings and 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

4.2 Transformation Phase

In the transformation phase, the instances of the conceptual model in ReactO are transformed to DeMO instances to conceptualize a DES model of the pathway. The transformation module uses the established mappings and the computed paths to every mapped class to retrieve those specific instances. It uses SWRL to retrieve the instances. First a simple transfer of the instances from ReactO to DeMO is done, where the instances of the classes mapped from ReactO are created as instances of the respective classes mapped in DeMO. Then DeMOForge fills out any missing information that does not require the intervention of the user and then finally it uses the extra information supplied by the user, for example the arc information, to link the transferred instances in DeMO to one another. The process is explained in finer detail below:

1. SWRL queries will automatically be generated and used to retrieve the ReactO instances that represent the pathway model. As every SWRL query is generated, if the query contains classes for which instances have already been retrieved through a path which is the subset of the path used to retrieve instances for the current class, then these instances are used in the generated query. For example for the path, pathway→reaction→has_catalyst→enzyme, since instances for reaction have already been retrieved using the path, pathway→reaction, they will be used to retrieve the instances of the enzyme class. We are using a trial version of the Jess API (provided by Sandia National Laboratories) to execute SWRL queries (in the process of applying for an academic license).
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 Hybrid Functional Petri Net formalism is captured, and SWRL rules are generated according to the formalism.
4. The SWRL rules created in step 3 along with the information given by the user (eg: Arc type information) are used create relationships between the DeMO instances so that they form a conceptual HybridFunctionalPetriNet model of the pathway.

The illustration in figure 8 below explains this further and shows the conceptual model before transformation in the ReactO ontology and after transformation in the DeMO ontology. In figure 8, only a tiny portion of the N_Glycan_Biosynthesis pathway with just one reaction is shown due to space constraints.

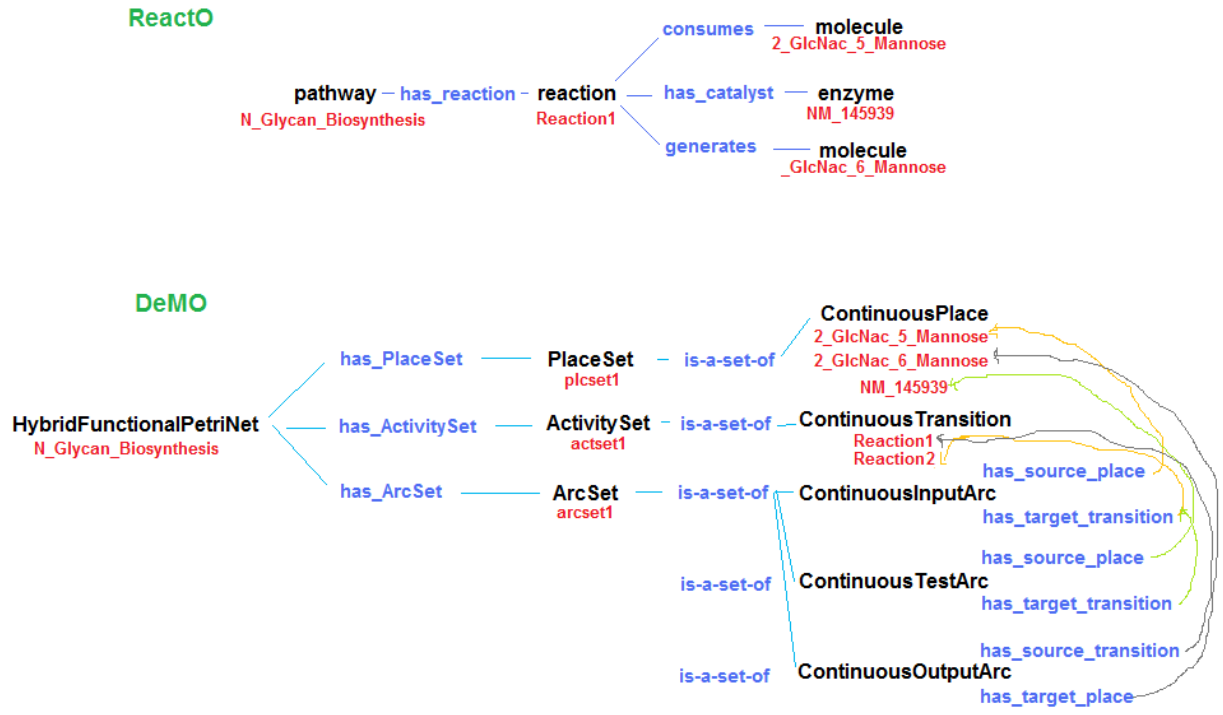

Figure 8: Before and after transformation of Reaction1 (some entities have been trimmed due to space constraints)

4.3 Code Generation

Once the transformation has been completed, the code generator can be used at any time to generate a JSIM executable simulation code of any DES model within DeMO. The code generator is isolated from the rest of the DeMOForge tool, programmatically, in terms of sequence of execution to make the application more flexible. In this case the user is asked for the name of the model again, which he wishes to simulate. The code generator looks up the model in DeMO and retrieves it, then stores it in a data structure which can handle the graph-based representation of the DES model and constructs a topology of the model internally. Also, the code generator notes the kind of DES model that needs to be simulated and detects the execution parameters or data values that are required in order to run this simulation, such as the token/fluid levels of the places. The user can feed these parameters from a spread sheet, a database or another ontology. The scenario management component of the code generator differentiates between the three and creates a hash-map of the variable parameters which will then be used during code generation (note, one generated model can be run under several scenarios). The code generator also has a layout manager which calculates the co-ordinates of each of the components, depending on the kind of DES model that will be simulated (for example the components of a process interaction model are different from those of a Hybrid Functional Petri Net model), and maps these co-ordinates on the canvas of the simulation engine.

Once the code has been generated, the user may at any point simply run the generated executable to simulate the model and view the animation in JSIM. Figure 9 shows the simulation of the portion of the biochemical pathway, modeled as a Hybrid Functional Petri Net. The red ellipses are the places (ContinuousPlace), and the blue rectangles are the transitions (ContinuousTransition). The fluid levels of the places can also be seen inside of them. Figure 10

shows the state of equilibrium of each of the substrates after a certain point of time (800ms in this case).

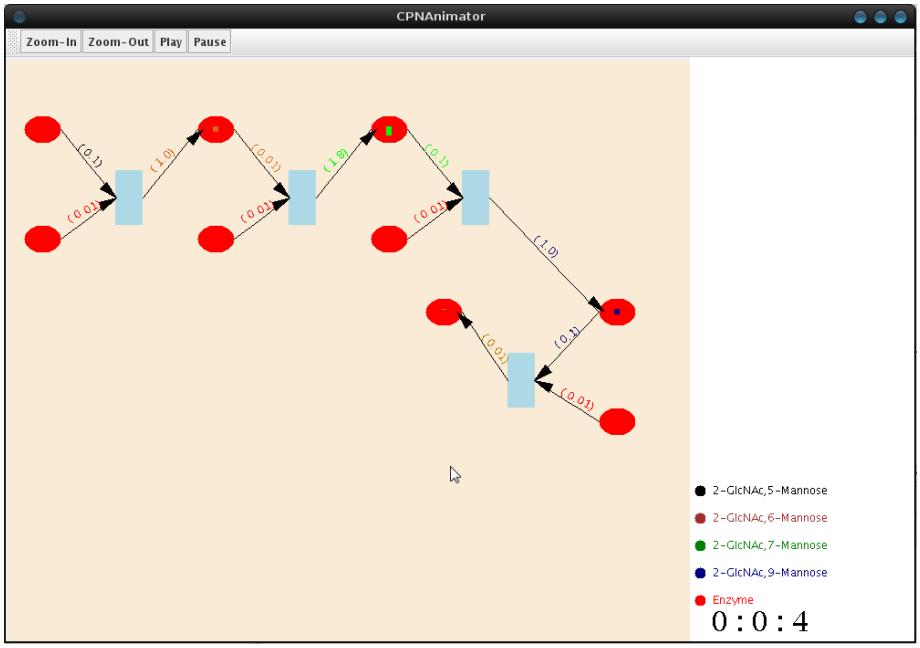

Figure 9: Simulation of the pathway model as a Hybrid Functional Petri Net in JSIM

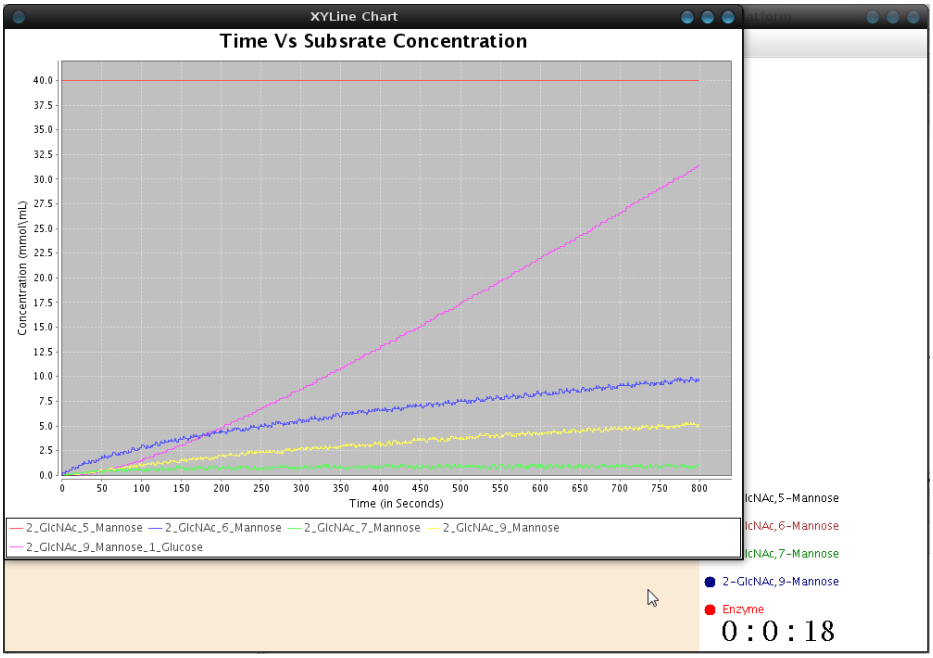

Figure 10: Simulation statistics in JSIM

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

As stated in section 3, we have developed a tool that enables the interoperation between a domain ontology and DeMO and transforms the conceptual model from the former to the latter. It then generates executable code which can be used to run the simulation of the conceptual model in JSIM. We used a portion of the N-Glycan Pathway instantiated in ReactO and the results obtained have been satisfactory and the application was able to generate the final product (DeMO with transformed model and executable simulation code). DeMOForge applied the methodologies of ontology mediation and facilitated the interoperability between ReactO (domain ontology) and DeMO (modeling ontology), particularly in the transformation of a conceptual biochemical model to a conceptual DES model. In fact, the techniques used in this paper such as mapping, knowledge retrieval and storage (via rule generation and rule execution) and transformation of knowledge can be used to facilitate interoperation between any two ontologies. We also discuss the use of rule systems to aid in the transformation process. The Semantic Web Rule Language (SWRL) was used to accomplish this. We not only use the rule system component of SWRL, but also use the querying system extensively. The use of SWRL in the process of ontology mediation in the context of ontology driven simulation has never been done before.

In a way, DeMO is learning about domain specific simulation models using the feature where it stores the mapping details within the ontology itself. Using this stored information about mappings and associated details between domain specific ontologies and DeMO, we can prompt the user with mapping suggestions, when a domain specific model with similar features as that of an already 'learned' model is being transformed into a DES model. As already discussed, it is hard to automatically align and map domain ontologies with DeMO. However, if ReactO has already been mapped to DeMO, then when an ontology with components similar to ReactO such as the Biochemistry Ontology [Dumontier, 2007] is to interoperate with DeMO, our tool can be programmed to align the previous mappings (documented knowledge) with the Biochemistry Ontology based on certain similarity measures thereby assisting the user in mapping the Biochemistry Ontology with DeMO.

Also, suggestion features (e.g., when suggesting the arc types) can be improved and can be more specific in its suggestions, by applying semantic reasoning on like components between the two ontologies. For example, the property 'consumes' in ReactO is a synonym of 'input' which would imply that this relationship would correspond to a type of an InputArc in DeMO.

Currently, we are laying the foundation to making the mapping process visually intuitive by displaying the model structures from the ontologies side by side as graphs with classes and properties rather than as class hierarchy trees as shown in figure 11. Using such a visually enhanced mapping tool, the user can simply click on a class in the domain ontology and then click on the corresponding class in DeMO, let the tool trim the ontologies or trim the ontologies himself by specifying parameters such as level depth and continue with the mapping.

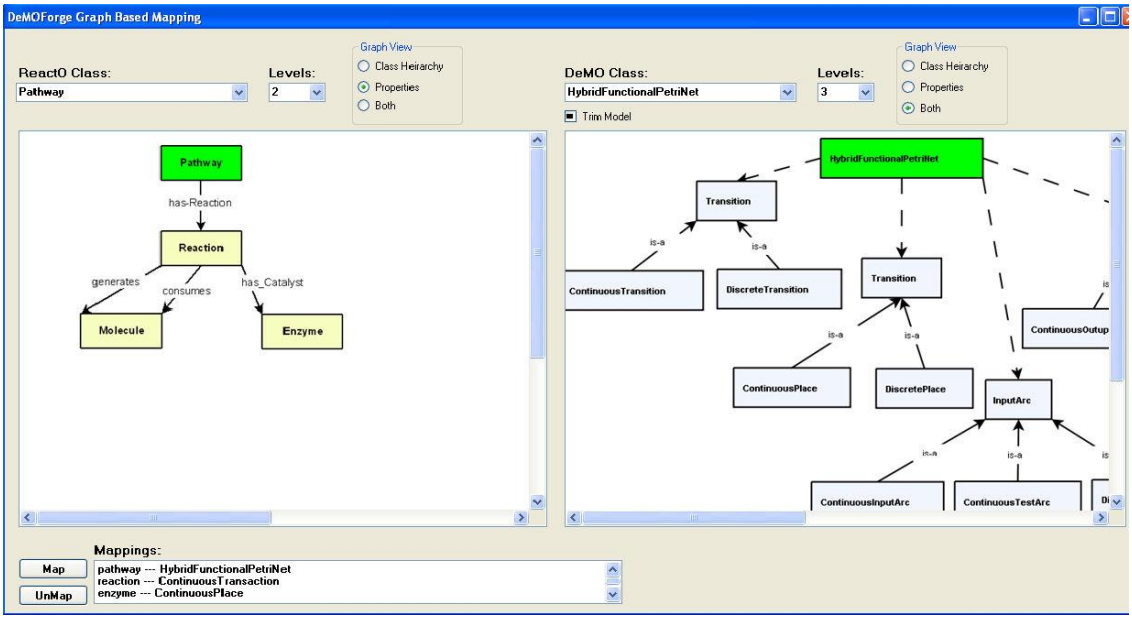

Figure 11: Visual mapping process

REFERENCES

Arsham, Hossein. Systems Simulation: The Shortest Route to Applications. In http://home.ubalt.edu/ntsbarsh/simulation/sim.htm. 1995.

Bapat, V. and Sturrock, David T. The arena product family: enterprise modeling solutions. In Proceedings of the 35-th conference on Winter simulation: driving innovation. Pages 210-217. 2003.

Bechhofer, Sean, Volz, R. and Lord, Phillip. Cooking the Semantic Web with the OWL API. Published by Springer Berlin. 2003.

Benslimane, S.M., Bensaber, D.A. and Malki, M. Multi-representation ontology-based information systems mediation. In Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference

Bloechle, W.K. and Schunk, D. Micro Saint Sharp simulation software.In Simulation Conference, 2003. Proceedings of the 2003 Winter. Pages 182-187 Vol.1. 2003.

Brickley, Dan and Guha R.V. Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation. 1999.

Bruijn, Jos de, Ehrig, Marc and Feir, Cristina. Ontology mediation, merging and aligning. In Semantic Web Technologies, editors John Davies and Rudi Studer. John Wiley & Sons, Ltd., 2006.

Dumontier, Michel. http://dumontierlab.com/?page=ontologies. Dumontier Lab, Carleton University. 2007.

Gruber, Thomas R. A translation approach to portable ontology specifications. In: Knowledge Acquisition. 5: 199-199. 1993.

Horrocks, I., Harmelen, Frank van, Peter Patel-Schneider et al. DAML+OIL. In IEEE Data Eng Bull 25(1):4-9. 2001.

Horrocks, Ian, Patel-Schneider, Peter F., Boley, H., Tabet, S., Grosof, B. and Dean, Mike. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. 2004.

Knublauch, H., Fergerson, R.W., Noy, N.F., and Musen, M.A. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In Lecture Notes in Computer Science. Published by Springer. Pages 229-243. 2004.

Lacy, L. W. 2006. Interchanging Discrete-Event Simulation Process-Interaction Models using the Web Ontology Language – OWL. Unpublished PhD Dissertation, Department of Industrial Engineering and Management Sytems, University of Central Florida, Orlando, Florida.

Liang Vei-Chung and Paredis Christiaan J. J. Foundations of multi-paradigm modeling and simulation: a port ontology for automated model composition. In proceedings of the 35th conference on Winter Simulation: driving innovation. Pages 613-622. 2003.

Marc, S. and Ralf, K. Mediating ontologies for communities of practice. In Lecture Notes in Computer Science. Pages 330-342. Published by Springer. 2005.

McGuinness, Deborah L. and Harmelen, Frank van. Feature Synopsis for OWL Lite and OWL. W3C Working Draft. 2002.

Miller, Eric and Schloss, Bob. Resource Description Framework (RDF) Model and Syntax. W3C Recommendation. 1997.

Miller, John A., Baramidze, G., Fishwick, Paul A. and Sheth, Amit P. Investigating Ontologies for Simulation Modeling. Proceedings of the 37-th Annual Simulation Symposium (ANSS'04), Arlington, Virginia (April 2004). Pages 55-71. 2004.

Miller, John. A, Seila, Andrew F. and Xiang, Andrew F. The JSIM Web-Based Simulation Environment. Future Generation Computer Systems (FGCS), Special Issue on Web-Based Modeling and Simulation, Vol. 17, No. 2 (October 2000). Pages 119-133. Elsevier North-Holland. 2000.

Mocan, Adrien. Ontology Mediation in WSMX. in ICWS 2005. Proceedings. 2005 IEEE International Conference on Web Services. 2005.

Moremen, Kelly. http://www.ccrc.uga.edu/~moremen/glycomics/. 2008.

Nair, Rajesh, Miller, John A., Zhang, Zhiwei and Zhao Hongwei. JSIM: A Java-Based Simulation and Animation Environment. Proceedings of the 30-th Annual Simulation Symposium (ANSS '97), Atlanta, Georgia (April 1997) pp. 31-42.

Nimmagadda, K. Ontology Driven Simulation of Biochemical Pathways Using Hybrid Petri Nets. Master's Thesis, University of Georgia. 2008.

Noy, N.F., Sintek, M., Decker, S., Crubezy M., Fergerson, R.W. and Musen, M.A. Creating semantic web contents with protege-2000. In IEEE Intelligent Systems. Pages 60-71. 2001.

Ong, Chee-Mun. Dynamic simulation of electric machinery : using matlab/simulink. 1998.

Park, Steve and Leemis, Larry. Discrete Event Simulation: A First Course. Prentice Hall. 2001.

Petri C.A. Fundamentals of a Theory of Asynchronous Information Flow. Proceedings of IFIP Congress 62. Pages 223-252. 1962.

Polleres, Axel. The Role of Ontology Mediation in DIP. 2006.

Prud'hommeaux, Eric and Seaborne, Andy. SPARQL Query Language for RDF. W3C Working Draft. 2004.

Rahimi, Shahram, Rana, Pravab J., Ahmad, Raheel and Gupta, Bidyut. Ontological Mediation for Multi-Agent Systems. In the publication, International Journal of Electronic Government Research, Volume 4, Issue 1 edited by Mehdi Khosrow-Pour, IGI Global. 2008.

Roher, M.W. Maximizing simulation ROI with AutoMod. In Simulation Conference, 2003. Proceedings of the 2003 Winter. Pages 201-209 Vol.1. 2003.

Silver, G., Bellipady, K., Miller, J. A., Kochut, K. and York, W. Supporting Interoperability using the discrete-event modeling ontology (DeMO). Proceedings of the 2009 Winter Simulation Conference. 2009.

Singh, Muninder P. and Huhns, Michael. Service-Oriented computing - Semantics, Processes, Agents. Published by Wiley. 2005.

Smith, B., Ceusters, W., Klagges, B., Kahler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector A.L. and Rosse C. Relations in biomedical ontologies. 2005.

Smith, B., Ceusters, W., Klagges, B., Köhler, J., Kumar, A., Lomax J., Mungall, C., Neuhaus, F., Rector, A. L. and Rosse, C. Relations in biomedical ontologies. In Genome BioMed Central. 2005.

Tan, J. J., Poslad, S., Titkov, L. Ontology mediation in heterogeneous venue services. In Proceedings of the second international joint conference on Autonomous agents and multiagent systems. 2003.

Teo, Y. and C. Szabo. 2008. CODES: An Integrated Approach to Composable Modeling and Simulation. In Proceedings of the 41st Annual Simulation Symposium (anss-41 2008), 103-110. April 2008.

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

Verzuilli, J. Using the Jena API to Process RDF. O' Reilly. In XML.com. 2001

York, William S., Kochut, Krys J. and Miller, John A. "Integration of Glycomics Knowledge and Data", in Hand Book of Glycomics, R.D. Cummings and J.M. Pierce. August 2009.

APPENDIX A

BIOCHEMICAL PATHWAYS AND PETRI NETS

Biochemical Pathways

Biochemical pathways can be defined as a network of reactions involving complex interactions between bio-molecules within a cell. Biochemical pathways include metabolic pathways, transduction pathways and gene regulatory pathways.

A metabolic pathway is a network of reactions occurring in a cell. In every pathway, a molecule or a substrate is typically catalyzed by enzymes to produce a modified molecular product. An example of an enzyme catalyzed reactions is shown in Figure 12.

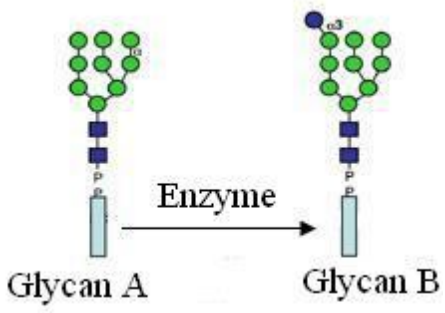

Figure 12: An enzyme catalyzed reaction [Nimmagadda 2008]

Signal-Transduction Pathways allow cells to perceive, process and respond to information incoming from outside the cell.

Gene regulatory pathways focus on the cells operating at the gene level. They orchestrate the level of gene expressions on each cell, by deciding if a gene should be transcribed into RNA and if yes, how it will be done.

The N-Glycan biosynthesis pathway is a type of a metabolic pathway.

Petri Nets

A Petri-net is a mathematical modeling language used to describe discrete distributed systems. The formal definition of a Petri-net is given below [Petri, 1994]:

Petri Net is a five tuple set, (P, T, F, W, M0), where:

P = {p1 , p2 , ..., pm }, finite set of places

T = {t1 , t2 , ..., tn }, finite set of transitions

$F \subseteq (P \times T) \cup (T \times P)$, set of arcs

$W : F \rightarrow \{1, 2, ...\}$, weighting function

$M0 : P \rightarrow \{0, 1, ...\}$, initial marking

The two types of vertices are places and transitions. Places are conditions and transitions are the discrete events that may occur. Directed arcs connect places to transitions or transitions to places but never a place to a place or a transition to a transition. Arcs may have weights assigned to them which define the amount of the place that must be produced before it can be supplied to a transition. Petri-nets were invented by Carl Adam Petri to describe chemical processes [Petri, 1962], aptly suiting our purpose of modeling a biochemical pathway as a Petri-net.

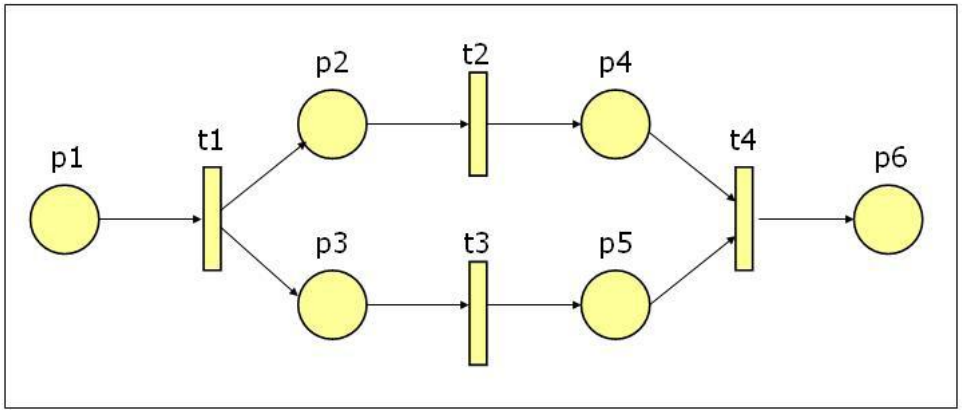

Figure 13: A Petri-net with four transitions and six places [Nimmagadda 2008]

A Petri-net is shown in Figure 13. The rectangular boxes represent transitions; the circles represent places – a transition may have both input places and output places. And the lines with arrowheads represent the directed arcs. Figure 14 shows a classification of Petri-nets.

Petri Net Type	Description
Timed	A timed duration is associated with transitions and places
Colored	Each token can have its own color, gives more descriptive power
Hierarchical	A place can represent another Petri Net
Stochastic	Probability distribution delays are associated with places and transitions
Hybrid	Places can hold integers and real values

Figure 14: Classification of Petri-nets

Modeling a biochemical pathway as a Petri-net

The illustration in figure 15 is that of a pathway for N-Glycan Biosynthesis and that in figure 16 illustrates how a portion of this pathway is modeled as a Petri-net. By looking at these figures we can see how a pathway is analogous to a Petri-net. Molecules and enzymes in a pathway can be represented as places in a Petri-net. Reactions in a pathway can be represented as transitions in a

Petri-net. Hybrid modeling methodologies can be used to model pathways and the most suited methodology used for biochemical simulation is the Hybrid Functional Petri Net (HFPN). The HFPN retains all the characteristics of a Petri-net and also provides means of accurately describing the continuous features of a biochemical pathway.

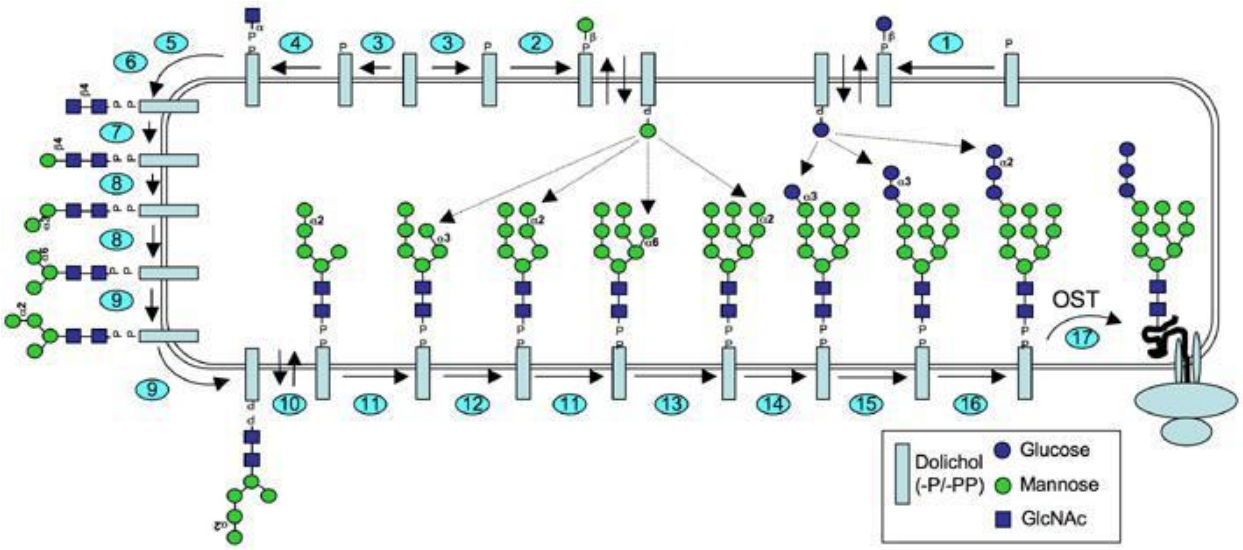

Figure 15: A Biochemical pathway [Mormen, 2008]

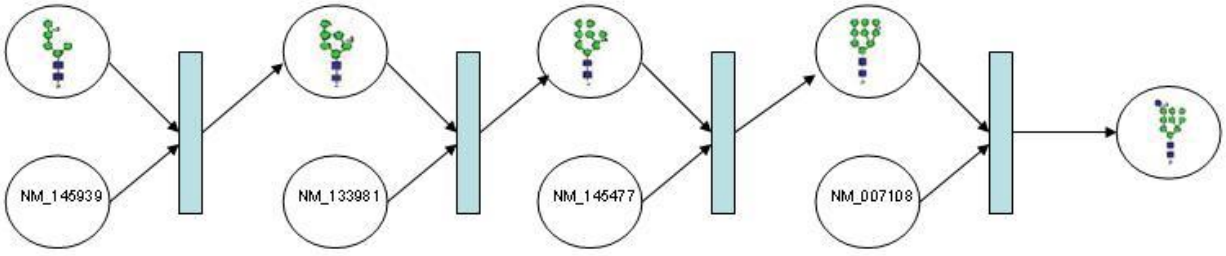

Figure 16: Portion of the pathway modeled as a Petri-net [Nimmagadda 2008]

APPENDIX B

USER GUIDE

Installation Instructions

1. Install Java (current system has been tested on Java 1.6): http://java.com
2. Install Ant: http://ant.apache.org/
3. Download and unzip DeMOForge.zip (current location: http://cs.uga.edu/~jam/downloads/DeMOForge.zip)
4. Download and unzip JSIM.zip (current location: http://cs.uga.edu/~jam/downloads/JSIM.zip)
5. Download the JESS API (license required for full version) from http://www.jessrules.com/jess/download.shtml and copy and paste jess.jar into DeMOForge/lib/ProtegeOWL
6. Go to the DeMOForge folder, and type 'ant' to compile
7. Go to the JSIM folder, and type 'ant' to compile

Execution and usage Instructions

DeMOForge - Mapping and Transformation tool

1. Run 'ant DeMOForge' from the DeMOForge folder.
2. Load the domain ontology and the DeMO ontology (Figure 17 and Figure 18).

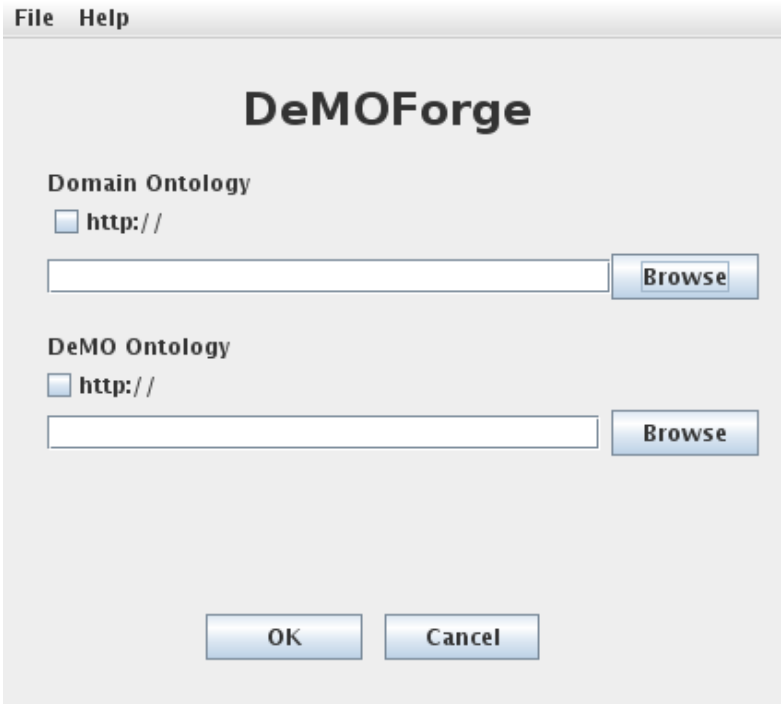

Figure 17: Ontology Load Screen

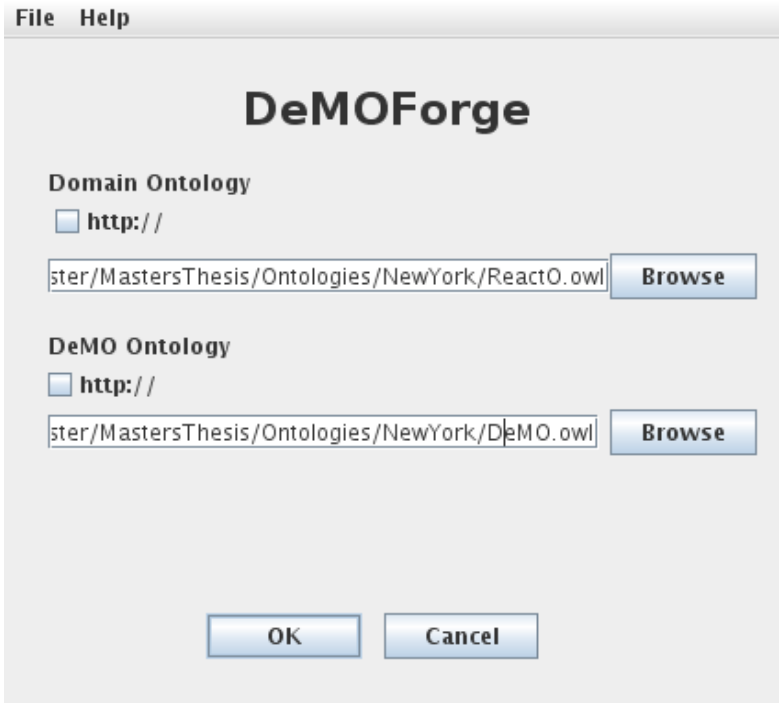

Figure 18: Ontologies selected

3. The mapping tool is loaded as shown in Figure 19: The class hierarchies of the domain ontology and the DeMO ontology can be seen side by side. The status bar is at the bottom of the screen.

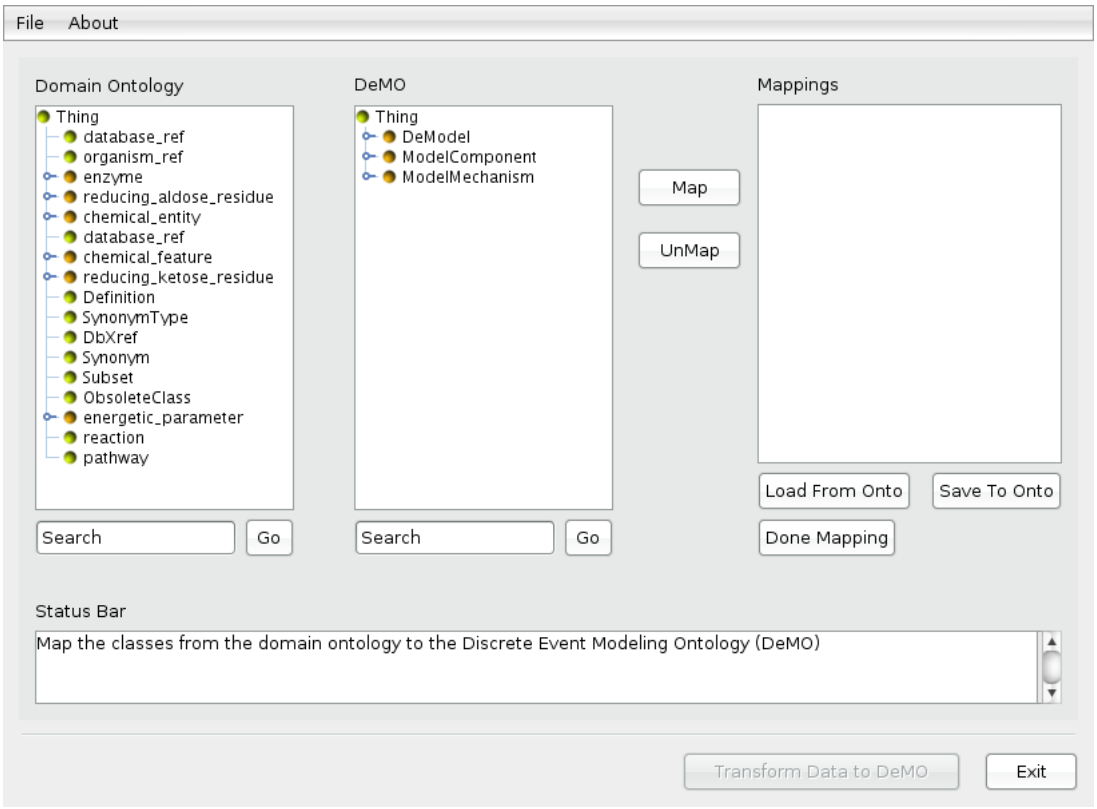

Figure 19: DeMOForge Mapping tool

4. Select the conceptual model class from the domain ontology and select the DES model formalism class that it corresponds to in the DeMO ontology and click ‘Map’ (Figure 20). Classes can be browsed for through the class hierarchy trees or the user can enter a search term in the search box just below the tree display.

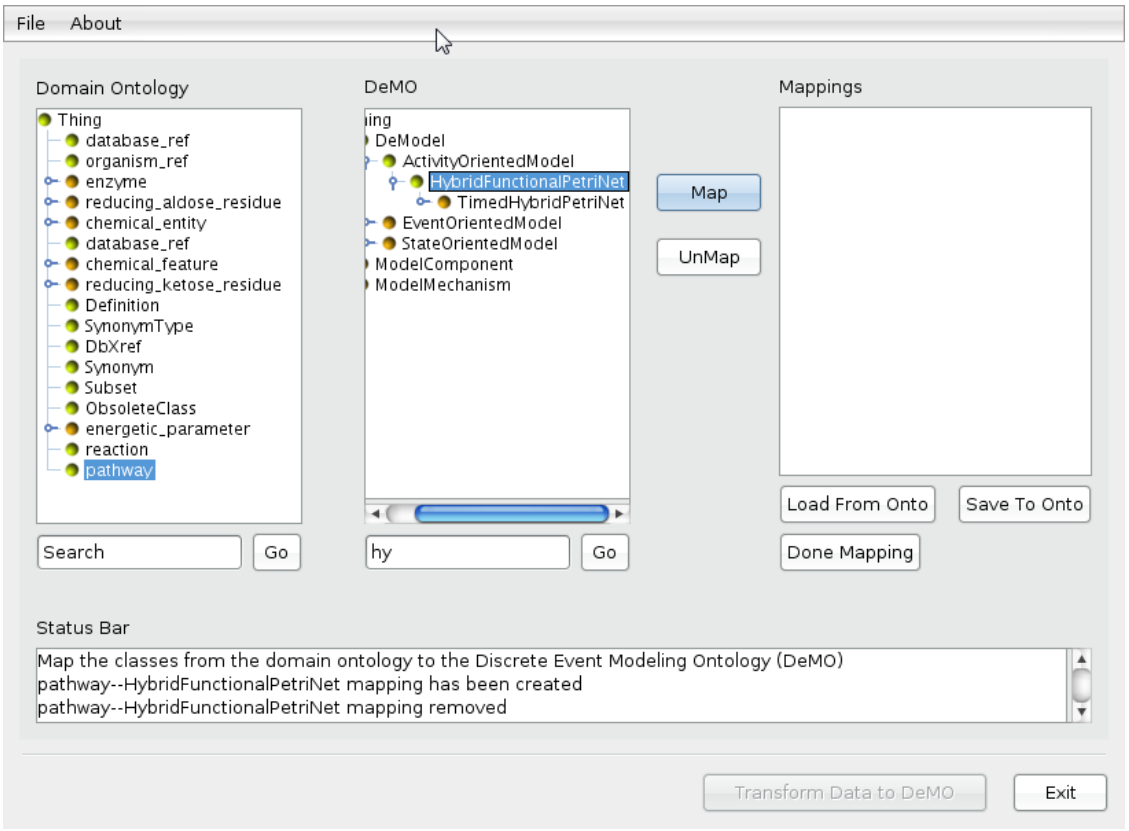

Figure 20: Aligning and mapping the ontologies

5. As soon as the model classes are mapped, both the ontologies are trimmed and only those concepts related to the particular model classes are displayed as shown in Figure 21.

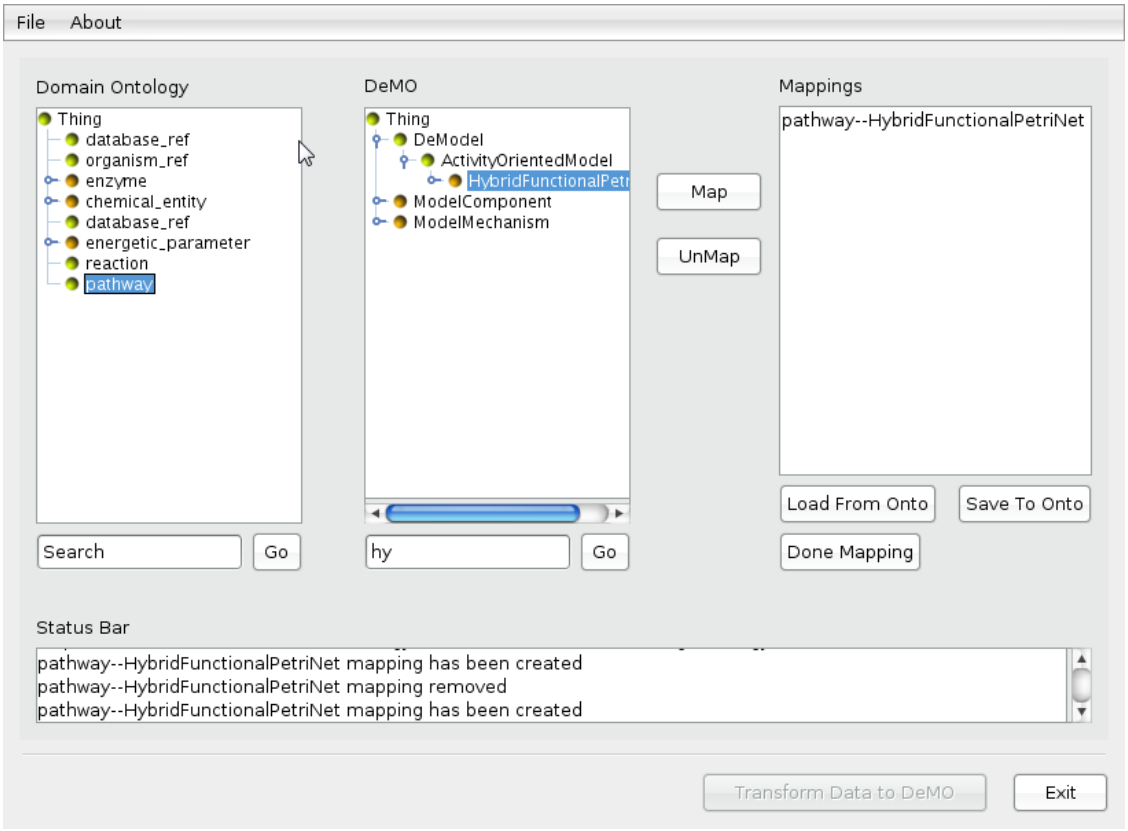

Figure 21: Trimmed Ontologies

6. Continue with this process until all the corresponding classes have been mapped as shown in Figure 22.

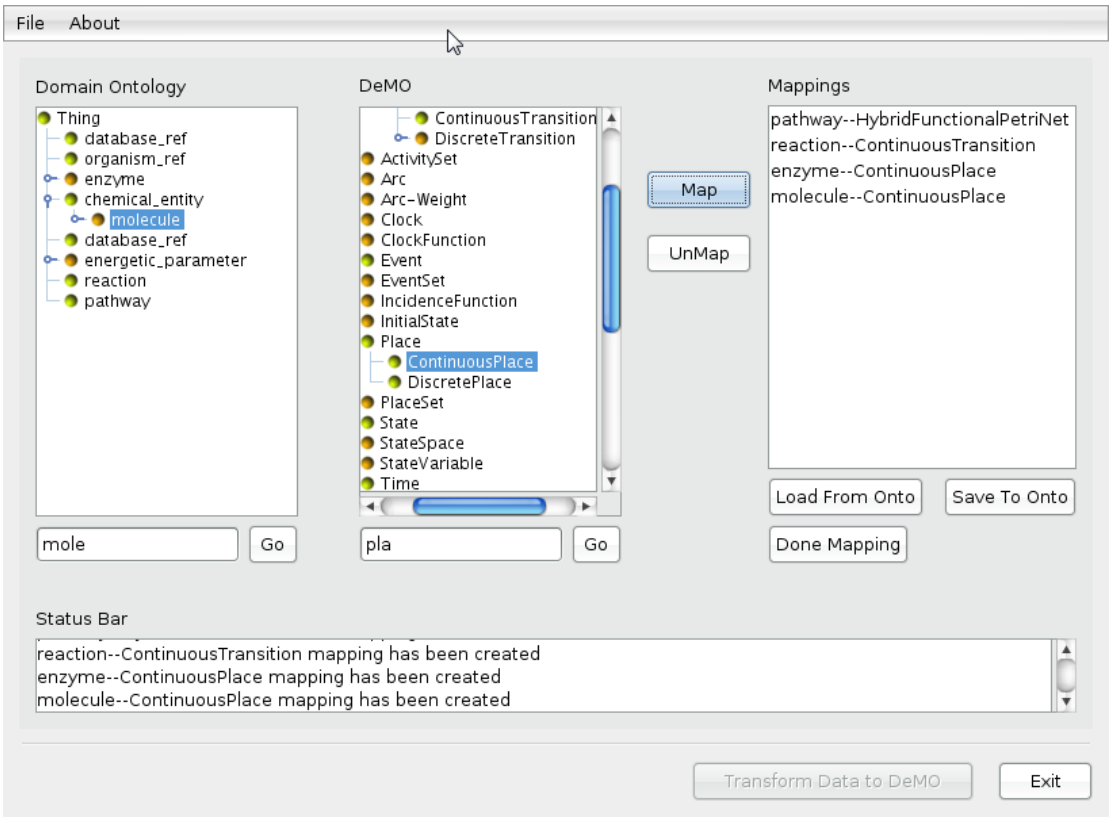

Figure 22: Done Mapping

7. At any point, the 'Save To Onto' (Save to Ontology) button can be clicked to save the mappings in DeMO and the 'Load To Onto' button can be used to load mappings from DeMO.
8. For complex mappings, such as that for a firing rate equation, which involves the formation of an equation, based on the data-type values held by one or more individuals, the user can use the following convention to create equations in the Equations.txt file.

   DES_class_name = [str[] + swrlImp[path,replaceClass,dataTypeValue]]*

   str[] - can contain any string that is to be used in the equation

   swrlImp[] - can contain a specific SWRL query to retrieve instances

   path - path used to retrieve instances OR SWRL query

replaceClass - replace this particular class with its instances

datatypeValue - specify data type property associated with the last class in the path, null if individual label is to be used

* - any combination of str[] and swrlImp[] separated by a '+' sign

Example:

firing_rate_equation = swrlImp[enzyme --> has_kinetic_constant --> kinetic_constant,enzyme,has_float_value] + str[[S]/] + swrlImp[enzyme --> has_Michaelis_constant --> Michaelis_constant,enzyme,has_float_value] + str[+[S]]

9. Click the 'Done Mapping' button once all the mappings are created.
10. The red colored mappings (Figure 23) are alerting the user as additional information is required pertaining to these mappings.

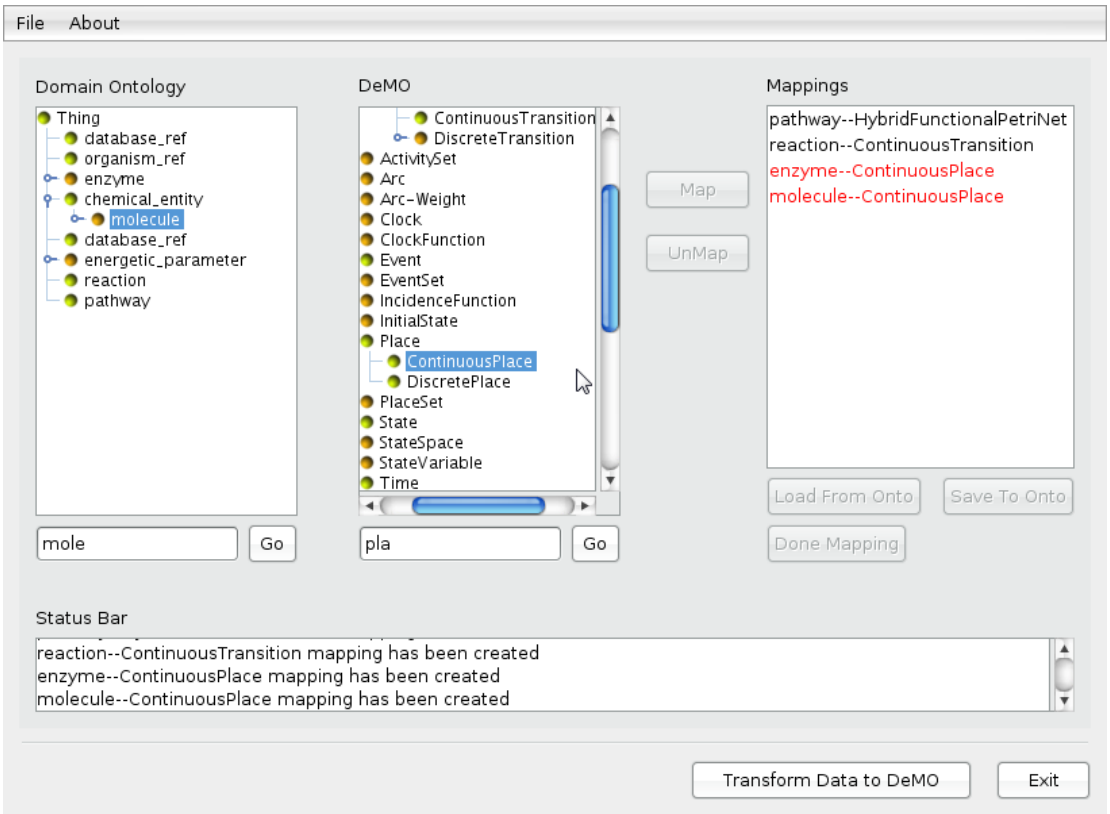

Figure 23: Red colored mappings

11. Click on the red colored mappings to resolve them. The 'Additional Details Required' window pops up (Figure 24) and the user has to select a path which will be used to retrieve the instances. The class listed at the end of the path is the class for which instances will be retrieved. For example reaction→consumes→molecule will return a different set of molecule instances when compared to reaction→generates→molecules. Along with the paths, the user may be asked to select an arc type if the node for which instances are being retrieved is a subclass of the Place class. In this case the user needs to select the arc that connects this place to its corresponding transition.

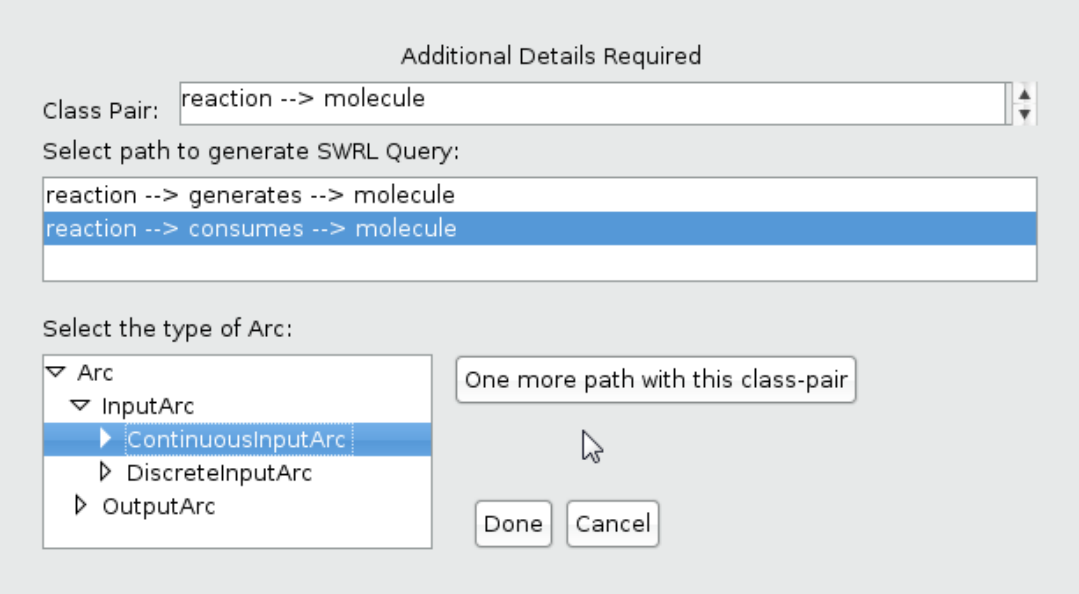

Figure 24: Additional Details Required

12. If more than one of the paths is relevant in the context, click 'One more path with this class-pair' and repeat step 9 (Figure 25).

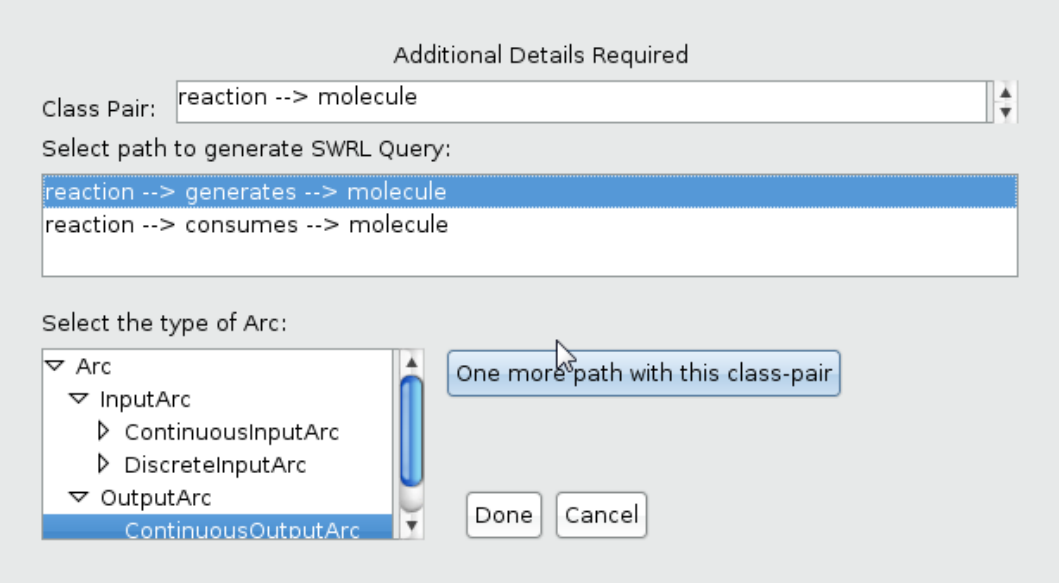

Figure 25: One more path with this class-pair

13. Once all the mappings have been resolved, the user sees that the red colored mappings are reverted to the normal black color as shown in Figure 26. The user can now click the 'Transform to DeMO' button.

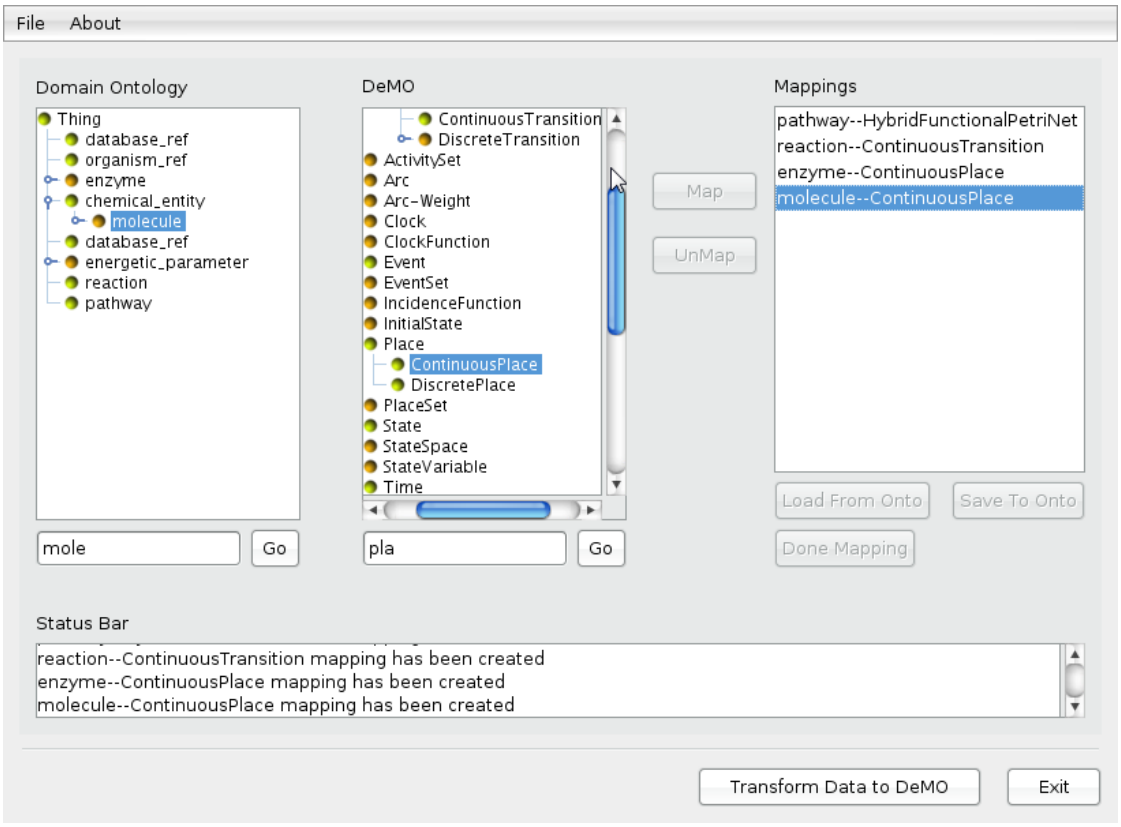

Figure 26: Red colored mappings resolved

14. The user has to select the instance of the model which he wants to transform (Figure 27)

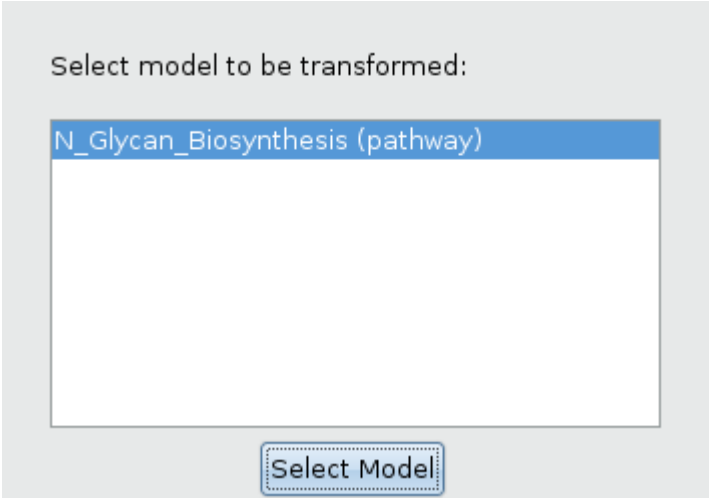

Figure 27: Model instance selection for transformation

15. The transformation of the model from the domain ontology to DeMO has been done. The user can now click the ‘Exit’ button (Figure 28).

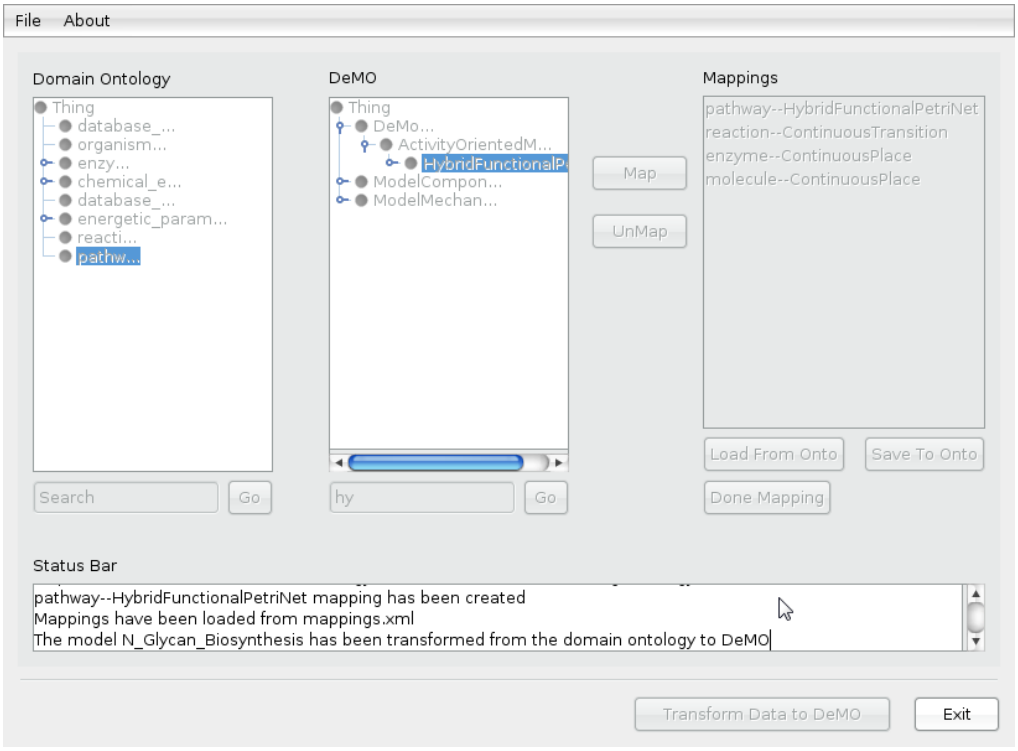

Figure 28: Transformation complete

DeMOForge - Mapping and Transformation tool

1. Run 'ant CodeGenerator' from the JSIM folder.
2. Load the DeMO ontology and click and click ‘Browse’ model (Figure 29).

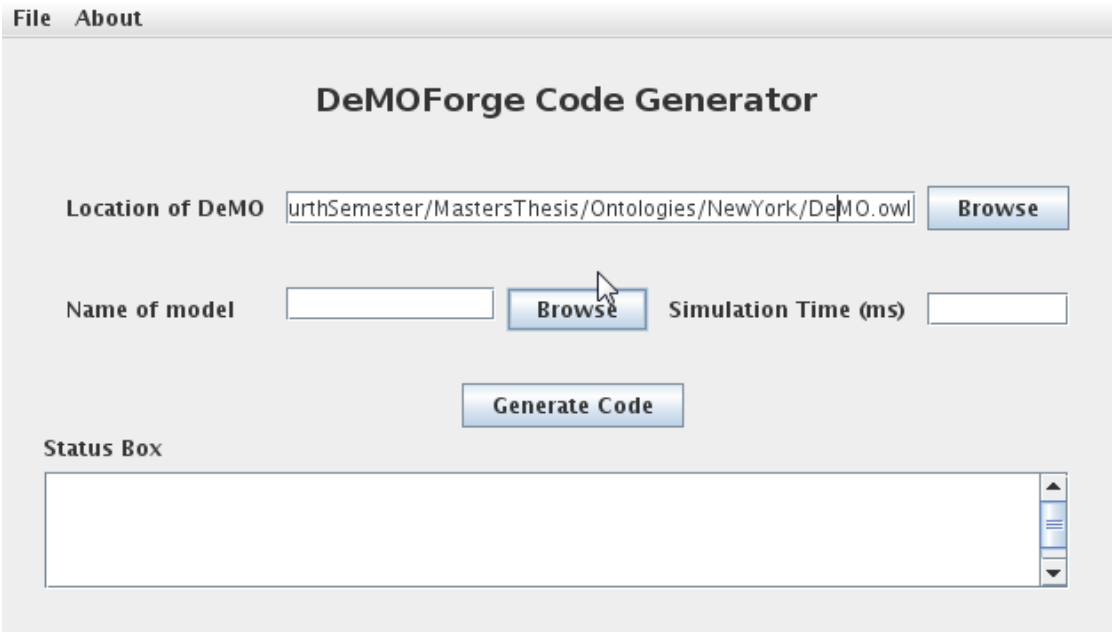

Figure 29: DeMOForge Code Generator – Load DeMO

3. Select the instance of the model to be simulated as shown in Figure 30.

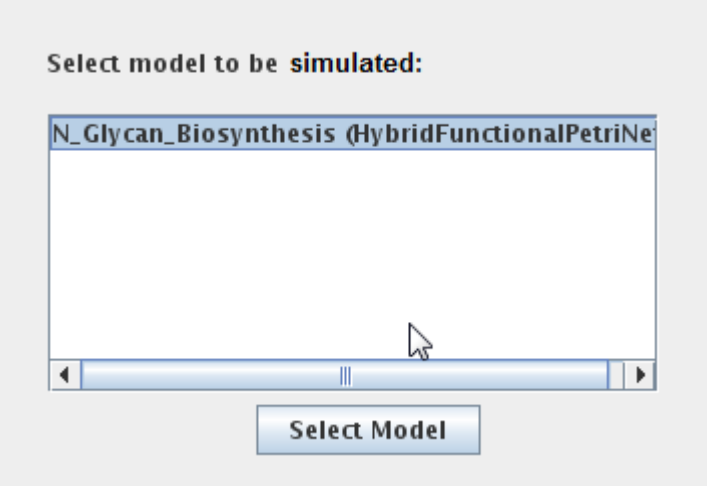

Figure 30: Model Instance selection for Simulation

4. Enter the simulation time (Figure 31) and press the 'Generate Code' button.

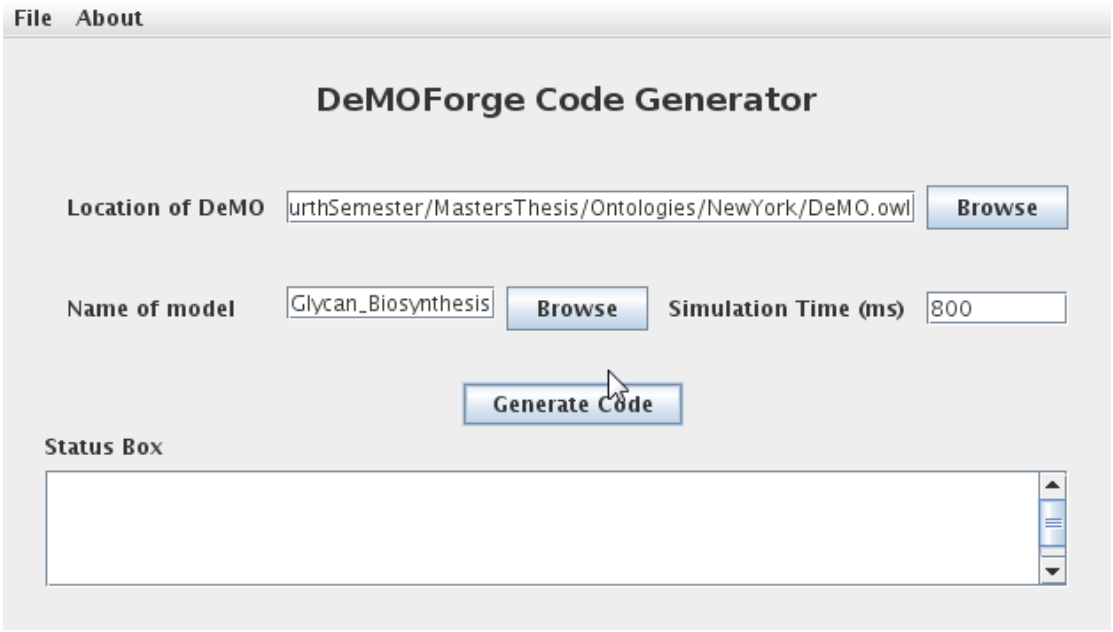

Figure 31: Press 'Generate Code'

5. The tool may need some additional data and it will prompt the user for it (Figure 32). The user can feed the data from a spreadsheet, a database or an ontology.

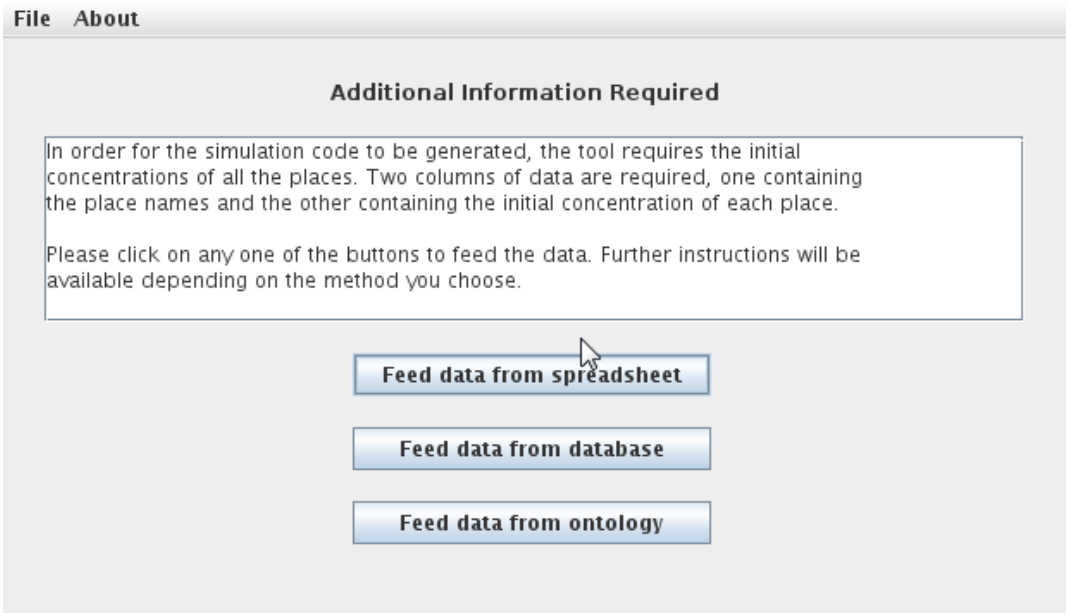

Figure 32: Choices for data feed

6. In this case, the data is fed from a spreadsheet and the user has to enter the column numbers which contains the place names and the place concentrations (Figure 33).

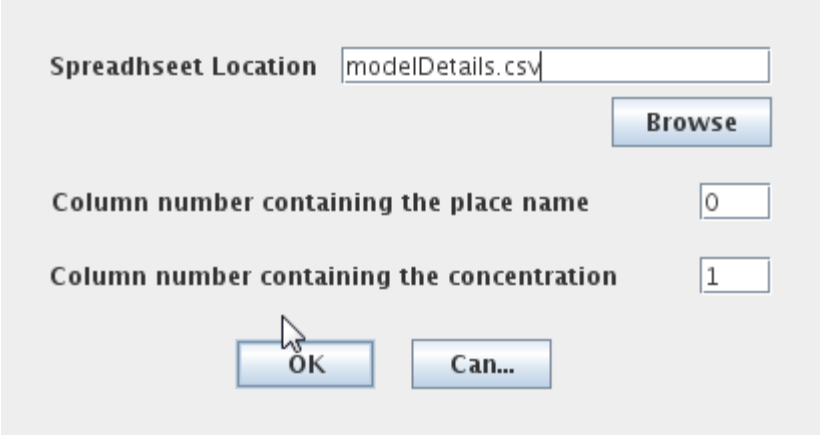

Figure 33: Data feed from spreadsheet

7. The code will be generated. The user can type 'ant ModelSim' from the root JSIM folder in order to simulate the model and view the animation.

APPENDIX C

DEMOFORGE JAVADOC

The JavaDoc for the DeMOForge tool can be found at the following download location:

http://cs.uga.edu/~jam/downloads/DeMOForge.zip.

Unzip the above file and open DeMOForge/doc/index.html

The JavaDoc summary of the 'demoforge' package is shown in Figure 34.

Package demoforge

DeMOForge Mapping and Transformation Tool

See:

Description

Class Summary	
ClassDetailsCore	This class contains the methods that describe the functionalities of the 'Additonal Details' GUI which pops up in the mapping process.
ClassDetailsGUI	Constructs a GUI to accept additonal details pertaining to class-class mappings
ClassUsage	The static methods of this class are used to get the class-usage (Protege style) of a particular class
ConceptPath	Structure to store the details of a path used in retreiving the instances of a class
DeMOForge	This is the main class of the DeMOForge Mapper and Transformer
DemoForgeCore	A convencience class which has the parameters to store the Ontology paths, OWLOntologies and OWLOntologyManagers
DemoForgeGUI	The GUI of the Mapping tool and the tranformation tool
MappingCore	The core functionality of the mapping tool
MappingProcessor	Methodology for processing the mappings once DoneMapping is clicked
ModelSelectionGUI	GUI for model selection
Node	The structure of a node in the model graph
OntoLoadGUI	GUI for loading ontologies
SwrlQueryExec	This class contains the methods to automatically create and excecute SWRL Queries
Transformer	The class which performs the transformation of a model from the domain ontology to the DeMO ontology
TreeTrimmer	Contains methodology for trimming ontology visualisations
XMLMappings	Load and Store Mappings in an XML format

Figure 34: JavaDoc summary of the 'demoforge' package