User-Guided Transformations for Ontology Based Simulation Design

Ozer Ozdikis
Norwegian University of Science and Technology

Umut Durak
German Aerospace Center (DLR)

Halit Oğuztüzün
Middle East Technical University

User-Guided Transformations for Ontology Based Simulation Design

Özer ÖZDİKİŞ, Siemens EC, ozer.ozdikis@siemens-enterprise.com
Umut DURAK, TUBİTAK-SAGE, udurak@sage.tubitak.gov.tr
Halit OĞUZTÜZÜN, Middle East Technical University, oguztuzn@ceng.metu.edu.tr

Keywords: Ontology based simulation, domain engineering, ontology languages, object-oriented design methods, model driven simulation development.

Abstract

Using domain knowledge represented as ontologies for the design of domain specific architectures is a promising approach for better reusability. Tool support is essential for this approach to be effective. We present a flexible, user-guided transformation method to generate a framework architecture model from a domain model. The latter is in the form of an OWL ontology, and the former is in the form of a UML class diagram. We introduce a transformation tool that allows the user to configure mappings from the source OWL constructs of the simulation ontology to the target UML constructs of the simulation design. This user-guided approach is demonstrated on a case study that involves building a framework architecture for ontology-driven trajectory simulation development.

1. INTRODUCTION

Transformation of domain models, in the form of OWL ontologies, to framework architectures, in the form of UML class diagrams, is an issue that arises in the context where domain engineering and model-driven development meet. We hold the view that this issue can be resolved to the satisfaction of the user with the aid of the user-guided model transformation tools.

The use of models to represent the outcomes of domain engineering activities, and the use of model transformations and code generation as a means to carry out these activities bring Model Driven Development (MDD), in particular, OMG's Model Driven Architecture (MDA), to bear on Domain Engineering [1], [2].

As information is gathered during domain analysis, representing the knowledge for ease of both human understanding and machine processability becomes a problem [3]. Using ontologies is proposed as a solution to this problem [4]. Using ontologies as domain models is called ontology based domain engineering. One of the promises of ontology based domain engineering is to derive reusable components from ontologies [5]. Referring to the potential benefits of ontologies for modeling and simulation that are listed by Miller and Fishwick in [6], in ontology based domain engineering applied to simulation development, ontologies are used as domain models that enable simulation reuse at knowledge, design and code level.

Ontology as a domain model, however, is the subject matter expert's view of the domain. Tool-supported methods of generating the software developer's view of the domain is desirable. Our present focus is on tool support for flexible transformations from a domain ontology into a UML class diagram.

As an ontology language, W3C's OWL is a well accepted semantic Web standard [7], [8]. When it comes to transforming available OWL ontologies into UML class diagrams, the "one size fits all" approach does not work. Because every domain model may require a different transformation procedure depending on the context, data types, conventions, and even the personal preferences of the software engineer who specifies the reuse infrastructure. A hard-coded transformation approach would not be flexible enough in general to be effective in practice. What we propose is a configurable transformation approach: Mappings from OWL ontology constructs to UML class diagram constructs will be guided by the user.

Our implementation for this approach provides a user interface to design mappings between the OWL and UML metamodel constructs. These mapping definitions are applied on the OWL ontology (domain model), and a UML class diagram (as a constituent of a framework architecture) is produced. This reconfigurable transformation tool has been employed in an industrial development effort, in which a domain ontology, called Trajectory Simulation ONTology, in short TSONT [9], is transformed to an object oriented application framework architecture. This case study will be discussed at length.

2. BACKGROUND

Three levels of abstraction are identified in OMG's Model Driven Architecture (MDA) for the modeling of systems [1], [2]. These are Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). CIM is a view of a system from the computation independent viewpoint. PIM is a view of a system from the platform independent viewpoint. PIM describes the system, but does not refer to the platform on which the system is implemented. PSM, in contrast, refines the PIM with the details that specify how that system utilizes a particular platform. PSM is supposed to be readily transformable to executable code.

The idea is to capture domain knowledge at a conceptual level with CIM and generate the models at PIM and PSM as automated as possible, to finally get a running code. Ontologies can serve as CIMs. They are the representation of the domain information prepared by the domain and knowledge engineers with the help of subject matter experts, without any concern about system development. Our focus is to allow the developer to guide the transformation from a CIM, which is represented by an OWL ontology, to a PIM, which is represented by a framework architecture description involving UML class diagrams.

Our transformation method from CIM to PIM (i.e. from OWL to UML) can be located in reference to the four-layer metamodeling hierarchy of OMG's Meta Object Facility (MOF) [10]. MOF is an extensible model driven integration framework for defining, manipulating, and integrating metadata and data in a platform independent manner. It provides a meta-metamodel at the top level, which is called M3 layer of the four-layer metamodeling hierarchy. MOF can be used to define more specific metamodels at M2 layer, such as the UML metamodel. We have defined a simplified simplified UML metamodel complying with the OMG's specification [11]. UML models conforming to the UML metamodel are at layer M1. Finally, the M0 layer includes the instances created from a UML model.

Similar to UML models, ontology models must also conform to a metamodel. Our ontology modeling hierarchy is based on Eclipse Metamodel Framework (EMF) [12]. The meta-metamodel at M3 layer in EMF is called Ecore. IBM implemented an Integrated Ontology Development Toolkit (IODT) for ontology driven development built on EMF [13]. IODT includes a library called EMF Ontology Definition Metamodel (EODM) which is an implementation of the OMG's Ontology Definition Metamodel [14]. EODM has OWL parsing, serialization, and reasoning features.

In the transformation from CIM to PIM, we have an ontology model as our source and a UML model as our target. Our approach facilitates the definition of the mapping rules between the EODM and UML metamodels at M2 layer. These rules are applied to a given OWL ontology to generate a UML class diagram. The modeling layers which are used in this transformation are shown in Fig. 1.

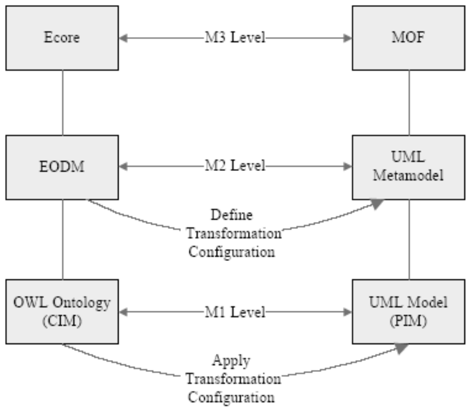


Fig. 1 - Relations between the modeling layers

3. THE OWL-TO-UML TRANSFORMATION TOOL

Our tool lets the user configure the transformation he needs. The transformation configuration is composed of transformation rules, mappings and constraints.

A rule is a collection of mappings from a specific OWL construct to some UML constructs. In other words, a rule includes mappings and a definition of source OWL constructs to apply these mappings. Source constructs can be OWL classes or OWL properties.

Following the description of the source construct, the user must specify, in terms of mappings, how to use them in the transformation. Depending on the source-target combinations, different mapping possibilities are provided to the user. The user can define three types of mappings in a rule: to a UML class, to an association, and to an operation parameter. A mapping is the prescription of how the target UML construct should be built using the source OWL construct.

Constraints can be defined to restrict the entities to be evaluated in a specific rule or mapping. Constraints are actually condition-value pairs applied on an OWL construct. While evaluating an OWL construct in a rule or mapping, these condition-value pairs are used to find out if that construct is selected and should be processed.

3.1. Available OWL Constructs at the Source

An OWL ontology involves classes, object properties, and datatype properties. A class has a classname and possibly super classes. A class may be defined as an intersection of, union of or complement of other classes. Further, a class may have different types of restrictions. These restrictions define the values (restrictionValue) that the class must take for a property (onPropertyName). A class may also be an enumeration class, which includes the list of individuals that are the members of the class. A property has a property name, domain and range information, and possibly super properties. Our tool lets the user use definitions of classes, object properties, and datatype properties as a source for the mappings to UML.

3.2. Available UML Constructs at the Target

A UML model can include the definitions for the UML classes, generalizations, associations, attributes, operations, and parameters for operations. A class has a name and possibly some attributes and operations, which are identified by their names within a class. An operation can have parameters of type "in", "out", "in-out", and "return". There can be generalizations and associations defined between classes. Generalizations represent the subclass-superclass relationships. Associations can be of type aggregation, composition, or directed association, possibly with multiplicity information attached at either end. One restriction in this metamodel definition is that the class names in the model and operation names in a class must be unique. This is because the OWL classes in the ontology are also identified with their names.

3.3. Mappings from source OWL to target UML

Our tool provides an interface to the user to define mappings between the available OWL constructs at the source and the UML constructs at the target, which were defined above. Mappings can be classified into 3 types regarding the target constructs. These are

- Mappings to UML classes
- Mappings to UML associations
- Mappings to UML operation parameters

During the transformation process, mappings to UML classes are resolved first. After the classes are ready in the target UML model, mappings to associations, and finally, operation parameters are evaluated. The name of the ontology is carried over to the target model as the package name.

3.3.1. Mappings to UML Classes

This mapping type is used to introduce new classes into the UML model, with their super class relationships, attributes, and operation names. Depending on the type of the rule, OWL classes or OWL properties in the input ontology are traversed considering the constraint definitions. For each selected OWL class/property, a new UML class is created in the UML model with the same OWL class/property name. If a class with the same name already exists in the UML model, it is not created again since the class names are supposed to be unique in a UML model according to our metamodel definition. In addition to the new UML class definitions, user can define further mappings between the OWL and UML class constructs. The available OWL constructs can be used to define the super classes, attributes, or operation names of the corresponding UML class.

3.3.2. Mappings to UML Associations

This mapping provides the opportunity to define associations between the UML classes in the target UML model. OWL classes/properties in the input ontology are traversed according to the constraint defined for the rule. The name of the currently traversed OWL class/property is used to identify the first end point in the UML association. Second end point in the association is found depending on the rule type and the mapping configuration. In the mapping definition, the user selects the association type to be used, where the choices are aggregation, composition, and directed association.

In the mapping, in addition to the association type, it is also possible to define constraints for each unique association definition. A special case for the min/max cardinality restrictions is that the user can define multiplicity of the association. The min/max cardinality restrictions have integer values. If the second association end is to be taken from the onPropertyName of the restriction, value of the restriction can be attached to the association as a multiplicity. This value can be used as exact, lower bound or upper bound multiplicity. End1 or end2 can also be selected for the desired placement of the multiplicity.

3.3.3. Mappings to UML Operation Parameters

There may be cases where some class in the ontology defines an operation with parameters. This mapping is used to fill the parameter names of operations in UML classes. The transformation for this type of mapping works as follows: OWL classes/properties in the input ontology are traversed one by one according to the constraint definitions for the rule. For each selected OWL class/property named C, an operation with name C is searched in UML classes of the target model. For each operation named C, the OWL construct defined in the mapping is used to feed the parameter names to this operation. This mapping type is evaluated as the last mapping type, so that we already have the operation names available in the UML classes of the result model.

The precondition for this mapping is that the correct operations have already been defined in desired UML classes. This is achieved by the operation name mappings in UML class transformations. Thus, mapping to UML operation parameters is just a matter of feeding the parameter names to the previously defined operations of previously defined classes.

The name of the owner UML class can be bound to a constraint in the mapping. In the example scenario, with a constraint defined for the owner class name, hasUnit could have been filtered out. So the parameter settings for Quantity operation in this UML class would have been ignored. It is also possible to define conditions on each individual parameter name. Parameter types can be defined as one of the four specified in the standard [11], namely “in”, “out”, “in-out”, or “return”. Using the constraint definitions on parameter names, user can add different types of parameters to an operation in different mapping configurations.

We provided graph panels to the user for the design of mappings between the OWL and UML constructs. Each graph corresponds to a rule. On these graphs, cells that represent the OWL and UML constructs are tied to each other, which correspond to mappings in a rule. Each graph includes exactly one cell for the OWL construct and one or more cells for the UML constructs.

The graph cell for the OWL construct represents the OWL classes or properties in the source, depending on the rule type. This graph cell enables the user to define constraints on the source OWL constructs.

The type of the UML graph cell defines the target UML construct to which we want to transform the constructs in the source. There are 3 types of UML graph cells corresponding to the 3 different mapping types. What is visualized within the graph is an abstract view of the mapping. User sets the details of the mappings by right clicking on the graph cells and making configurations on separate windows. The whole transformation configuration can be exported to files so that it can be imported and used later again.

Rules, mappings, and constraints can be defined in any order. Once they are defined, user can start the transformation process. The tool traverses all rule definitions, identifies the selected OWL classes/properties according to the constraints and adds UML classes to the UML model. After this step, all UML classes will be available in the UML model to define their operations, attributes, superclass relationships, and associations. This is achieved by a second traversal of the mapping graphs. After all the mappings are processed, all UML structures in the target model are serialized into an XMI file [15]. This XMI file can be imported and interpreted by any standard-compliant UML tool.

4. CASE STUDY: FROM TSONT TO SIMULATION FRAMEWORK ARCHITECTURE

The tool is applied to build an ontology based trajectory simulation framework [16]. The trajectory simulation ontology called TSONT [9] serves as a domain model for the trajectory simulation development projects at TUBITAK-SAGE. TSONT is a domain model, where the domain is trajectory simulation, rather than, say, aerodynamics. Note that the mathematical models in a computable form and the methods of computing them are included in TSONT. Thus, TSONT essentially serves as a simulation conceptual model, in the sense of Pace [17]. Moving from such a model to an architecture model is in parallel with moving from an analysis model to a design model, in the software engineering sense, see Pressman [18].

A development methodology is formulated to reuse this ontology based on Model Driven Engineering. In the reuse scenario, the domain model (i.e. TSONT) is transformed to the Platform Independent Framework Architecture as a step of the abstract design process. This transformation is accomplished with our transformation tool presented in this paper. The same transformation was previously carried out by hand.

There are six main classes at the top of the trajectory simulation ontology [9]. The most important for the transformation is the Trajectory_Simulation_Class, because each of its subclasses should be represented as a UML class in the resulting model. Operation and attribute names are adapted from the OWL restriction definitions for each class. Subclasses of Trajectory_Simulation_Function provide the operation parameter information. The hierarchy including some selected example classes in TSONT is shown in Fig. 2.

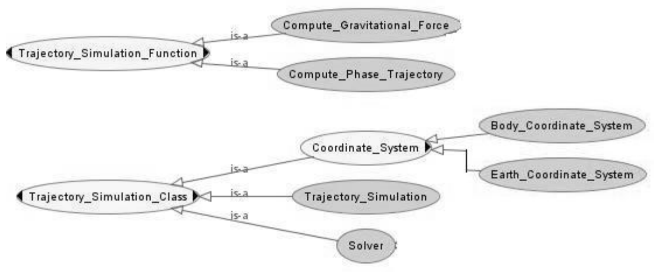


Fig. 2 - Selected Example Classes from TSONT

We define two rules to fulfill the transformation requirements for the trajectory simulation ontology. In the first rule, we define mappings for UML classes and UML associations using the subclasses of Trajectory_Simulation_Class as our source. The second rule is used for mappings to UML operation parameters, essentially, for setting the parameter names in the target UML operations. Since this information is taken from the subclasses of Trajectory_Simulation_Function, we had to define a separate rule to have a separate set of source OWL classes. The first rule can be described as “the rule for classes and associations”, whereas second rule as “the rule for operation parameters”.

Rule-1: Rule for classes and associations:

The mappings defined in this rule must be applicable only for a restricted set of OWL classes. The criterion to find the source OWL classes is the subClassOf relationship with the Trajectory_Simulation_Class. So we define a constraint in this rule for subclasses, with condition "equals" and constraint value "Trajectory_Simulation_Class". As a result, the source OWL classes are restricted to be the subclasses of Trajectory_Simulation_Class for all mappings in this rule.

This rule will include five mappings. With these mappings, (1) the UML classes will be selected, (2) the subclass-superclass relationships will be defined, (3) operation names in the target UML classes will be selected, (4) attribute names for the target UML classes will be defined and (5) associations between the target UML classes will be created. As a result, for this first rule and its 5 mappings, we design a configuration in our transformation tool. A screenshot from our transformation tool for this rule is shown in Fig. 3. It is represented with the tabbed panel labeled "Rule0".

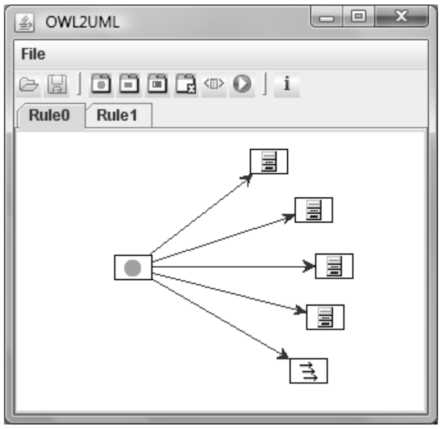
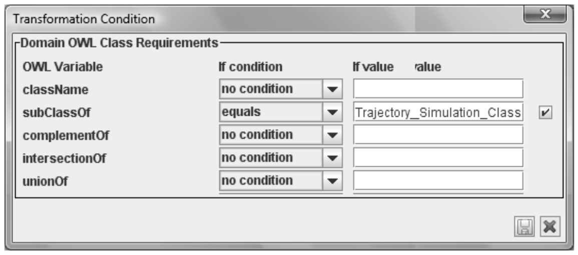


Fig. 3 - Configuration for the first rule

In the above panel, the icon on the left, which is the source of the mapping arrows, represents the source OWL constructs in the source model. Right clicking on this icon opens a window to define constraints for this rule. In this window we define the constraint which is shown in Fig. 4.



Transformation Condition

Domain OWL Class Requirements

OWL Variable	If condition	If value value	
className	no condition		
subClassOf	equals	Trajectory_Simulation_Class	✓
complementOf	no condition		
intersectionOf	no condition		
unionOf	no condition		

Fig. 4 - Constraint for the first rule

According to the constraint in Fig 4, only the OWL classes which are subclasses of the Trajectory_Simulation_Class will be processed, and others will be ignored.

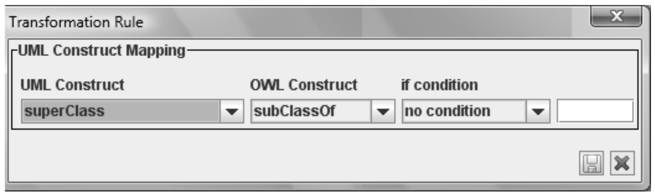
The first 4 icons on the right side of Fig.3 represent mappings to UML classes, whereas the 5th icon is a mapping to UML associations. User can add any number of mappings in a rule. In this scenario, we added 4 class mappings and 1 association mapping in our first rule. Right clicking on these icons opens new windows to configure each specific mapping.

Mapping-1: Definition of OWL Classes:

All OWL classes which are subclasses of Trajectory_Simulation_Class must be created as a UML class. This is accomplished by adding a UML class mapping. With this mapping, the tool traverses all OWL classes and if an OWL class is found to be the subclass of Trajectory_Simulation_Class, it adds a new UML class to the target UML model. The name of the new UML class is same as the OWL class name. As a result of this definition, we will have the classes like Coordinate_System, Body_Coordinate_System, Earth_Coordinate_System, Solver and all of their subclasses in the resulting UML model.

Mapping-2: Definition of super classes:

One other requirement is that, if two OWL classes have a subclass relationship, this relationship should also be shown in the UML model. This is achieved in the second UML class mapping with a mapping configuration for super classes. The mapping configuration is shown in Fig. 5. With this configuration, for each subClassOf relationship in OWL, a new super class relationship is added in UML between the corresponding sub/super classes. For example, Body_Coordinate_System will be represented as the subclass of Coordinate_System in the resulting UML model.



Transformation Rule

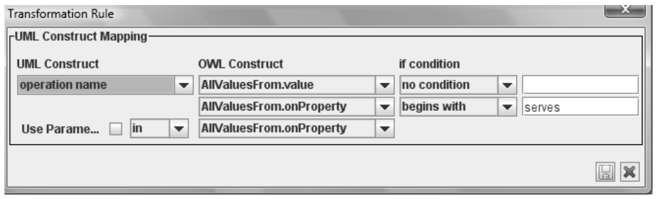
UML Construct Mapping

UML Construct	OWL Construct	if condition	
superClass	subClassOf	no condition	

Fig. 5 - Mapping configuration for super classes

Mapping-3: Definition of operation names:

The operation names for each class are to be taken from the AllValuesFrom restrictions if the restriction is defined on a property whose name has the prefix "serves". It is expected that if an OWL class has an AllValuesFrom restriction defined on a property named "servesX", all values at the restriction should be created as an operation for the corresponding UML class. As an example, the OWL class Trajectory_Simulation has an AllValuesFrom restriction on the property servesComputeTrajectory and the value of this restriction is Compute_Trajectory. So an operation named Compute_Trajectory will be added to the UML class Trajectory_Simulation as a result of this mapping configuration shown in Fig. 6.



Transformation Rule

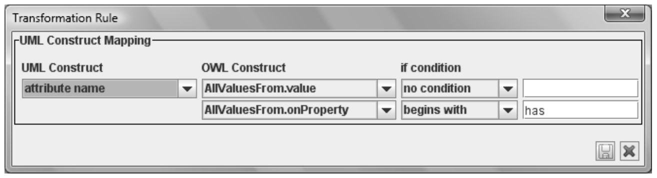
UML Construct Mapping

UML Construct	OWL Construct	if condition	
operation name	AllValuesFrom.value	no condition	
	AllValuesFrom.onProperty	begins with	serves
Use Parame... ☐ in	AllValuesFrom.onProperty		

Fig. 6 - Mapping for the definition of operation names

Mapping-4: Definition of attributes:

Attribute information for each class is to be taken from the AllValuesFrom restriction. However in this case, the restriction must be defined on a property whose name has the prefix “has” and the value of the restriction must be used as attribute names of the target UML class. To achieve this, we add a fourth UML class mapping and configure it as in Fig. 7. For example, the OWL class Trajectory_Simulation has an AllValuesFrom restriction on the property hasPhase and the value of this restriction is Trajectory_Simulation_Phase. So an attribute named Trajectory_Simulation_Phase will be added to the UML class Trajectory_Simulation as a result of this mapping configuration.



Transformation Rule

UML Construct Mapping

UML Construct	OWL Construct	if condition	
attribute name	AllValuesFrom.value	no condition	
	AllValuesFrom.onProperty	begins with	has

Fig. 7 - Mapping for the definition of attributes

Mapping-5: Definition of associations:

Similar to the attribute definitions, associations must also be taken from the AllValuesFrom restriction values if the restriction is defined on a property whose name has the prefix “has”. The 5th icon in the rule panel in Fig.3 represents the association mapping to define aggregation associations. Using the same example as the attributes, we will have an aggregation association from Trajectory_Simulation class to the Trajectory_Simulation_Phase class in the resulting UML model.

Rule-2: Rule for operation parameters:

In Mapping-3 of the first rule, we defined which operations a UML class should have. Now we must define mappings to identify from where to get the parameter names for these operations. The source OWL classes for the parameter names are subclasses of Trajectory_Simulation_Function. So we add a new rule tab in our transformation tool, which is labeled as “Rule1” in Fig.8.

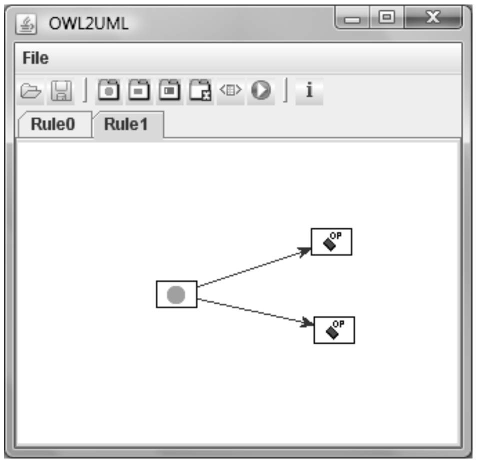
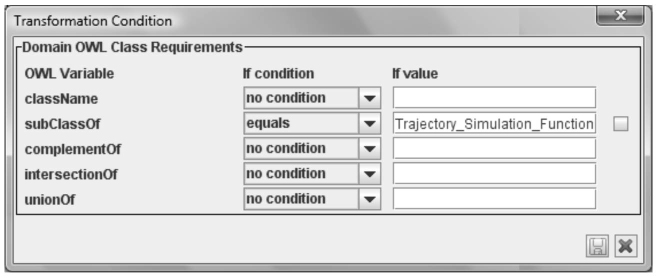


Fig. 8 – Configuration for the second rule

Since we must process only the subclasses of Trajectory_Simulation_Function, we define a constraint in this rule for subclasses, with condition “equals” and constraint value “Trajectory_Simulation_Function” as shown in Fig.9.



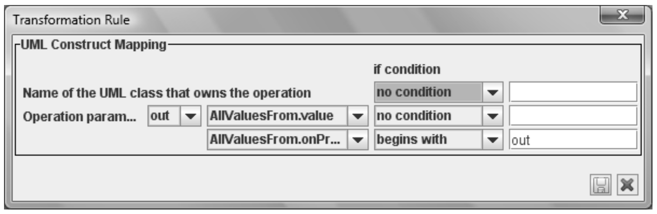
Transformation Condition

Domain OWL Class Requirements

OWL Variable	If condition	If value	
className	no condition		
subClassOf	equals	Trajectory_Simulation_Function	☐
complementOf	no condition		
intersectionOf	no condition		
unionOf	no condition		

Fig. 9 – Contraint definition for the second rule

Operation parameters are supposed to be taken from the OWL classes with the same name as the operation name in a UML class. These parameter names must be the values of AllValuesFrom restrictions. If the restriction is defined on a property whose name begins with “in”, the parameter type should be “in”. If the property name begins with “out”, parameter type should be “out” (Fig.10). For these “in” and "out" types, we define two operation parameter mappings in this rule.

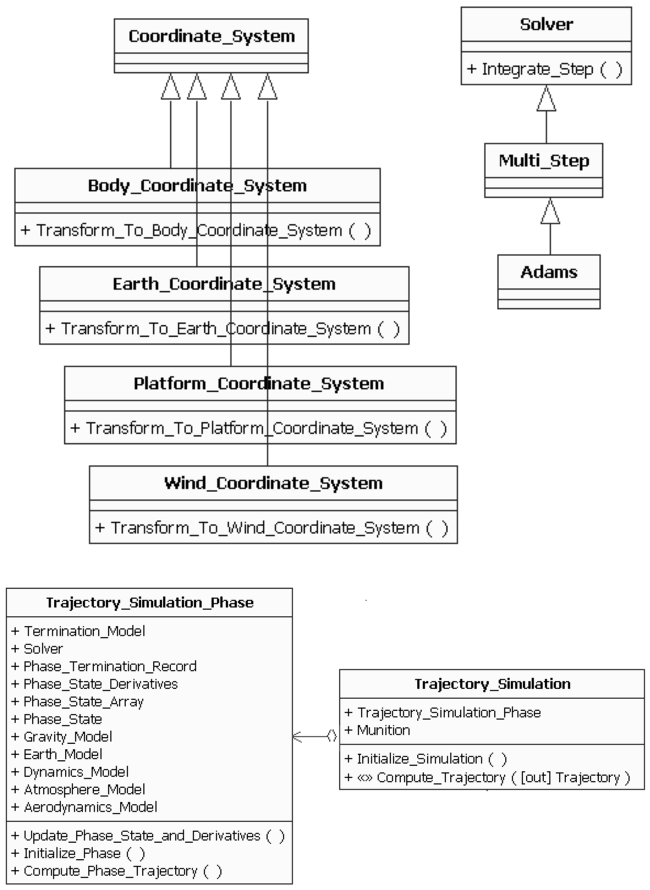

Transformation Rule

UML Construct Mapping

			if condition	
Name of the UML class that owns the operation			no condition	
Operation param...	out	AllValuesFrom.value	no condition	
		AllValuesFrom.onPr...	begins with	out

Fig. 10 - Mapping for the definition of parameters type "out"

For example Trajectory_Simulation class has the operation Compute_Trajectory as a result of Mapping-3. During the evaluation of the mappings in this rule, it is found that the Compute_Trajectory has an AllValuesFrom restriction on the property outTrajectory and the value of this restriction is Trajectory. So the parameter Trajectory of type "out" is added to the Compute_Trajectory operation in Trajectory_Simulation class of the target UML model.



Coordinate_System
Solver
+ Integrate_Step ( )
Multi_Step
Adams
Body_Coordinate_System
+ Transform_To_Body_Coordinate_System ( )
Earth_Coordinate_System
+ Transform_To_Earth_Coordinate_System ( )
Platform_Coordinate_System
+ Transform_To_Platform_Coordinate_System ( )
Wind_Coordinate_System
+ Transform_To_Wind_Coordinate_System ( )
Trajectory_Simulation_Phase
+ Termination_Model
+ Solver
+ Phase_Termination_Record
+ Phase_State_Derivatives
+ Phase_State_Array
+ Phase_State
+ Gravity_Model
+ Earth_Model
+ Dynamics_Model
+ Atmosphere_Model
+ Aerodynamics_Model
+ Update_Phase_State_and_Derivatives ( )
+ Initialize_Phase ( )
+ Compute_Phase_Trajectory ( )
Trajectory_Simulation
+ Trajectory_Simulation_Phase
+ Munition
+ Initialize_Simulation ( )
+ «» Compute_Trajectory ([out] Trajectory)

Fig. 11 - Resulting UML class diagram

After the definitions of these rules and mappings, we performed the transformation and the UML class diagram represented as an XMI file is produced, which is successfully imported by Rational Rose XDE [19] and ArgoUML [20]. Fig. 11 is a sample from the resulting UML class diagram. Only the important UML structures are shown in this sample, including the examples above.

We have developed an object-oriented framework based on the resulting class diagram, and constructed several trajectory simulation applications by framework completion. An account of knowledge, design and code reuse thus achieved appears in [16].

5. RELATED WORK

An early effort on transformation of ontologies into UML class diagrams is DUET [21]. It provides a UML based environment for visualizing, developing, and manipulating DAML-OIL ontologies. DUET is a plug-in for Rational Rose 2000 and ArgoUML, which can import DAML-OIL ontologies as UML class diagrams into the UML tool, and export a class diagram as an ontology. The most obvious problem with the approach of DUET is that, it does not provide any flexibility to guide the transformation process. In other words, the rules of transformation are hard-coded and cannot be easily modified depending on the user needs. Moreover, it has a tool dependency since it runs only with Rational Rose 2000 and ArgoUML.

With the latest releases of IODT (since release 1.1.2), EODM includes a transformation feature from OWL models to EODM Ecore models [22]. This enables the interoperability between OWL models and other EMF supported models. This is because the EODM Ecore model can be used as an intermediate model to support transformation between OWL and other modeling languages. Once the OWL is represented as an EODM Ecore model, it can be transformed to a UML Ecore model through MTF [23] transformations. MTF is a rule based transformation framework developed by IBM, which provides a language to define mappings between models conforming to the EMF specifications. However, because of the differences in the expressiveness of OWL and Ecore, this approach does not produce the expected results. Some information is lost during this hard-coded, non-configurable transformation from OWL models to EODM Ecore model, such as the restriction definitions or inter-class relationships. The information represented in the resulting EODM Ecore model is very limited, considering the full set of OWL constructs. Moreover, even after the production of EODM Ecore model, design of MTF rules to generate the UML Ecore model requires in-depth know-how about MTF and Ecore structures.

6. CONCLUSIONS AND FUTURE WORK

The specific contribution our work is a tested tool for ontology based simulation design by introducing a user-guided transformation process to bridge the gap between the domain modeling and software modeling realms with minimum loss of information and maximum simplicity. By mapping the OWL and UML constructs on a user interface, the knowledge in an ontology is automatically transformed into a UML class diagram. This class diagram can be understood by a developer, without any need for OWL knowledge.

In addition to the configurable construction of mappings, one other distinguishing feature in our approach is the handling of domain specific semantics captured in ontologies. Treating the transformation from OWL to UML as an ordinary XML transformation causes loss of information. Especially the semantics of restrictions and enumerations need special handling and interpretation. Our tool makes it available to use these constructs as the sources of the mappings. It also allows the user to activate some inferences on the ontologies. For example, the subclasses of an OWL class (as well as sub-properties of an OWL property) can be retrieved in a recursive way, whereas it is also possible to process only the direct subclasses of an OWL class.

We strived for simplicity with respect to tool's features. Consider, for example, designation of the visibility (public/protected/private) of attributes. It could be offered as a part of the user's options in specifying mappings. However, this could complicate the user interface and consistency checking slightly. We have followed a more straightforward approach, which is to set the visibility to "public" as a default for classes and attributes, and to let the user set the visibility on the UML class diagram whenever necessary.

In this research we utilized ontology based model driven software engineering practices for simulation development. This paper presents both an example of an ontology based simulation development and the application of state of the art software engineering practices in simulation domain.

We believe it is worthwhile to try the user-guided approach in other transformation problems as well. In cases where the source and target metamodels are well-structured around comparable basic concepts, and the uses of the target models are clearly defined there seems to be a potential to achieve the efficacy of transformations at the expense of negligible effort in constructing them.

Intelligent agents that use domain ontologies like TSONT to reason over composability of services for M&S can be envisioned (see [24]). Ontology driven service oriented trajectory simulation using TSONT is a future work.

7. REFERENCES

[1] OMG, Object Management Group, http://www.omg.org

[2] MDA, OMG Model Driven Architecture, http://www.omg.org/mda

[3] R.Prieto-Diaz, 1990, “Domain Analysis: An Introduction”, ACM SIGSOFT Software Engineering Notes, ACM Press.

[4] W. Hesse, 2005, “Ontologies in the Software Engineering Process”, EAI 2005 - Proceedings of the Workshop on Enterprise Application Integration, Berlin.

[5] R.A. Falbo, G. Guizzardi, and K.C. Duarte, 2002, “An Ontological Approach to Domain Engineering”, International Conference on Software Engineering and Knowledge Enginnering, Ischia, Italy.

[6] J.A. Miller and P.A. Fishwick, 2004, “Investigating Ontologies for Simulation Modeling”, Proceedings of the 37th Annual Simulation Symposium (ANSS’04), Arlington, VA, USA.

[7] OWL Web Ontology Language Overview, http://www.w3.org/TR/2004/REC-owl-features-20040210

[8] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, 2004, “A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools”, The University of Manchester, Stanford University.

[9] U. Durak, H. Oguztuzun, and K. İder, 2006, “An Ontology for Trajectory Simulation”, Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, USA.

[10] MOF, Meta-Object Facility, http://www.omg.org/mda/specs.htm#MOF

[11] UML, Unified Modeling Language Infrastructure, v2.1.1, http://www.omg.org/cgi-bin/doc?formal/07-02-06

[12] EMF, Eclipse Modeling Framework Project, http://www.eclipse.org/modeling/emf/?project=emf

[13] IODT, Integrated Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk

[14] EODM, EMF Ontology Definition Metamodel, http://www.eclipse.org/modeling/mdt/?project=eodm#eodm

[15] XMI, XML Metadata Interchange, http://www.omg.org/technology/documents/formal/xmi.htm

[16] U. Durak, H. Oguztuzun, and K. İder, 2008, “An Ontology Based Trajectory Simulation Framework”, Journal of Computing and Information Science in Engineering, Vol 8.

[17] D.K. Pace, 2000, “Ideas About Simulation Conceptual Model Development”, Johns Hopkins APL Technical Digest, vol. 21, no. 3.

[18] R.S. Pressman, 2009, *Software Engineering: A Practitioner's Approach*, 7th Edition, McGraw-Hill.

[19] Rational Rose XDE Developer for Java, http://www-306.ibm.com/software/awdtools/developer/java/

[20] ArgoUML, http://argouml.tigris.org/

[21] D. Gasevic, D. Djuric, and V. Devedzic, 2006, “Model Driven Architecture and Ontology Development”, Springer, pp. 164-165.

[22] Y. Pan, G. Xie, L. Ma, Y. Yang, and Z. Qui, 2005, “An MDA-Based System for Ontology Engineering”, IBM research report: IBM TR RC23795.

[23] MTF, Model Transformation Framework, http://www.alphaworks.ibm.com/tech/mtf

[24] A. Tolk, 2006, “What comes after the Semantic Web, PADS Implications for the Dynamic Web”, Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006), Singapore, May 2006, pp. 55-62.