A Knowledge Engineering Approach to UML Modeling

Bingyang Wei
Department of Computer Science
Texas Christian University
Fort Worth, Texas 76129
b.wei@tcu.edu

Jing Sun
Department of Computer Science
The University of Auckland
Auckland 1142, New Zealand
j.sun@cs.auckland.ac.nz

Yi Wang
Department of Electrical
and Computer Engineering
Manhattan College
Bronx, NY, 10471
yi.wang@manhattan.edu

***Abstract*—Multiple-viewed requirements modeling allows requirement engineers to acquire the requirements of a system from different perspectives. Requirements are then specified in various UML models. Maintaining the requirements knowledge encoded in UML notations is tedious and error-prone, since most UML CASE tools provide poor support for reasoning and query. Ontology is a formal notation for describing concepts and their relations in a domain. Since requirement is a kind of knowledge, we propose to use knowledge engineering approach for managing the consistency and completeness of UML models. In this paper, an ontology for UML diagrams is coded in a semantic web language, OWL (Web Ontology Language). The transformation of UML Class Diagram, Sequence Diagram and State Diagram to OWL knowledge base is presented. In the end, a semantic query language, SPARQL, is used to query the knowledge base. We demonstrate the feasibility of this approach through an example software system.**

I. Introduction

Requirements engineers usually views the system under development from different perspectives. The structure, behavior and interaction aspects of the system are among the most commonly considered perspectives. UML (Unified Modeling Language) makes this multiple-viewed modeling possible by providing different types of models, e.g., class, state machine and interaction models. As a result, each UML model holds partial requirements from a particular view in the form of UML diagrams, and all the models together constitute the overall description of the system. Since software requirement about a domain is indeed a kind of knowledge, the Requirements Engineering process is in fact a kind of Knowledge Engineering process. An important concern of requirements engineers during multiple-viewed modeling is the management of requirements knowledge: consistency among different models, model query, acquisition of enough useful requirements to make a model more complete. However, it is difficult for a requirements engineer to know whether a model is consistent or complete and what requirements are missing in the current model [4][5].

The application of ontology in Requirements Engineering to support elicitation, analysis, specification, validation and management of requirements is one of the solutions to improve the quality of requirements [2]. To be more specific, knowledge representations based on formal logic is considered as an effective way to represent and manage requirements knowledge [3] [7] [10]. In this paper, we are combining the popularity of the semi-formal UML notations and the Description Logic based OWL (Web Ontology Language), so that UML diagrams' semantics can be represented in OWL. Our main objective is to add a semantic layer on top of different types of UML diagrams. Different UML diagrams, although contain different kinds of requirements, can be queried and reasoned together for different purposes.

The remainder of the paper is organized as follows. In Section II, UML metaclasses are organized in a taxonomy and encoded in OWL. Section III to V present the way we transform an example system's UML models to OWL individuals and relations based on the UML Ontology. With the UML diagrams represented in OWL, query and analysis are conducted in Section VI. Section VII discusses the related work and current limitations, and Section VIII concludes the paper.

II. The UML Ontology

The Object Management Group published the latest UML Specification, *OMG® Unified Modeling Language® version 2.5.1* [6] in December, 2017. We extracted the taxonomy of UML model elements according to *Chapter 7 Common Structure*, *Chapter 8 Values*, *Chapter 9 Classification*, *Chapter 10 Simple Classifiers*, *Chapter 11 Structured Classifiers*, *Chapter 13 Common Behavior*, *Chapter 14 StateMachines*, *Chapter 15 Activities*, *Chapter 16 Actions* and *Chapter 17 Interactions*. 127 metaclasses and their inheritance relationships are organized in the class hierarchy in Protégé [8]. The complete ontology in RDF/XML syntax is available at github.com/Washingtonwei/uml-ontology.

III. UML Class Diagram in OWL

In order to demonstrate the idea of transforming UML models to OWL notations, we choose an example software system, the University Information System (UnivSys). In this example, UML models for a university seminar enrollment service system are created: Seminars can be created, scheduled, opened, enrolled, and closed; students can enroll in or

DOI reference number: 10.18293/SEKE2018-114

drop a seminar if a certain requirement is met. All three UML diagrams come from Ambler's book [1] with modifications.

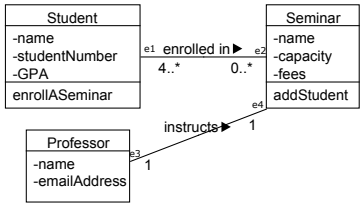

Fig. 1. Class diagram of UnivSys.

A UML model consists of a number of model elements, each of which may be used to make statements about different kinds of individual things within the system being modeled [6]. In Protégé, individuals are created and linked together to represent the semantics of a UML model. Since all the metaclasses of model elements belonging to UML class model are considered in the UML Ontology defined in the previous section, the UnivSys class diagram in Figure 1 can be easily represented as OWL individuals of various UML metaclass types. Those OWL individuals are then connected through predefined relations to form the meaning of the class diagram. Table I and II list the OWL individuals and object properties (relations between two OWL classes) extracted from the UnivSys class diagram. Here is a brief summary: each **`Class`** individual owns **`Property`** individuals as attributes and **`Operation`** individuals as operations. An **`Association`** individual owns two **`Property`** individuals as member ends. Each of them has a **`Type`** individual.

TABLE I
INDIVIDUALS CREATED FOR UML CLASS DIAGRAM

Individual(s)	UML Metaclass
StudentClass, SeminarClass, ProfessorClass	**`Class`**
studentName, studentNumber, GPA, seminarName, capacity, fees, professorName, emailAddress, associationEnd1, ..., associationEnd4	**`Property`**
enrolledInAssociation, instructsAssociation	**`Association`**
enrollASeminarOperation, addStudent	**`Operation`**

TABLE II
RELATIONS FOR UML CLASS DIAGRAM

Relation	Domains	Ranges
ownedAttribute	**`Class`**	**`Property`**
ownedOperation	**`Class`**	**`Operation`**
memberEnd	**`Association`**	**`Property`**
type	**`TypedElement`**	**`Type`**

IV. UML SEQUENCE DIAGRAM IN OWL

In this section, we focus on transforming UML Sequence Diagram (Figure 2) to OWL.

OWL individuals identified in the UnivSys sequence diagram, their corresponding UML metaclasses, and predefined relations are listed in Table III and Table IV, respectively. Each **`Lifeline`** individual represents a **`ConnectableElement`** individual whose **`Type`** individual is specified by relation type. Each **`Lifeline`** individual owns a number of **`MessageOccurenceSpecification`** individuals through relation events. Those occurrences represent the send or receive event occurrences, which are then linked to **`Message`** individuals. A **`Message`** individual specifies the content through signature and argument. The signature can be either **`Operation`** or **`Signal`**. Both of them may include **`ValueSpecification`** individuals as arguments. In this particular sequence diagram, each **`ValueSpecification`** individual is related to a **`InstanceSpecification`**, for example, theStudentInstance individual and seminarInstance individual. Besides the relations between OWL individuals, two data properties for **`Message`** individuals are considered: messageKind ("complete" , "found", "lost", "unknown") and messageSort("asynchCall", "asynchSignal", "createMessage", "deleteMessage", "reply", "synchCall").

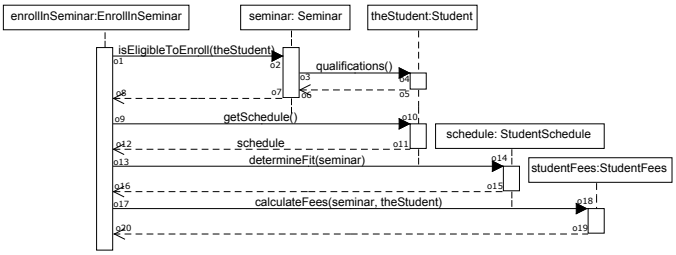

Fig. 2. A snippet of sequence diagram of UnivSys.

TABLE III
INDIVIDUALS CREATED FOR UML SEQUENCE DIAGRAM

Individual(s)	UML Metaclass
enrollInSeminarLifeline, seminarLifeline, theStudentLifeline, scheduleLifeline, studentFeesLifeline	**`Lifeline`**
connectableElement1, connectableElement2, connectableElement3, connectableElement4, connectableElement5	**`ConnectableElement`**
EnrollInSeminar, Seminar, Student, StudentSchedule, StudentFees	**`Type`**
o1, o2, o3, o4, ..., o19, o20	**`MessageOccurrenceSpecification`**
isEligibleToEnrollMessage, qualificationsMessage, qualificationsReply, isEligibleToEnrollReply, getScheduleMessage, getScheduleReply,determineFitMessage, determineFitReply, calculateFeesMessage, calculateFeesReply	**`Message`**
theStudentInstanceValue, seminarInstanceValue	**`InstanceValue`**
isEligibleToEnrollOperation, qualificationOperation, getScheduleOperation, determineFitOperation, calculateFeesOperation	**`Operation`**
theStudentInstance, seminarInstance	**`InstanceSpecification`**

TABLE IV
RELATIONS FOR UML SEQUENCE DIAGRAM

Relation	Domains	Ranges
events	**Lifeline**	**Occurrence-Specification**
represents	**Lifeline**	**ConnectableElement**
signature	**Message**	**NamedElement**
argument	**Message**	**ValueSpecification**
sendEvent	**Message**	**MessageEnd**
receiveEvent	**Message**	**MessageEnd**
type	**TypedElement**	**Type**
instance	**InstanceValue**	**Instance-Specification**
classifier	**Instance-Specification**	**Classifier**

V. UML STATE DIAGRAM IN OWL

Figure 3 illustrates the behavior of a seminar during its lifetime. Based on the UML Specification, the semantics of the state diagram is represented in Protégé in the form of individuals (Table V) and their relations (Table VI). **Transaction** plays an important role in capturing the meaning of a state machine diagram. Each **Transaction** individual connects to two **State** individuals by sourceVertex and targetVertex. A transaction owns a **Trigger** individual that is related to an **Event** individual, a **Constraint** individual as guard and a **Behavior** individual as effect. Every constraint is related to a **ValueSpecification** individual by constrainedElement relation, and has a specification that evaluates to a Boolean value. Each **State** individual may own behaviors as its entry, exit and doActivity. Besides the relations between OWL individuals, one data property for **PseudoState** instances is considered: pseudostateKind ("choice", "deepHistory", "entryPoint", "exitPoint", "fork", "initial", "join", "junction", "shallowHistory", "terminate"); another data property for **Transition** instances is considered: transitionKind("external", "internal", "local").

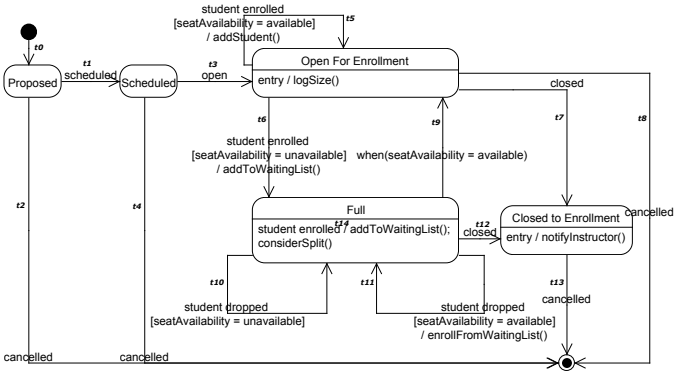

Fig. 3. State diagram of a seminar.

VI. QUERYING UML MODELS USING SPARQL

Section III to V convert University Information System's three UML models to OWL individuals. This process results in 127 distinct OWL individuals. Interested readers can find those individuals at github.com/Washingtonwei/uml-ontology. With the three UML diagrams properly represented in OWL, query can be conducted using SPARQL. SPARQL (SPARQL Protocol and RDF Query Language) is a semantic query language. It is an effective way to retrieve and manipulate data stored in RDF format. The underlying structure of an OWL knowledge base is a collection of triples, each consisting of a subject, a predicate and an object. For example, the description of "a class A owns a property B as an attribute" is stored in OWL as a triple: *A ownedAttribute B*. In a SPARQL query, there are several important parts: a SPARQL variable starts with a question mark and can match any individual in the OWL knowledge base. In the WHERE clause, triple patterns

TABLE V
INDIVIDUALS CREATED FOR UML STATE DIAGRAM

Individual(s)	UML Metaclass
InitialState, ProposedState, ScheduledState, OpenForEnrollmentState, FullState, ClosedToEnrollmentState, FinalState	**State**
transition0, transition1, transition2, ..., transition13, transition14	**Transition**
completionTrigger, scheduledTrigger, openTrigger, studentEnrolledTrigger, studentDroppedTrigger, closedTrigger, cancelledTrigger, whenSeatAvailableTrigger	**Trigger**
completionEvent, scheduledEvent, openEvent, studentEnrolledEvent, studentDroppedEvent, closedEvent, cancelledEvent, whenSeatAvailableEvent	**CallEvent**
seatAvailableConstraint, seatUnavailableConstraint	**Constraint**
seatAvailableExpression, seatUnavailableExpression	**OpaqueExpression**
addStudent, logSize, addToWaitingList, notifyInstructor, enrollFromWaitingList, considerSplit	**OpaqueBehavior**
addStudentOperation, logSizeOperation, addToWaitingListOperation, notifyInstructorOperation, enrollFromWaitingListOperation, considerSplitOperation	**Operation**
seatAvailability	**Element**

TABLE VI
RELATIONS FOR UML STATE DIAGRAM

Relation	Domains	Ranges
sourceVertex	**Transition**	**Vertex**
targetVertex	**Transition**	**Vertex**
trigger	**Transition**	**Trigger**
guard	**Transition**	**Constraint**
effect	**Transition**	**Behavior**
entry	**State**	**Behavior**
exit	**State**	**Behavior**
doActivity	**State**	**Behavior**
event	**Trigger**	**Event**
changeExpression	**ChangeEvent**	**ValueSpecification**
specification	**Constraint**	**ValueSpecification**
constrainedElement	**Constraint**	**ValueSpecification**
method	**BehavioralFeature**	**Behavior**

are defined in which any parts can be replaced with a SPARQL variable. The SELECT result clause returns a table of variables and values that satisfy the query. Protégé has built-in support for SPARQL. One sample query is presented here.

- "***What are the associated classes of SeminarClass?***"

```
PREFIX
:<http://www.semanticweb.org/uml-ontology#>
SELECT ?associatedClass
  WHERE {
    ?end1 :type :SeminarClass .
    ?association :memberEnd ?end1 .
    ?association :memberEnd ?end2 .
    ?end2 :type ?associatedClass .
    FILTER(?associatedClass!=:SeminarClass)
}
```

Explanation: Select any individual such that it is the type of an end (:end2), which is a member end (:memberEnd) of an association (?association), whose the other end has type SeminarClass (:SeminarClass). The FILTER makes sure that we are looking for a class other than SeminarClass.

Answers from Protégé:

- http://www.semanticweb.org/uml-ontology#StudentClass
- http://www.semanticweb.org/uml-ontology#ProfessorClass

VII. Related Work and Discussion

Since UML Class Diagram can also be used to represent ontology of a domain, most work related to UML and OWL emphasizes the transformation between UML Class Diagram and OWL ontology [3][14]. Furthermore, OMG's ODM(Ontology Definition Metamodel) includes UML profiles for RDF and OWL, which provide a standard graphical notation for RDF vocabulary and OWL ontology development using UML tools. Our work is inspired by the work of Van Der Straeten [9] and Wei [12][13][11]. Van Der Straeten and colleagues used Description Logic to check the inconsistency between different versions of UML models. Wei and colleagues specified overlaps of heterogeneous UML models in Conceptual Graphs and used the overlap for identifying missing requirements. However, few work aims at providing a comprehensive and complete UML Ontology that covers all UML Specification diagrams. Furthermore, our work takes advantage of SPARQL to query and check consistency of different UML models.

One important contribution of this work is the construction of a comprehensive UML ontology in the RDF/XML syntax which can be queried and reasoned in Protégé and other Semantic Web tools like Apache Jena. By organizing the latest UML Specification metaclasses in OWL, we invite the community to contribute to its further refinement. For research purposes, we restrict ourselves to a subset of the UML metamodel. The current UML ontology includes 127 most commonly used UML metaclasses and 25 metarelations, which can support defining the semantics of UML Class, Sequence and State Diagrams in OWL notation. Future work will include all the metaclasses and metarelations defined in the UML Specification, so that more UML diagrams can be converted to OWL for analysis. Another current limitation is the manual conversion from a given UML diagram to OWL. Future work will focus on automation of transforming UML models to OWL and direct invocation of the OWL API from within a UML CASE tool. Another future work would be designing a user friendly querying interface based on natural language query, so that users don't need to master SPARQL language.

VIII. Conclusion

In this paper, an approach to manage UML model knowledge is proposed and validated through a simple illustrative example software system. The formalism used is W3C standardized OWL, which is a Description Logic based formalism. Three types of UML diagrams have been converted to OWL knowledge base. Queries and reasoning can be conducted towards the resulting knowledge base. Future work will include more OWL reasoning in terms of consistency, knowledge acquisition and inference on the UML knowledge base. The results can help requirements engineers better understand their models and provide possible inconsistencies suggestions.

References

[1] S. W. Ambler. *The object primer: Agile model-driven development with UML 2.0*. Cambridge University Press, 2004.

[2] D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva. Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering*, 21(4):405–437, 2016.

[3] D. Djurić, D. Gašević, and V. Devedžić. Ontology modeling and mda. *Journal of Object technology*, 4(1):109–128.

[4] D. Firesmith. Are your requirements complete? *Journal of Object Technology*, 4(1):27–44, 2005.

[5] F. J. Lucas, F. Molina, and A. Toval. A systematic review of uml model consistency management. *Information and Software Technology*, 51(12):1631–1645, 2009.

[6] O. M. G. (OMG). Unified modeling language, version 2.5.1. OMG Document Number formal/17-12-05 (https://www.omg.org/spec/UML/2.5.1), 2017.

[7] L.-j. SHAN and H. ZHU. A formal descriptive semantics of uml. *Computer Engineering & Science*, 3:026, 2010.

[8] S. U. Stanford Center for Biomedical Informatics Research (BMIR). Protégé, a free, open-source ontology editor and framework for building intelligent systems. https://protege.stanford.edu/, 2018.

[9] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using description logic to maintain consistency between uml models. In *«UML» 2003-The Unified Modeling Language. Modeling Languages and Applications*, pages 326–340. Springer, 2003.

[10] B. Wei. *A comparison of two frameworks for multiple-viewed software requirements acquisition*. PhD thesis, Ph. D. thesis, University of Alabama in Huntsville, 2015.

[11] B. Wei and H. S. Delugach. A framework for requirements knowledge acquisition using uml and conceptual graphs. In *Software Engineering Research, Management and Applications*, pages 49–63. Springer, 2016.

[12] B. Wei and H. S. Delugach. Transforming uml models to and from conceptual graphs to identify missing requirements. In *International Conference on Conceptual Structures*, pages 72–79. Springer, 2016.

[13] B. Wei, H. S. Delugach, E. Colmenares, and C. Stringfellow. A conceptual graphs framework for teaching uml model-based requirements acquisition. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 71–75. IEEE, 2016.

[14] J. Zedlitz, J. Jörke, and N. Luttenberger. From uml to owl 2. In *Knowledge Technology*, pages 154–163. Springer, 2012.