Mehran University Research Journal of Engineering & Technology
Vol. 37, No. 4, 521-534 October 2018
p-ISSN: 0254-7821, e-ISSN: 2413-7219
DOI: 10.22581/muet1982.1804.07

Ontology-Based Verification of UML Class/OCL Model

ABDUL HAFEEZ*, SYED HYDER ABBAS MUSAVI**, AND AQEEL UR REHMAN*

RECEIVED ON 09.05.2017 ACCEPTED ON 13.11.2017

ABSTRACT

Software models describe structures, relationships and features of the software. Modern software development methodologies such as MDE (Model Driven Engineering) use models as core elements. In MDE, the code is automatically generated from the model and model errors can implicitly shift into the code, which are difficult to find and fix. Model verification is a promising solution to this problem. However, coverage of all facets of model verification is a painful job and existing formal/semi-formal verification methods are greatly inspired by mathematics and difficult to understand by the software practitioners.

This work considers particularly UML Class/OCL (Unified Modeling Language Class/Object Constraint Language) model and presents an ontology-based verification method. In the proposed method, a class diagram is transformed into ontology specified in OWL (Web Ontology Language) and constraints into SPARQL NAF (Negation as Failure) queries. This work tries to demonstrate that the proposed approach can efficiently cover all aspects of UML Class/OCL model verification.

Key Words: Software Verification, Model Verification, Unified Modeling Language Class/ Object Constraint Language Model.

1. INTRODUCTION

Our daily life is extremely dependent upon software. They are everywhere, for example, in a smartphone, high-end television sets, and even they drive the vehicle. However, the history shows the failure of the software cause lives and economic losses [1] and correctness of the software is a key issue. Testing of the software before implementation is very important. Although, testing has two major limitations: (1) the testing only checks the absence of errors (2) and testing is performed in later phases of software development. The cost of errors correction in later phases is higher than earlier phases [2]. On the other side, modern software are becoming more and more complex and large. They require a lot of human efforts and time, and software development companies want to release software as early as possible [3]. Therefore, new software development methodologies have been introduced for accelerating the software development. MDE is one of them in which software models are considered as a nucleus of software development.

Authors E-Mail: (abdul.hafeez@hamdard.edu, dean@indus.edu.pk, aqeel.rehman@hamdard.edu)
* Department of Computer Science, Hamdard University, Karachi.
** Faculty of Engineering Science & Technology, Indus University, Karachi.

UML is a graphical modeling language and it is commonly used in MDE [4]. It is used in software specification, analysis, design, documentation, and even for code generation [5]. UML offers a number of models for dealing with various aspects of software [6-7]. The class model is an important part of UML and describes the system through concepts, relationships, and constraints [8]. OCL is combined with a class model for specifying the integrity constraints and business rules. However, MDE approach is not also free from error risks. In MDE, models are created in the initial stages of software development and in the initial stages, software development team does not fully aware of the business domain and their constraints. Therefore, models can develop with errors, and these errors can implicitly shift into the code [9]. A promising solution to this problem is model verification.

Model verification is also a solution to the problems which are faced by testing such as model verification checks the correctness of model and makes sure that the model is bug-free. Model is created during the early phases of software development, therefore, error checking is economical in the early phases [10]. Current UML Class/OCL models verification methods are sound and provide great efforts to check the correctness. However, they are based on formal/semi-formal methods, therefore, their notation extremely inspired by mathematics [11]. They are entirely different from the UML class model and difficult to understand by the software practitioners. They also have some limitations such as support of various basic data types (string and date), graphical constraints (xor and dependency relationships) and support of logical consequences. On the other side, ontology and UML class model have many similar elements and both are used for modeling real-world concepts [12].

This work presents an ontology-based verification method, particularly for UML Class/OCL model. Currently, the proposed method supports OCL invariants and does not support OCL operations. However, in this work, ontology as the target notation is motivated by the fact that the current ontology reasoners support reasoning over thousands of ontological items within a reasonable time [13], and all the verification facets mentioned in existing benchmarks can be easily achieved through the ontology-based method. In the proposed approach class diagram is transformed into ontology specified in OWL-DL and OCL constraints into SPARQL NAF queries.

The rest of the paper is organized as follows. Section 2 discusses the background and related work. Section 3 describes the proposed solution. Section 4 presents an example of ontology-based verification and results. Finally, section 5 and 6 present our conclusions and points out future works.

2. BACKGROUND AND RELATED WORK

2.1. Ontology

Ontology is the concept of metaphysics, which is using by the philosopher from mid of sixteen century for categorization and representation of real-world entities. Ontology has many elements (e.g. classes, relations, and individuals) similar to the UML class model elements. Currently, software engineering professional are also integrating ontology in software development practices (processes, methods, tools, etc.). Many researchers [14-17] have been used ontology for representation and verification of various software artifacts.

Mahmud [14] proposed domain specific language called ReSA for an embedded system. The ReSA utilizes axioms of ontology for specification of the embedded system. They perform scalable formal verification of various Simulink models. Nguyen et. al. [15] presented an ontology-based integrated framework for verification of goal-oriented and use case modeling techniques. They developed a tool called GUITAR, it takes textual requirements and transforms into the structured specification for automatic reasoning. Corea et. al. [16] presented an ontology-based approach for verification of business processes. They specified business rules as a logic program and used ontology reasoner for discovering model elements which violate the rules.

Liao et. al. [17] presented ontology-based notification-oriented data intensive EIS (Enterprise Information System). A notification-oriented paradigm is a new approach for software and hardware specification. They also pointed out the challenges which faced by the legacy EIS in the fourth industrial revolution and presented ontology-based potential solutions.

The different researcher also used ontology for transformation and verification of various UML models. He et. al. [18] verified UML behavior model through ontology. In this approach, UML behavior model is divided into the static and dynamic elements. The static elements are transformed into the OWL-DL and dynamics elements are transformed into the DL-safe rules and then they are verified by the reasoner. Dilo et. al. [19] presented a comparison between UML and web ontology language and identified that both have many common elements e.g. classes, relationships, attributes. They also identified the differences of both languages e.g. UML class model has many relationships (association, generalization, composition) and OWL only has an object property. At last, they concluded that both are compatible with each other.

Bahaj et. al. [20] presented an alternative translation method of UML class model into the ontology and categorized aggregation and composition as a special type of association. Belghiat et. al. [21] proposed the graph-based transformation of class diagram meta-model into the ontology. Parreiras et. al. [22] combined UML and ontology for representing software models and incorporated MOF (Meta Object Facility) meta-model as the backbone for UML and ontology.

2.2 UML Class/OCL Model Verification

Verification of UML Class/OCL model through formal/ semi-formal notation discussed in many works. UML only provides graphical elements for representing software components without any formal foundation [23]. In UML well-formedness rules are defined by meta-model and OCL without any proof. Hence, the majority of early works only formalized UML meta-model and well-formedness rules by different formal methods (such as Z notation, B method).

France et. al. [23] used Z notation for the formalization of UML core meta-model and they translated the UML meta-model into the compositional schema. The schema contains many sub-schemas which correspond to every component of core UML meta-model. Different formal methods have different strength in different areas and a single method cannot cover all aspects of UML model verification and validation. In this regard, Kim et. al. [24] proposed an integrated verification and validation framework in which suitable formalism can be selected by the designer according to the need.

Truong et. al. [25] presented the transformation of UML class model into the B method and verified consistency of a class model against UML well-formedness rules. In this method, the UML well-formedness rules are transformed into the invariant of B abstract machine. Some works also used semi-formal methods for formalization and verification of UML Class/OCL model e.g. CSP (Constraint Satisfaction Problem) and Alloy. Cabot et. al. [26] presented incremental verification of UML Class/OCL model through CSP. They argued that verification of constraints after every structure event (Insert Entity, Update Attribute, Delete Entity, etc.) can be very costly and inefficient. They introduced a term PSEs (Potential Structure Events). The PSEs are events which can cause of constraints violation. In this technique, PSEs for every integrity constraint are recorded and instances of entities and relationships are incrementally verified according to the PSEs. They presented fully automatic, decidable solution for bounded verification of UML Class/OCL model.

Moaz et. al. [27] transformed advanced features of UML class model (multiple inheritance, interface) into alloy specification and performed various analyses such as the intersection of two or more classes and refinement analysis.

Shaikh et. al. [28] reduced the complexity of UML Class/OCL model verification through model slicing. In this approach, the model is divided into many sub-models and unnecessary elements are removed from the sub-models (slices). They reported the model slicing technique reduces the verification time of a large model with few constraints. However, if the model has many disjoint sub-models, then fewer partitions will be made and efficiency will not be gained.

Moreover, in the area of UML Class/OCL model verification, Gogolla et. al. [29] presented comprehensive guidelines for future UML Class/OCL model verification methods. These guidelines more or less cover all aspects of UML Class/OCL model verification and may be considered as functional requirements for new verification methods. These requirements are partly overlapping, therefore, the core requirements summarized in Table 1. The next section briefly describes these requirements for further detail see [29].

2.3 Requirements for UML Class/OCL Model Verification Method

Requirement-1: The consistency verification ensures that a non-empty model should be created without violation of any constraint. Constraints can be local or global and simple or complex. The local constraint is applied on a single class and the global constraint is applied on many classes. The simple constraint performs easy computation and complex constraint performs the enormous computation. The verification method should support the local/global and simple/complex constraints.

TABLE 1. SUMMARY OF REQUIREMENTS [29]

ID	Requirements	Description
Req-1	Consistency Verification	Local/Global Constraints, Simple/Complex Constraints
Req-2	Intensive Arithmetic Computation	Support of integer, and real number operations and functions
Req-3	Intensive String Processing	Support of String values and string function
Req-4	Consequences	Infer new information
Req-5	Large no of instances	10 to 30 instances of each class

Requirement-2: Constraints can perform arithmetic computation on both integer and real numbers. Verification method should support the intensive arithmetic computation on integer and real numbers.

Requirement-3: Constraints can also perform string computation and can use string functions. Verification method should support the string processing.

Requirement-4: Verification method should be able to infer consequences (new facts) from a set of asserted facts or axioms.

Requirement-5: Verification method should support a large number of instances because sometimes verification of minimum cardinality cannot explore the complete features of the model. Therefore, at least 10-30 instances of each class should be supported by a verification method.

3. PROPOSED SOLUTION

Ontology and UML Class/OCL model both are used for representing real-life entities and both have many common elements e.g. classes, properties, instances, and generalization. However, ontology has an advantage over UML Class/OCL model. It has a proper formal foundation. The main difference between ontology and Class/OCL model is: the ontology works on OWA (Open World Assumption) and Class/OCL model works on CWA (Close Work Assumption). In OWA, unknown assumptions are considered true, and in CWA unknown assumptions are considered false. For closing the world this work represents constraints into the SPARQL NAF queries. SPARQL is not only a query language. It also provides other constructs for performing different functionality e.g. ASK and CONSTRUCT. ASK can be used for verification consistency of constraints and CONSTRUCT can be used for inferring new assertion from the existing one. Fig. 1 shows the verification steps of the proposed method. Initially, the class diagram is transformed into the ontology (specified by OWL-DL) and OCL constraints are transformed into the SPARQL ASK. After that, the correctness of the model against the constraints is verified and finally, feedback is returned to the user. The rest of the section presents the translation of UML Class/OCL model into the ontology and how the proposed method realizes all the requirements mentioned in section 2.3.

3.1 Transformation of Class diagram

3.1.1 Translation of Classes

In the proposed method UML classes are transformed into ontology classes. UML supports UNA (Unique Name Assumption) in which each instance of a class is considered as a unique entity. On the other side, in ontology two different instances can be considered as a same entity. However, by the combination of other ontology constructs the semantic of UNA can be achieved. For Example, in each class, an extra datatype property (ID) is attached as a key through HasKey construct. Individuals of a class are annotated as All different and Classes are declared mutually disjoint.

Declaration (Class (Class Name))
Has Key (Class Name (key Attribute))
Functional Data Property (Key Attribute)
Inverse Functional Data Property (Key Attribute)

3.1.2 Class Attribute

tributes of the class are represented by datatype property. The domainof property represents respective ontology class and range represents an appropriate datatype.

Declaration (Data Property (Attribute Name))
Data Property Domain (Attribute Class Name)
Data Property Range (Attribute Data type)

3.1.3 Translation of Association Relationship

Association relationships between classes are transformed into the object properties. Additionally, inverse properties are added for representation of two-way association communication. Multiplicities of associations are represented by the ontology qualified cardinalities.

Declaration (Object Property (A))
Object Property Domain (A C1)
Object Property Range (A C2)
Declaration (Object Property (A^-))
Object Property Domain (A^- C2)
Object Property Range (A^- C1)
Declaration (Inverse Object Properties (A A^-))

3.2 Realization of Requirements

3.2.1 Constraints Consistency

According to the section 2.3, the first requirement is constraints consistency. In the proposed method, local and global constraints can be easily represented and verified by ASK NAF queries. Table 2 shows the representation of local constraints PaperLenght which presented in [29] through ASK NAF query. The representation of global constraint illustrated in Table 3, where the constraint involves two classes.

TABLE 2. TRANSFORMATION OF OCL CONSTRAINT

CL Constraint	ASK NAF
context Paper inv paperLength: self.wordCount < 10000	ASK Where { ?paperinstance :WordCount ?WC }

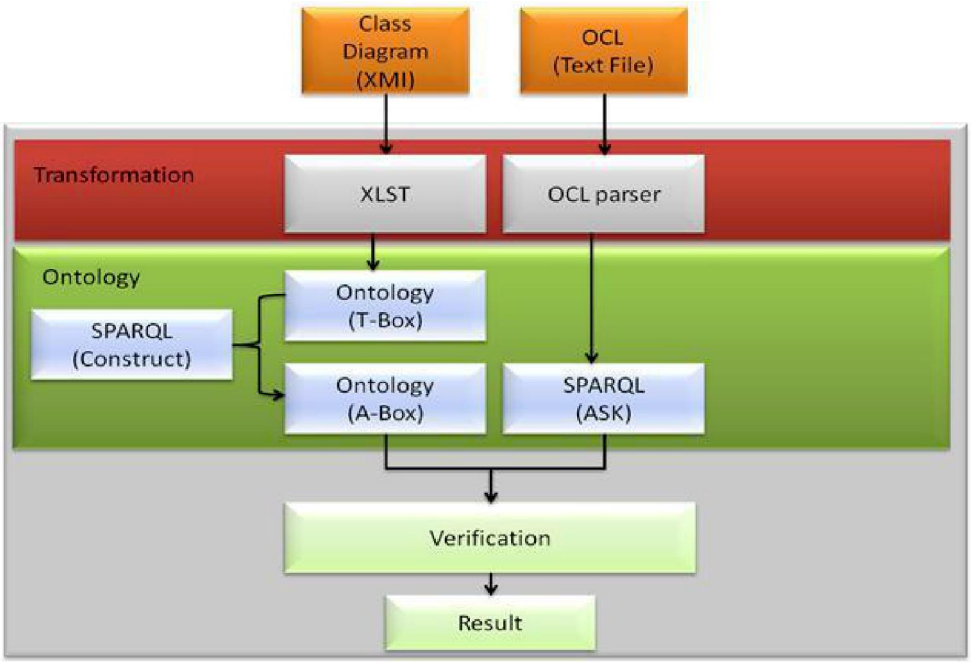


FIG. 1. VERIFICATION STEPS OF PROPOSED METHOD

3.2.2 Intensive Arithmetic Computation

The second requirement for new UML/OCL model verification is the intensive support of arithmetic computation. This requirement can be easily realized by the proposed method. Ontology supports all numeric data types such as integer, float, decimal. SPARQL supports all standard arithmetic operations (+,-,*, /, etc), and numeric functions (floor, ceil, absolute, min, max, etc). Constraints with massive arithmetic computations can be easily specified through SPARQL ASK. **Table 3** shows the example of intensive arithmetic constraint and equivalent SPARQL ASK query.

3.2.3 Intensive use of String

The most important requirement for the new UML Class/ OCL model verification method is support of string because existing methods rarely support the constraints which have string operations. However, Ontology supports string data types and SPARQL has many built-in string functions e.g. Substr, Strlen, Ucase. Table 4 shows the SPARQL ASK query for constraint mentioned in [29] which has string computation.

3.2.4 Logical Consequences

Support of logical consequences is a very vital requirement for new verification methods. Since this requirement is also not supported by most of the existing methods. In ontology, SPARQL CONSTRUCT queries are used for specifying the inference rules and generate the new dataset. Therefore, they can be used for performing logical consequences on UML Class/OCL model. **Table 5** shows the SPARQL CONSTRUCT for bigamy logical consequence which described in [29].

TABLE 5. CONSEQUENCE REPRESENTATION

Logical Consequences
CONSTRUCT { ?Iper :isbigamy "yes" } Where { ?Iper RDF:TYPE :CPerson ?Iper :Married ?IPer2 }Group by ?Iper1 Having (COUNT (?Iper2) > 1) }

TABLE 3. TRANSFORMATION OF GLOBAL CONSTRAINT WITH ARITHMETIC COMPUTATION

OCL Constraint	ASK NAF
Context Department inv NumberEmployees: self.employee->size()<= Employee.allInstances()->size()/2	ASK Where { {SELECT (COUNT (?IEMP) as ?deptEmp) Where { ?Idept :RoleEmployee ?IEmp ?IEmp RDF:TYPE :CEmployee } Group By (Idept) } {SELECT (COUNT(?IEMP)/2 AS EEMP) Where { ?IEMP RDF:TYPE :CEmployee } } Filter (!(DEmp <=?EEMP)) }

TABLE 4. OCL CONSTRAINT WITH STRING PROCESSING TRANSFORMATION

OCL Constraint	ASK NAF
inv nameCapitalThenSmallLetters: let small:Set(String)= Set{'a','b','c','d','e','f','g','h','i','j','k','l','m', 'n','o','p','q','r','s','t','u','v','w','x','y','z'} in let capital:Set(String)= Set{'A','B','C','D','E','F','G','H','I','J','K','L','M', 'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in capital->includes(name.substring(1,1)) and Set{2..name.size}->forAll(i | small->includes(name.substring(i,i)))	ASK Where { ?Iper RDF:TYPE :CPerson ?Iper :name ?na FILTER (REGEX (?na, STR(UCASE(SUBSTR(?na,1,1)))) && (REGEX(?na , STR(LCASE(SUBSTR(?na,2,STRLEN(?na)-1)))))

3.2.5 Large Number of Instances

Ontology is used for making formal models of real-world entities and ontology reasoner can perform reasoning over the large models. Modern reasoning and a rule engine can process thousands of ontological items within a reasonable time [13]. Next section demonstrates through a real-life example that the proposed solution can also achieve this requirement efficiently.

4. UML CLASS/OCL MODEL VERIFICATION EXAMPLE

This section illustrates the whole formalization and presents how the proposed method effectively verifies the UML Class/OCL model. Fig. 2 shows a UML Class model of a company. The model has three classes (Employee, Department, and Project) and two associations (Work In and Control). The multiplicity constraint of association Work In states that employee can work utmost one department and department must hold at least one employee and utmost many. The multiplicity constraint of association Control specifies that department can control minimum one project and maximum five projects and project control by utmost one department.

The company model also has OCL constraints which specified in Table 6. The Dept Budget and Pro Budget constraints state that Budget of the department and project should be greater than zero. Pro Bud Less Dept constraint specifies that the project budget should be less than their respective department. Emp H Date Greater B date constraint states that the employee Hire Date should be greater than his date of birth. The constraint Dept Bud Greater all Pro states that the budget of all departmental projects should be less than or equal to their department budget. The last constraint Emp ID Intial Letter then No specifies that the EID should be started by the letter and followed by the numbers. It is possible to infernew properties from the existing one such as Large Department which specifies if a department controls 4-5 projects then it will be considered as a large department.

The company model has all the requirements which described in section 2.3. It has intensive arithmetic computation in Dept Bud Greater all Pro constraint, intensive string processing in Emp ID Intial Letter then No constraint, and inferred consequences in Large Department. Table 7 shows the complete translation of

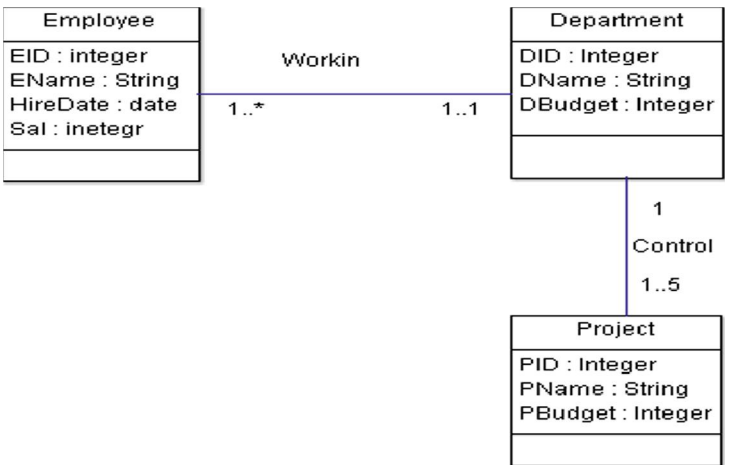


FIG. 2. UML CLASS/OCL COMPANY MODEL

the company model into the ontology and ASK NAF. For analysis of a large number of instances, 30 instances of Employee, 10 instances of Department and 20 instances of Project have been created and linked to each other as shown in Fig. 4. The valid properties setting is shown in Table 8 and invalid properties setting is shown in Tables 9.

The proposed method stated in the previous section has been implemented in a prototype tool. The tool uses Jena framework for representing class diagram into the ontology and OCL constraints into the JENA ARQ (JENA implementation of SPARQL). The tool interacts with Pellet reasoner to perform verification on generated ontology, and ARQ queries executed one by one to check the consistency of constraints. The current version of the tool does not support automatic transformation. However, a future release of the tool will support the automatic transformation. Fig. 5(a-b) illustrates the verification results of both valid and invalid company model using the prototype tool.

The proposed method more or less supports all aspects of UML Class/OCL model verification presented in existing literature. Almost all existing work use formal methods for verification of UML Class/OCL model which are extensively inspired by mathematics. On the other side, the current method is ontology-based and ontology and UML Class model have many common elements. Most of the verification methods only work on integer data types. However, the proposed method supports all data types such as number, string, date and also provides related functions for performing the advanced computation. Ontology is based on a decidable fragment of first-order logic and current ontology reasoners are powerful enough that can do reasoning over thousands of elements.

TABLE 6. OCL CONSTRAINTS OF COMPANY MODEL

Context Department Inv: DeptBudget: self.Dbudget >0
Context Project Inv: ProBudget: self.Pbudget >0
Context Project Inv: ProBudLessDept:self.PBudget <= self.department.Dbudget
Context Employee Inv: EmpHDateGreaterBdate: self.hireDate > self.DOB
Context Department Inv: DeptBudGreaterallPro: self.dBudget >= self.project (iterate(pro:Project;sum:Integer=0 | sum + pro.PBudget)
inv EmpIDIntialLetterthenNo: let No:Set(String)= Set{'1','2','3','4','5','6','7','8','9','0} in let Letter:Set(String)= Set{'A','B','C','D','E','F','G','H','I','J','K','L','M', 'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in letter->includes(EID.substring(1,1)) and Set{2..EID.size}->forAll(i | No->includes(EID.substring(i,i)))

TABLE 7. COMPLETE ONTOLOGY-BASED TRANSLATION OF COMPANY MODEL

Ontology for Company Model
1. **Class**: *Department*
2. **Class**: *Project*
3. **Class**: *Employee*
4. **ObjectProperty**: *Control***Domain**: *Department***Range**: *Control max 3 :Project, :Control min 1 :Project*
5. **ObjectProperty**: *Workin***Domain**: *Employee***Range**: *Workin exactly 1 :Department*
6. **ObjectProperty**: *IControl* **Domain**: *Project***Range**: *IControl exactly 1 :Department*
7. **ObjectProperty**: *IWorkin* **Domain**: *Department***Range**: *IWorkin some :Employee*
8. **DataProperty**: *EID* **Domain**: *Employee***Range**: *xsd:integer*
9. **DataProperty**: *EName* **Domain**: *Employee***Range**: *xsd:string*
10. **DataProperty**: *sal***Domain**: *Employee***Range**: *xsd:float*
11. **DataProperty**: *HireDate***Domain**: *Employee***Range**: *xsd:dateTime*
12. **DataProperty**: *PID***Domain**: *Project* **Range**: *xsd:integer*
13. **DataProperty**: *DID***Domain**: *Department***Range**: *xsd:integer*
14. **DataProperty**: *DName* **Domain**: *Department***Range**: *xsd:string*
15. **DataProperty**: *PBudget***Domain**: *Project* **Range**: *xsd:double*
16. **DataProperty**: *PName***Domain**: *Project***Range**: *xsd:string*
17. **DataProperty**: *DBudget***Domain**: *Department***Range**: *xsd:double*

SPARQL NAF Queries for Company Model OCLs	
Constraint: DeptBudget **ASK Where**{*?Department :DBudget ?DB* **Filter** (!(*?DB >0*) }	**Constraint: ProBudget** **ASK Where** {*?Project :PBudget ?PB* **Filter** (!(*?PB > 0*) }
Constraint: ProBudLessDept **ASK Where**{*?Department :control ?Project;* *?Project :PBudget ?pb.* *?Department :DBudget ?db* **Filter**(!(*?pb <= ?db*)) }	**Constraint: EmpHDateGreaterBdate** **ASK Where** { *?Employee :DOJ ?DOJ.* *?Employee :DOB ?DOB.* **Filter** (!(*?DOJ > ?DOB*)) }
Constraint: DeptBudGreaterallPro **ASK Where** { *?DIndividual***rdf:type***Department.* *?DIndividual :DBudget ?DBD* {**SELECT (SUM**(*?PB*) *AS ?APB*) **Where** {*? PIndividul* **rdf:type** *Project.* *?PIndividual :PBudget ?PB.* *?PIndividual :control? DIndividual.*} **group by** (*?DIndividual*)} **Filter** (!(*?APB < ?DBD*)) }	**Constraint: EmpIDIntialLetterthenNo** ASK Where {? *EIndividual***rdf:type** :*Employee* *?EIndividual :name ?na* **FILTER(REGEX** *(?na*, **STR(UCASE(SUBSTR***(?na,1,1))))***&&** (REGEX(?na,(SUBSTR(?na,2,STRLEN(?na)-1)),"\\d")}

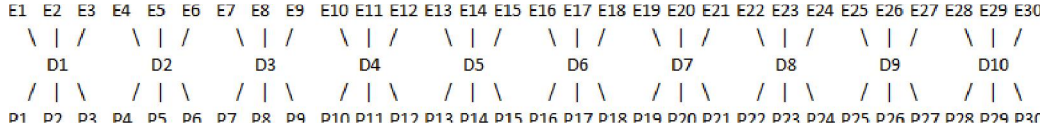


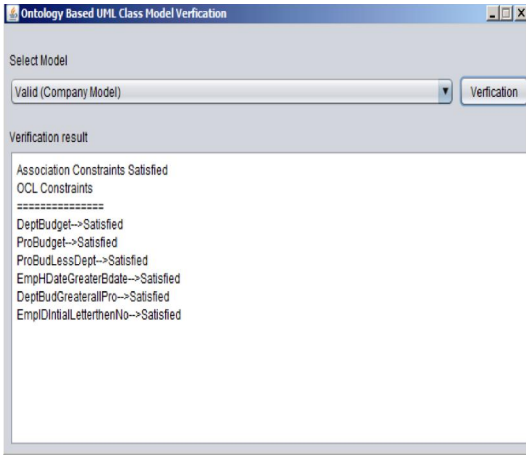
FIG. 4. COMPANY MODEL INSTANCES AND LINKS

TABLE 8. VALID PROPERTIES SETTING FOR COMPNAY MODEL

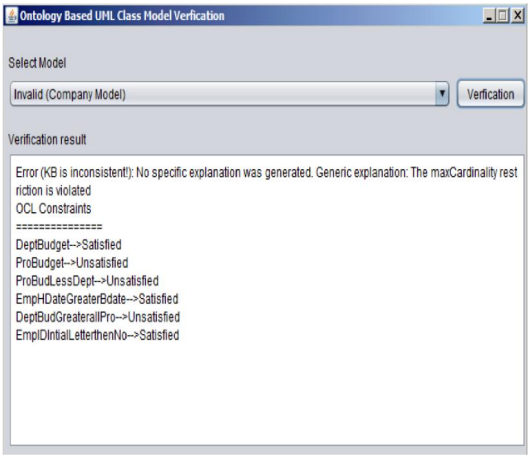
EID	Hire Date	DOB	DID	Dbudget	PID	Pbudget
E1	12.12.2005	12.12.1978	D1	100	P1	1
E2	13.12.2005	13.12.1978			P2	5
E3	14.12.2005	14.12.1978			P3	9
E4	15.12.2005	15.12.1978	D2	500	P4	13
E5	16.12.2005	16.12.1978			P5	17
E6	17.12.2005	17.12.1978			P6	21
E7	18.12.2005	18.12.1978	D3	900	P7	25
E8	19.12.2005	19.12.1978			P8	29
E9	20.12.2005	20.121978			P9	33
E10	21.12.2005	21.12.1978	D4	1300	P10	37
E11	22.12.2005	22.12.1978			P11	41
E12	23.12.2005	23.12.1978			P12	45
E13	24.12.2005	24.12.1978	D5	1700	P13	49
E14	25.12.2005	25.12.1978			P14	53
E15	26.12.2005	26.12.1978			P15	57
E16	27.12.2005	27.12.1978	D6	2100	P16	61
E17	28.12.2005	28.12.1978			P17	65
E18	29.12.2005	29.12.1978			P18	69
E19	30.12.2005	30.12.1978	D7	2500	P19	73
E20	31.12.2005	31.12.1978			P20	77
E21	01.01.2006	01.01.1979			P21	81
E22	02.01.2006	02.01.1979	D8	2900	P22	85
E23	03.01.2006	03.01.1979			P23	89
E24	04.01.2006	04.01.1979			P24	93
E25	05.01.2006	05.01.1979	D9	3300	P25	97
E26	06.01.2006	06.01.1979			P26	101
E27	07.01.2006	07.01.1979			P27	105
E28	08.01.2006	08.01.1979	D10	3700	P28	109
E29	09.01.2006	09.01.1979			P29	113
E30	10.01.2006	10.01.1979			P30	117

TABLE 9. INVALID PROPERTIES SETTING FOR COMPNAY MODEL

EID	Hire Date	DOB	DID	Dbudget	PID	Pbudget
E1	15.12.2005	15.12.1978	D2	500	P4	130
E5	16.12.2005	16.12.1978			P5	230
E6	17.12.2005	17.12.1978			P6	270
E10	21.12.2005	21.12.2010	D4	1300	P10	37
E11	22.12.2005	22.12.1978			P11	41
E12	23.12.2005	23.12.1978			P12	45



(a) VALID MODEL VERIFICATION RESULT



(b) INVALID MODEL VERIFICATION RESULT

FIG. 5. VERIFICATION RESULTS

5. CONCLUSION

UML Class/OCL model is an important part of UML. It serves as a graphical notation for representing real-world entities without any formal verification mechanism. Numerous formal and semi-formal methods have been used for verification of UML Class/OCL model. This paper proposes a new method for verification of UML Class/OCL model and outlines how different features of the model can be mapped into ontology and SPARQL NAF Queries. Moreover, this work proposes how CWA and UNA can be obtained in ontology through different techniques. This work also presents how the proposed method can tackle all aspects of UML Class/OCL model verification presented in existing literature such as consistency verification, extensive integer computation, string processing, and logical consequences. Finally, implemented the proposed method and developed a prototype tool to provide a proof of concept.

6. FUTURE WORK

The future work will cover more elements (xor constraints and dependency relationships) of the UML class diagram which have not yet been covered by any existing method and will focus on scalability problem. Furthermore, the tool will be extended with the support the automatic transformation of UML Class/OCL model into the ontology and SPARQL NAF queries.

ACKNOWLEDGMENT

Authors acknowledge the support by Jena team for providing the assistance in using the API for ontology processing. Authors are indebted to the referees for valuable comments/suggestions and also thankful to the Editorial Board, Mehran University Research Journal of Engineering & Technology, Jamshoro, Pakistan, for providing a platform to publish our research.

REFERENCES

[1] Baier, C., and Katoen, J.P., "Principles of Model Checking", The MIT Press Cambridge, Massachusetts, USA, 2008.

[2] Anastasakis, K., Bordbar, B., Georg, G., and Ray, I., "UML2Alloy: A Challenging Model Transformation", ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, Volume 4735, pp. 436-450, Springer, USA, 2007.

[3] Traore, I., and Aredo, D.B., "Enhancing Structured Review with Model-Based Verification", IEEE Transactions on Software Engineering, Volume 30, No. 11, pp. 736 - 753,2004.

[4] Przigoda, N., Filho, J.G., Niemann, P., Wille, R., and Drechsler, R., "Frame Conditions in Symbolic Representations of UML/OCL Models", ACM/IEEE International Conference on Formal Methods and Models for System Design, pp. 65-70, Kanpur, 2016.

[5] Anastasakis, K., Bordbar, B., Georg, G., and Ray, I., "On Challenges of Model Transformation from UML to Alloy", Software & Systems Modeling, Volume 9, No. 1, pp. 69-86, Springer, 2010.

[6] Malgouyres, H., and Motet, G., "A UML Model Consistency Verification Approach Based on Metamodeling Formalization", ACM Symposium on Applied Computing, pp. 1804-1809, 2006.

[7] Cadoli, M., Calvanese, D., Giacomo, G.D., and Mancini, T., "FiniteSatisfiability of UML Class Diagram by Constraint Programming", International Workshop on Description Logics, Volume 104, BC, Canada, 2004.

[8] Shaikh, A., and Wiil, U.K., "A Feedback Technique for Unsatisfiable UMLOCL Class Diagrams", Software Practice and Experience, Volume 44, No. 11, pp. 1379-1393, Wiley, 2013

[9] Shaikh, A., and Wiil, U.K., "Efficient Verification-Driven Slicing of UML/OCL Class Diagrams", International Journal of Advanced Computer Science and Applications, Volume 7, No. 5, pp. 530-547, UK, 2016.

[10] Encarnaci, M., Barrio-sol, M., Cuesta, C.E., and Fuente, P.D., "UML Automatic Verification Tool with Formal Methods", Volume, 127, pp. 3-16, 2005.

[11] Clark, T., and Evans, A., "Foundations of the Unified Modeling Language", 2nd BCS-FACS Conference on Northern Formal Methods, pp. 1-14, Springer, UK, 1997.

[12] Cali, A., Calvanese, D., Giacomo, G.D., and Lenzerini, M., "A Formal Framework for Reasoning on UML Class Diagram", 13th International Symposium on Foundations of Intelligent Systems, pp. 503-513, France, 2002.

[13] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., and Katz, Y., "Pellet: A Practical OWL-DL Reasoner", Journal of Web Semantics: Science, Services and Agents on the World Wide Web archive, Volume 5, No. 2, pp. 51-53, 2007.

[14] Mahmud, N., "Ontology-Based Analysis and Scalable Model Checking of Embedded Systems Models" M{ä}lardalen University, 2017.

[15] Nguyen, T.H., Grundy, J.C., and Almorsy, M., "Ontology-Based Automated Support for Goal-Use Case Model Analysis", Software, Quality Journal, Volume 24, No. 3, pp. 635–673, 2016.

[16] Corea, C., and Delfmann, P., "Detecting Compliance with Business Rules in Ontology-Based Process Modeling", Proceedings of 13th Internationale Tagung Wirtschaftsinformatik, pp. 226-240, 2017.

[17] Liao, Y., Panetto, H., Simão, J.M., and Stadzisz, P.C., "Ontology-Based Model-Driven Patterns for Notification-Oriented Data-Intensive Enterprise Information Systems", Proceedings of 7th International Conference Information Social Technology, pp. 148-153, 2017.

[18] He, H., Wang, Z., Dong, Q., Zhang, W., and Zhu, W., "Ontology-Based Semantic Verification For UML Behavioral Models", International Journal of Software Engineering and Knowledge Engineering Volume 23, No. 2, pp. 117-145, World Scientific, 2013.

[19] Dilo, A., Zlatanova, S., and Oosterom P.V., "Modeling Emergency Response Processes: Comparative Study on OWL and UML", Joint ISCRAM-China and GI4DM Conference, 2008.

[20] Bahaj, M., and Bakkas, J., "Automatic Conversion Method of Class Diagrams to Ontologies Maintaining Their Semantic Features", International Journal of Soft Computing and Engineering, Volume 2, No. 6, 2013.

[21] Belghiat, A., and Bourahla, M., "UML Class Diagrams to OWL Ontologies: A Graph Transformation Based Approach", International Journal of Computer Applications, Volume 41, No. 3, pp. 41-46, 2012.

[22] Parreiras, F.S., and Staab, S., "Using Ontologies with UML Class-Based Modeling: The TwoUse Approach", Data & Knowledge Engineering, Volume 69, No. 11, pp. 1194-1207, Elsevier, 2009.

[23] France, R., Evans, A., and Lano, K., "The UML as a Formal Modeling Notation",International Conference on UMLBeyond the Notation LNCS 1618, pp. 325-334, Berlin, Germany, 1998.

[24] Kim, S.K., and Carrington, D., "A Formal Mapping between UML Models and Object-Z Specifications", International Conference on Formal Specification and Development in Z and B, Volume 1878, pp. 2-21, UK, 2000.

[25] Truong, N.T., and Souquieres, J., "An Approach for the Verification of UML Models Using B", 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 195-202, Czech Republic, 2004.

[26] Cabot, J., and Teniente, E., "Incremental Integrity Checking of UMLOCL Conceptual Schemas", Journal of Systems and Software, Volume 82, No. 9, pp. 1459-1478, Elsevier, Spain, 2009.

[27] Maoz, S., Ringert, J.O., and Rumpe, B., "CD2Alloy Class Diagrams Analysis Using Alloy Revisited Model Driven Engineering Languages and Systems", Lecture Notes in Computer Science Volume 6981, pp. 592-607, Springer, New Zealand, 2011

[28] Shaikh, A., and Wiil, U.K., "Evaluation of Tools and Slicing Techniques for EfficientVerification of UMLOCL Class", Advances in Software Engineering Archive, pp. 1-18, Hindawi New York, USA, 2011.

[29] Gogolla, M., Buuttner, F., and Cabot, J., "Initiating a Benchmark for UML and OCL Analysis Tools", Tests and Proofs Lecture Notes in Computer Science, Volume 7942, pp. 115-132, Hungry, 2013.