Ontology-Based Transformation and Verification of UML Class Model

Abdul Hafeez[1], Syed Abbas[2], and Aqeel-ur-Rehman[3]
[1]Department of Computer Science, SMI University, Karachi
[2]Faculty Engineering Science and Technology, Indus University, Karachi
[3]Faculty of Engineering Science and Technology, Hamdard University, Karachi

Abstract: *Software models describe structures, relationships and features of the software system. Especially, in Model Driven Engineering (MDE), they are considered as first-class elements instead of programming code and all software development activities move around these models. In MDE, programming code is automatically generated by the models and models' defects can implicitly transfer to the code. These defects can harder to discover and rectify. Model verification is a promising solution to the problem. The Unified Modelling Language (UML) class model is an important part of UML and is used in both analysis and design. However, UML only provides graphical elements without any formal foundation. Therefore, verification of formal properties such as consistency, satisfiability and consequences are not possible in UML. This paper mainly focuses on ontology-based transformation and verification of the UML class model elements which have not been addressed in any existing verification methods e.g. xor association constraint, and dependencies relationships. We validate the scalability and effectiveness of the proposed solution using various UML class models. The empirical study shows that the proposed approach scales in the presence of the large and complex model.*

Keywords: *UML Class Model Verification, Dependency Relationship, XOR Association Constraints.*

Received September 11, 2017; accepted January 28, 2019
https://doi.org/10.34028/iajit/17/5/9

1. Introduction

Software design models represent real-world entities on a smaller scale and provide a clear understanding of the system. In Model Driven Engineering (MDE), they are considered as the nucleus of all development activities and are recognized as first-class elements instead of programming language code [21, 25, 45]. They are not only used for documentation, but they are core arte facts and processable by a computer [38]. In MDE, the model to model transformation automatically transfer source model to the target model [45]. The automatic transformation provides the systematic reuse of existing arte facts. However, it can cause some problems, for example, models may be developed with errors, and these errors can implicitly transfer to the target model (in MDE, programming code is also considered as a model) [41, 46].

Unified Modelling Language (UML) is a graphical modelling language and is commonly used in MDE [21, 37]. It offers various diagrams for dealing with different aspects of software [14, 29]. The class diagram is the most important part of UML [2, 14, 29, 37, 41] and performs a key role in software analysis and design [38, 42]. It describes the system through concepts, relationships, and constraints [16]. The main elements of the class diagram are classes and different types of relationships such as dependency, association, and generalization [39]. Association and generalization are also dependency relationships; however, they have specific semantics. [39]. These three relationships (dependency, association, generalization) are the basic relational building block of UML and in object-oriented modeling, they are considered most important elements [9].

UML only provides graphical elements for designing models without reasoning support, due to lack of formal foundation [27, 48]. Therefore, many researchers have used many formal and semi-formal methods for verification of UML class model such as Z notation, B method, Alloy, Constraint Satisfaction Problem (CSP). The current UML class model verification methods are sound and provide great effort to check the correctness of the model. However, they do not support some important elements of the UML class model. A comparison of existing class model verification methods presented in [43], which claimed that the dependency relationships have not been supported by any verification method. The xor constraint on association is another graphical constraint which is not supported by current verification methods. However, some verification methods which support verification of Object Constraints Language (OCL) cam verified xor constraint if xor specified in the form of OCL. Although, OCL has some limitations such as UML specification does not restrict constraint specification language and constraints can be defined through any formal language such as OCL or informal language such as JAVA, C# or even natural language [33, 50]. Furthermore, most of the UML CASE tools do not support OCL or provide limited support [24, 34]. The actual use of OCL in the software industry has been nearly insignificant [19, 34]. Due to its pure textual nature, designers are uncomfortable when they combine it with diagrammatic paradigm and organizations which heavily use UML even they lightly use OCL in their projects [19, 34, 47].

The ontology also specifies the real-world concepts like UML class model and also supports reasoning. It has various elements such as classes, relations, and individuals like the UML class model. Recently, software engineering practitioners have started integrating ontology in software development practices (processes, methods, tools, etc.,) and in software components for improving the quality [15].

This work proposes ontology-based formalization and verification of UML class model elements (graphical xor constraint and dependency relationships) which have not been addressed by any existing verification method.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents an ontology-based formalization of the UML class model elements. Section 4 describes ontology-based reasoning on UML class model elements. Section 5 presents the implementation and empirical results obtained. Finally, section 6 presents our conclusions and points out future work.

1.1. Motivation

Most of the existing verification methods use formal or semi-formal methods and their specification notations are enormously inspired by mathematics. They are greatly different from the UML class model and they are difficult to be understood by software practitioners. On the other side, ontology and UML class model have many similar elements such as classes, relationships, and generalization. However, in this work, ontology as the target notation for transformation and verification is motivated by the fact that the reasoning on the ontology-based model can be easily performed by ontology reasoner and they have matured enough to support large problem space. They can perform reasoning and knowledge inference on thousands of ontological items within a reasonable time [44]. Therefore, ontology-based verification can improve the reliability of Model Driven Engineering in order to check the correctness of the model before the model to model transformation.

2. Related Work

Verification of UML class model has been addressed in several works. In the existing literature, different correctness properties have been addressed by researchers according to different aspects such as static model, dynamic model, inter-model and intra-model [11]. In the static UML class model, only structure elements such as association and generalization are considered for verification. In dynamic model, the behavior parts of the model such as operations are considered for verification. In the inter-model, consistency among different models is verified, and in the intra-model, consistency of model against the constraints is verified. Early verification works focused on the formalization of meta-model and well-formedness rules through different formal methods such a Z notation, B method, and Description Logic [6, 20, 27]. Furthermore, they also performed different analyses on UML class model such as diagrammatical transformation analysis performed by [19], in which a model is deduced from the other model through the numerous transformations. The authors of [37] performed an intersection between two or more class diagrams and performed refinement analysis. However, most of the recent works focus on the consistency of UML class models and almost all works verified satisfiability of the model [1, 6, 10, 11, 31, 43].

France *et al*. [20] proposed a precise formal foundation for UML core meta-model in Z notation and argued that a formal representation of UML meta-model provides many benefits such as clarity, consistency checking, extendibility, refinement, and proof checking. In this work, the UML core model is represented through a compositional schema which has many sub-schemas and each sub-schema closely correspond to the elements of core UML meta-model. Kim and Carrington [22, 23] defined abstract syntax of UML meta-model in Object-Z. In [23] authors argued that the formal representation of abstract syntax (meta-model) of language is a most recurrent technique for defining the semantics of the language. In [22] authors argued that the different formal methods have different strengths in different areas and a single method cannot deal with all aspects of UML model verification and validation. Consequently, the authors presented an integrated verification and validation framework which supports different formalisms.

Ledang and Souquières [27] and, Ledang [28] proposed transformation of the class model into the B method. In [28] authors checked the consistency of the UML class diagram against the well-formedness rules through B prover. In this approach, well-formedness rules are transformed into the invariants of B abstract machine. In [27] authors presented the transformation of OCL constraints into the B method. In this approach, OCL basic types (integer, float, etc.,) are transformed into the B basic types and operators (+, -, etc.,) transformed into the B basic operations. In [49] authors integrated all their previous work and presented the transformation of the class model and UML meta-model with well-formedness rules into the B specification for verification.

Cadoli *et al*. [14] presented the transformation of UML class model into the Constraint Satisfaction Problem (CSP) and proposed linear inequality-based method for finite model verification. In this work, authors addressed two verification problems

1. Satisfiability.
2. Full satisfiability.

The satisfiability verifies whether a finite non-empty instance model (object diagram) of the UML class model can be generated without violation of the constraints, and full satisfiability verifies whether an instance model can be generated without violation of the constraints where all classes can be successfully populated. However, this method does not support attributes, association classes and n-ary associations.

Cabot and Teniente [12] proposed incremental verification of a class model with OCL integrity constraints. In [12] authors argued that the verification of integrity constraints after every structure event (insert an entity, update attribute, delete an entity, etc.,) may be very expensive and inefficient. This work introduced, the term “Potential Structure Events” (PSEs) that includes only those events which can render constraints violation. In this approach, PSEs for every integrity constraint are recorded and only those instances of entities and relationships are verified which relate to any PSEs. In [13] authors used constraint programming for their method which was proposed in [12] and presented fully automatic, decidable solution for bounded verification of the UML Class/OCL model. The decidability is achieved through establishing finite bounds on instances of classes, associations, and domain for attributes. They also pointed out issues of the bounded verification and argued that the inadequate finite bound can miss defects if it is set too small or it may be time-consuming if set too large. Their proposed approach set initially large finite bounds and then bounds are tightened as much as possible through the interval constraint propagation technique [16].

Bordbar and Anastasakis [10] transformed UML Class/OCL model into the Alloy. In this method, UML meta-model transformed into Alloy and class model into the Alloy signature as an instance of the meta-model. Przigoda *et al*. [37] proposed the transformation of advanced features (multiple inheritance, and interface) of UML class model into the alloy. This work also supports various analyses on a class diagram such as intersection and refinement analysis.

Berardi *et al*. [6] represented UML class model through description logic and verified inconsistencies and redundancies. They argued that description logic-based reasoning supports high expressiveness of UML class model. Mainly, this work performs consistency verification (satisfiability) and class equivalence. They reported that the reasoning complexity of the UML class model is exptime-hard with minimum supporting features such as binary association, minimal multiplicity and generalization [7]. Maraee and Maraee [4], Maraee and Balaban [30], and Maraee *et al*. [31]. represented generalization set, qualified association, and association classes through linear inequalities. They also presented redundancy elimination method for wider constraints (Universal, and Extensional) of association, generalization, aggregation, composition and qualified association.

Shaikh *et al*. [43] used a model slicing technique for reducing the verification complexity of UML Class/OCL model. They reported that the slicing techniques decrease verification time of large model with fewer constraints and if the model has many disjoint sub-models then minimum slices will be created and efficiency will not be gained. They extended the work with the support of non-disjoint sub-models where the common class is used among several constraints. They also introduced a feedback technique for unsatisfiable UML Class/OCL model [42]. Seiter and Drechsler [40] pointed out the consistency among verified models is also important and current UML model verification methods do not focus on consistency among verified models. They proposed a framework for managing consistency among verified models.

Various research works have also used ontology for transformation and verification of UML class model. Xu *et al*. [51] performed a comparison of UML and Web Ontology Language (OWL) and specified that both have many similarities e.g., classes, relationships, and attributes. They also pointed out differences between UML and OWL such as UML has various relationships among classes (such as association, aggregation, and composition) and OWL only has an object property. Finally, they concluded that both are compatible with each other. Bahaj and Bakkas [3] proposed a transformation technique from class diagram to ontology and considered encapsulation, aggregation, and composition as special types of association. Belghiat and Bourahla [5] presented graph-based transformation of class diagram meta-model into the ontology. Parreiras and Staab [35] combined UML with OWL-DL for representing software model. They They integrated Metaobject Factory (MOF) meta-model as the backbone of both UML and OWL.

3. Class Model to Ontology

This section describes the transformation of UML class model elements into the ontology. Firstly, elements which are common in both and have already been represented in existing work are presented with little augmentation. Then transformations of elements which do not have direct corresponding ontology elements and have not been addressed in any existing work are presented.

As an example, consider the UML class model shown in Figure 1 a partial representation of the library information system. The class model involves several classes (Book, publisher, Librarian, etc.,) as well as various associations (Record, Write, Review, etc.,) a dependency relationship between Librarian and Catalog classes. In the model, the Book class is connected to Person class by Write and Review associations which are mutually exclusive by xor constraint and semantically specifies that if a person writes a book, then he/she cannot review the same book and vice versa. Further, the Book class is connected to Member class by Borrowed and Reserved associations which are also mutually exclusive by xor constraint and semantically specifies that if a member borrows a book, then he/she cannot reserve the same book and vice versa. Finally, the Book class connects to Publisher class by Donated by and Purchase by associations which are mutually exclusive by xor constraint, and semantically specify that if a book purchased by the publisher then it will not be donated by the publisher and vice versa.

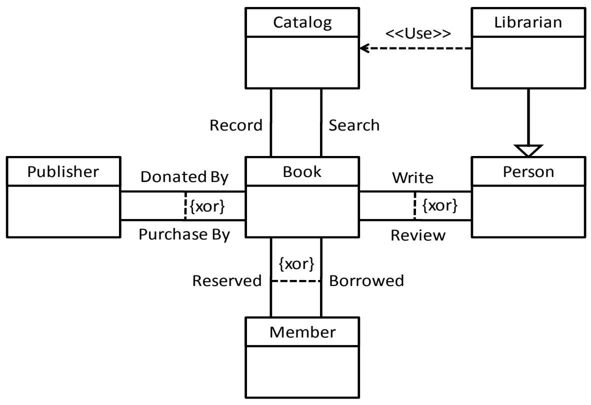


Figure 1. A partial UML class model of the library information system.

3.1. Translation of Classes, Attributes, and Associations

In the existing work, UML classes are transformed into ontology's classes. However, UML supports Unique Name Assumption (UNA) where each instance of the class is considered as a different entity. On the other side, the ontology does not support UNA. Although, the semantic of UNA can be achieved in the ontology by the addition of some supplementary elements for example, at the class level an additional functional data property Object Identifier (OID) is attached to each class as a key through the Has Key construct. At the instance level, the all different assertion is used to differentiate instances of classes. The attributes of classes are transformed into the data type property. The association relationships between classes are represented by the object property, and related classes are assigned as domain and range. Additionally, an inverse property is also declared for representing two-way communication. The multiplicities are transformed into qualified cardinalities.

3.2. XOR Association Constraints

In a class diagram, classes are connected to each other through multiple associations and these associations can be mutually exclusive by xor constraint as shown in Figure 1 where the Book and Member classes are connected by Reserved and Borrowed associations. The xor constraint can also be applied on single association when an association is connected to more than one class as shown in Figure 2 where Account class (source) is connected to the Person and Company classes (target) by Belong association. In this case, xor constraint restricts the instance of source class can be linked to the instance of one target class.

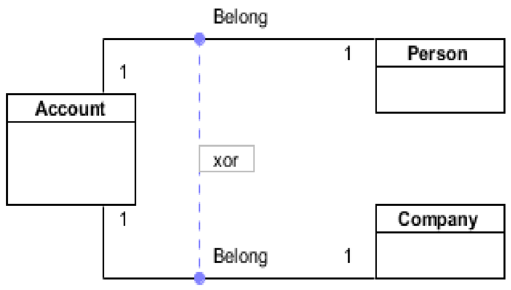


Figure 2. XOR constraint on single association.

In the proposed method, for the first case (where xor applied on multiple associations) xor associations are declared as disjoint to each other. For example, Reserved and Borrowed associations of library information system (shown in Figure 1) are formalized as:

$$Book \sqsubseteq \forall(Reserved.Book \sqcup Borrowed.Book \quad Reserved \sqsubseteq \neg Borrowed \tag{1}$$

For the second case (where a single association connects multiple classes) the source class declared with additional restrictions. For example, the Belong association of bank model (shown in Figure 2) is formalized as:

$$Account \sqsubseteq \exists(Belong.person \sqcup Belong.Compnay) \sqcap (\neg\exists(Belong.Compnay \sqcap Belong.Person)) \tag{2}$$

3.3. Dependency Relationship

The dependency relationships between classes are semantic relationships, which specify that a change in the class (independent/supplier class) can affect other classes (dependent/client classes). In UML, the dependency relationships are used in various diagrams. This work only focuses on the dependency relationships which are relevant to the class model and especially impact on the model consistency and satisfiability. The dependency relationships can be categorized into different groups:

- Abstraction: shows the relationship between two classes on different levels of abstraction (for example analysis and design model) or same level (client represents a more concrete form of supplier). Generally, abstraction dependency is subdivided into the trace, refine, realization and derivation.
- Permission: in the permission dependency, supplier class grants access it private contents to the client class. Generally, permission dependency is subdivided into access, import, and permit. The access and import dependencies are used in the package diagram.
- Usage: the usage dependency specifies that a class requires another class for proper execution. In usage dependency, the presence of supplier is mandatory for the client. It is only dependency type which is used in the same model level. Generally, usage dependency is subdivided into the call, create, insatiate, send and use.

The create and insatiate dependencies create an instance of the supplier class. In insatiate dependency, the operation of client class creates an instance of the supplier class and in create dependency supplier class object is declared as property of client class.

The trace and refine are used when the connection is established between classes in the different models. In these relationships, the client is considered more detail specification of the supplier. The realization dependency is established between abstract class/interface and a concrete class. The concrete class provides an implementation of operations specified in abstract class/interface. In all abstract dependency types except drive, client class instance does not need the instance of the supplier and they relate to each other only on classifier level. In model verification, the crucial dependencies are: where the presence of supplier class instance is mandatory for client class instance. Therefore, the drive, call, create, and use are dependency types in which instance of the client requires the supplier's instance. Therefore, this work only considers these dependencies for transformation and verification.

In the proposed approach, dependency relationships are transformed into the object property with additional restrictions. For example, the use and call dependencies are transitive and drive, create are transitive as well as asymmetric. However, OWL-DL does not support reasoning over object property which is declared both transitive and asymmetric. Though, the semantic of transitive and asymmetric can be achieved through some techniques for example in the proposed solution dependency relationships which are transitive and asymmetric are declared as object property with two additional sub-properties. One sub-property is marked as transitive and other is marked as asymmetric. Then related classes are connected with the sub-properties. When inference is generated by the reasoner then consequently the related classes are linked by the parent object property. Table 1 shows the transformation of dependency relationships into the ontology.

Table 1. Representation of different dependency relationships in ontology.

Dependency Relationships	**Equivalent ontology**	**Description**
Create	$createT, createA \sqsubseteq create$ $createT \equiv createT \circ craeteT$ $createA \equiv \neg createA^{-}$	Sub-property Transitive Asymmetric
Drive	$driveT, driveA \sqsubseteq drive$ $driveT \equiv driveT \circ driveT$ $driveA \equiv \neg driveA^{-}$	Sub-property Transitive Asymmetric
Call	$call \equiv call \circ call$	Transitive
Use	$use \equiv use \circ use$	Transitive

4. Reasoning on Class Model

A class model is a combination of various relationships such as associations, generalization, dependency relationship and aggregation. If the verification method does not support any one of them, thus verification of the entire model cannot be possible. Existing verification methods either ontology-based or others do not support dependency relationships and graphical XOR constraints. Moreover, they do not deal with the consequences, which is an important part of the verification. Since due to the consequences sometimes the model can be inconsistent. Thus, the transformation of a class model into the ontology perfectly captures the semantics of the elements and provides the ability to reason.

The next section shows the soundness and completeness of the proposed transformation of xor constraints and dependency relationships.

4.1. Soundness and Completeness

4.1.1. Case 1 xor Constraint (Soundness)

Given a transformed ontology model (which has type 1 xor constraints) M. The model is considered sound

$$\begin{gathered} iff\ \forall\ p_i, p_{i+1}, p_{i+2} \ldots p_n \in P, \\ \mathcal{M} \vDash p_i \sqcap p_{i+1} \sqcap p_{i+2} \sqcap \ldots \sqcap p_n \sqsubseteq \perp \end{gathered} \tag{3}$$

Where P is set represents the disjoint object properties.

4.1.2. Case 1 xor Constraint (Completeness)

Any complete and constraint violation free set of instances I has a model. This can be proved by defining the canonical interpretation $\mathfrak{I}$I induced by I:

a) The domain of $\Delta^{\mathfrak{I}_I}$ of $\mathfrak{I}_I$ consist of all the instance of classes in I;
b) For all classes C we define $C^{\mathfrak{I}_I} = \{b | c(b) \in I\}$;
c) For disjoint object properties $P_1\ and\ P_2$ we define
$$P^{\mathfrak{I}_I} = \{(b,d) |\ P_1^{\mathfrak{I}_I}(b,d) \sqcap \left(\neg\ P_2^{\mathfrak{I}_I}(b,d)\right) \in I\}$$

According to the definition $\mathfrak{I}_I$ satisfies all the assertion in I.

- *Example* 1 (Case 1 XOR Constraint) Consider a fragment of the class model presented in Figure 1 where an instance of the class Person can be linked to Book class instance either through Write or Review Otherwise, the model will be unsatisfied.

4.1.3. Case 2 Xor Constraint (Soundness)

Given a transformed ontology model (which has type 2 xor constraints) *M.* The model is considered sound

$$\begin{gathered} iff\ \forall\, p_i, p_{i+1}, p_{i+2} \dots p_n \in P, \\ \mathcal{M} \vDash p_i.(a,b)\ \sqcap\ p_i.(a,c) \sqsubseteq \bot \end{gathered} \tag{4}$$

Where a is an instance of source class and b and c are instances of the target classes and P represent set of an object property.

4.1.4. Case 2 Xor Constraint (Completeness)

Any complete and constraint violation free set of instances I has a model. This can be proved by defining the canonical interpretation $\mathfrak{T}$I induced by I:

a) The domain of $\Delta^{\mathfrak{T}_I}$ of $\mathfrak{T}_I$ consist of all the instance of classes in *I*;
b) For all classes *C* we define $C^{\mathfrak{T}_I} = \{b|c(b) \in I\}$;
c) For object properties $P_1\ and\ P_2$ we define $P^{\mathfrak{T}_I} = \{(b,d)(d,c)|\ P_1^{\mathfrak{T}_I}(b,d) \sqcap \left(\neg\ P_1^{\mathfrak{T}_I}(b,c)\right) \in I\}$

According to the definition $\mathfrak{T}_I$ satisfies all the assertion in *I.*

- *Example* 2 (Case 2 XOR Constraint) Consider a class model presented in Figure 2, according to the proposed approach an instance of Account class can be connected to either Person or Company instance through association "belong". Otherwise, the model will be unsatisfied.

4.1.5. Create and Drive Dependency (Soundness)

Given a transformed ontology model (which has create or drive dependency constraints) M. The model is considered sound

$$\begin{gathered} iff\ \forall\, t_i, t_{i+1}, t_{i+2} \dots t_n \in T, \\ \forall\, a_i, a_{i+1}, a_{i+2} \dots a_n \in A, \\ \forall\, c_i, c_{i+1}, c_{i+2} \dots c_n \in C \\ T, A \sqsubseteq C \\ t_i.(b,d)\ \sqcap\ t_i.(b,e) \equiv t_i(b,e) \\ a_i(b,d) \equiv\neg\ a_i(d,b) \\ \mathcal{M} \vDash t_i(b,e), \sqcap \left(\neg\ a_i(b,d)\right) \sqsubseteq c_i \end{gathered} \tag{5}$$

Where *T* is a transitive object property, *A* is an asymmetric object property, *C* is the super property of *T* and *A* and *b*,*d* and *e*, are classes.

4.1.6. Create and Drive Dependency (Completeness)

Any complete and constraint violation free set of instances I has a model. This can be proved by defining the canonical interpretation $\mathfrak{T}$I induced by I:

a) The domain of $\Delta^{\mathfrak{T}_I}$ of $\mathfrak{T}_I$ consist of all the instance of classes in *I*;
b) For all classes *C* we define $C^{\mathfrak{T}_I} = \{b|c(b) \in I\}$;
c) For all transitive object properties *T* we define $T^{\mathfrak{T}_I} = \{(b,d,c)|\ (T^{\mathfrak{T}_I}(b,d) \sqcap T^{\mathfrak{T}_I}(d,c)) \Rightarrow T^{\mathfrak{T}_I}(b,c) \in I\}$
d) For all asymmetric object properties *A* we define $A^{\mathfrak{T}_I} = \{(b,d)|\ (A^{\mathfrak{T}_I}(b,d) \sqcap (\neg\ A^{\mathfrak{T}_I}(d,b)) \in I\}$ For all super properties *S* of *A* and *T* we define $S^{\mathfrak{T}_I} = \{(b,d,c)|\ (A^{\mathfrak{T}_I}(b,d) \sqcap (\neg\ A^{\mathfrak{T}_I}(d,b) \sqcap ((T^{\mathfrak{T}_I}(b,d) \sqcap T^{\mathfrak{T}_I}(d,c))\ \Rightarrow S^{\mathfrak{T}_I}(b,c) \sqcap \neg\ S^{\mathfrak{T}_I}(c,b)) \in I\}$

According to the definition $\mathfrak{T}_I$ satisfies all the assertion in *I.*

5. Empirical Evaluation

We conducted an empirical evaluation to answer the two research questions: which address the scalability and performance of the proposed approach. The research questions are:

- Does the proposed approach scale to a practical extent?
- How does the proposed approach compare to OCL based approaches?

The following subsections overview the subject of the study and the experimental setup, and describe, for each research question, the measurements performed and the achieved results.

5.1. Experiment Setup and Subject of the Study

We implemented our approach as a Java prototype[1] that relies upon the Jena library for processing the ontology. The Jena is an open source Java framework which provides support for extracting and writing Resource Development Framework (RDF), and OWL graph. It includes a rule-based inference engine to perform reasoning on ontology. The input UML class model of the prototype tool is represented in Extendable Markup language (XMI) XML Metadata Interchange). XMI is an Object Management Group (OMG) standard for exchanging metadata information by XML. Specifically, the XMI is intended to provide a common format for UML diagrams for sharing among different Computer Added Software Engineering (CASE) tools. So the input models of our tool must be specified in XMI version 2.41.

As the subject of our study, we considered 9 UML class models in which 5 models have xor constraints and 4 models have dependency relationships. The brief detail of class models used in the analysis are as follows:

[1]All source files of prototype tool and model files are available on https://sites.google.com/view/uml2onto

- *Salesman*: the Salesman model is an invalid model which contains 3 classes and 1 xor. This model is variant of XOR model which presented in [36].
- *Library*: the first example library model contains 8 classes, 8 associations in which 6 associations annotated with xor constraint (first case multiple associations). This model derived from model which presented in [18].
- *Restaurant*: the second example "Restaurant" has 8 classes, 8 associations in which 6 associations annotated with xor constraint (second case single association). This model derived from model which presented in [32].
- *Cinema Ticking*: the second example "Cinema Ticking" has 4 classes, 4 associations in which 4 annotated with xor constraint (both cases).
- *Script* 3: the script 3 model is a programmatically generated model, for verification of many xor associations. This model is variant of script model which presented in [43].
- *REST Full Web Service*: the Representational State Transfer (REST) Full Web service class model describes the simpler extensible framework for REST full web service [17].
- *Monopoly Game*: the Monopoly Game class diagram illustrates the static view of a game startup that describes the creation of various startup components through *Create* dependency [26].
- *Monitor Web Service Usage*: the Monitor Web Service Usage class model describes the technical implementation of several web services for business models including a free trial, charging on calls, charging on a monthly subscription [8].
- *Script* 5: The script 5 model is a programmatically generated model, for verification of many dependency relationships. This model is variant of script model which presented in [43].

We used UML to CSP and Alloy for comparison with the proposed method because these two verification methods are widely used and other methods such as USE and Mova are used for model validation. UML2Alloy transforms the UML/OCL class model into Alloy specification; therefore, they are same as in Alloy. For checking performance and scalability of the proposed method, the experiments run on Intel Core i7 3.40 GHz machine with 4GB of RAM. However, to allow for a fair comparison between the different methods, the experimental runs were each executed on a computer having the same characteristics. The comparison experiments were run Intel Core2Duo 1.34 GHz machine with 2 GB of RAM. Due to UMLtoCSP does not support 64-bit architecture.

Table 2. Description of xor models used in the evaluation and verification time.

Model Name	Classes	Associations	Xor	Case 1	Case 2	Verification Time
Salesman	3	2	1	✓	×	0.035
Library	6	8	3	×	✓	0.103
Cinema Ticking	4	4	2	✓	×	0.080
Restaurant	8	8	3	✓	✓	0.100
Script III	100	200	100	×	✓	0.547

5.2. RQ1: Does the Proposed Approach Scale to A Practical Extent?

5.2.1. Measurements and Setup

The proposed approach should be fast enough and scale effectively as classes and xor associations increase. For this reason, to respond to RQ1, we applied the proposed approach to various class models as shown in Table 2. For each UML class model, we measured the verification time with the proposed approach. In the experiment, we used different UML class models which focus on different types of xor constraints. Table 2. Shows the detail of each model regarding the number of classes, number of associations, number of xor constraints and type of xor constraint.

For verification of dependency relationships, we also performed an analysis on four different models as shown in Table 3. For each model we measured the verification time, especially, we analyzed the relationship between execution time and model size.

Table 3. Description of dependency relationships models used in the evaluation and verification Time.

Model Name	Classes	Associations	Xor	Case 1	Case 2	Verification Time
Salesman	3	2	1	✓	×	0.035
Library	6	8	3	×	✓	0.103
Cinema Ticking	4	4	2	✓	×	0.080
Restaurant	8	8	3	✓	✓	0.100
Script III	100	200	100	×	✓	0.547

5.2.2. Results

Table 2 shows the average execution time (in seconds) required to verify the xor model. The Table 2 also shows that the approach scales to a practical extent. For first model Salesman, the proposed approach requires on average 0.035 seconds with 3 classes and 1 xor association. In the case of Library model which have xor constraints of type 2 (xor on multiple associations), the proposed approach requires on average 0.103 seconds with 8 classes and 3 xor association constraints. In the case of Restaurant model which have xor constraints of type 1 (xor on single association), the proposed approach required on average 0.100 seconds with 8 classes and 3 xor associations. In the case of Cinema model which has both types of xor constraints takes on average 0.080 seconds. Finally, for checking the performance of the proposed method on a large model experiment performed on Script III which has 100 classes, 200 associations, and 100 xor constraints and requires on average 0.570 seconds. Library and Restaurant have the same number of classes, associations, and xor constraints but the type of xor constraints are different. The verification result shows that the verification time of both types of xor constraint more or less same.

Table 3 shows the average execution time (in seconds) required to verify dependency relationships. In the case of test inputs containing 9 classes and 3 dependency relationships, the approach requires on average 0.079 seconds. In the case of test inputs containing 6 classes and 5 dependency relationships the proposed approach requires on average 0.157 seconds. In the case of input containing 11 classes and 10 dependency relationships the proposed approach required on average 0.109 seconds. Finally, in the case of test input containing 100 classes and 100 dependency relationships, the proposed approach requires on average 0.473 seconds to verify the model.

Models containing hundreds of xor association and dependency relationships are particularly complex to verify and there is very less chance that a single model contains a number of elements like Scripts 3 and 5, which highlighted the scalabils and efficiency of our approach.

5.3. RQ2: How Does the Proposed Approach Compare to OCL Based Approaches?

5.3.1. Measurements and Setup

To be justified, the proposed approach should provide an advantage over OCL based approach that does not support direct verification of xor constraint. To respond to RQ2, we thus compared the performance of the approach proposed with UML to CSP and UML2Alloy which support verification of UML class model with OCL. In the comparison, we used four xor models

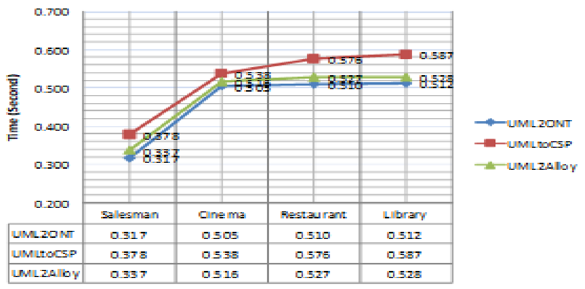
1. Salesman.
2. Library.
3. Cinema Ticketing.
4. Restaurant.

The comparison of dependency relationship verification with other verification method is not possible due to current verification methods do not support verification of the dependency relationships.

5.3.2. Results

Figure 3 shows the compassion results between existing methods (UML to CSP and UML to Alloy) and the proposed method. The x-axis of Figure 3 reports models and the y-axis reports the execution time taken in a second. As the results show proposed approach is a little bit efficient from the existing method. However, the proposed method has the following additional advantages over the OCL based approaches:

1. The graphical xor constraint automatically transformed into the ontology for verification and there is no need to manually transformed graphical xor constraints into the OCL which is also a time consuming and hectic task. Furthermore, OCL has many limitations which have been discussed in section 1.
2. The existing methods (UML to CSP and UML 2 Alloy) support bounded verification where they find a solution on limited search space. For example, in this evaluation, we set scope 4 for Alloy.



	Salesman	Cinema	Restaurant	Library
UML2ONT	0.317	0.505	0.510	0.512
UMLtoCSP	0.378	0.538	0.576	0.587
UML2Alloy	0.337	0.516	0.527	0.528

Figure 3. Comparison of different verification methods.

6. Conclusions and Future Work

This paper presents an ontology-based transformation and verification of UML class model elements (xor constraint and dependency relationships) which have been not supported by any existing method. Such transformations map class model elements into the ontology and facilitate the analysis such as consistency, satisfiability, and consequences. A benefit of this approach is that ontology has efficient reasoners which can perform reasoning on a large model in a reasonable time. As our future work, we plan to explore the transformation of OCL constraints into the ontology and perform verification of other unsupported UML class model elements, which will reveal further insight into existing state of relevant knowledge.

References

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

[2] Artale A., Calvanese D., and Ibáñez-García A., "Full Satisfiability of UML Class Diagrams," *in Proceedings of International Conference on Conceptual Modeling*, Vancouver, pp. 317-331, 2010.

[3] Bahaj M. and Bakkas J., "Automatic Conversion Method of Class Diagrams to Ontologies Maintaining their Semantic Features," *International Journal of Soft Computing and Engineering*, vol. 2, no. 6, pp.65-69, 2013.

[4] Balaban M. and Maraee A., "Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 3, pp. 1-45, 2013.

[5] Belghiat A. and Bourahla M., "From UML Class Diagrams to OWL Ontologies: A Graph Transformation Based Approach," *in Proceedings of International conference on Web and Information Technologies*, Sidi Bel Abbes, pp. 330-335, 2012.

[6] Berardi D., Calvanese D., and Giacomo G., "Reasoning on UML Class Diagrams," *Artificial Intelligence*, vol. 168, no. 1-2, pp. 70-118, 2005.

[7] Berardi D., Calvanese D., and Giacomo G., "Reasoning on UML Class Diagrams is EXPTIME-Hard," *in Proceedings of the International Workshop on Description Logics*, Rome, 2003.

[8] Bilgrami K., https://www.codeproject.com/Articles/34993/Mnitor-your-Web-Services-usage-via-NET-SOAP-Exten, Last Visited, 2016.

[9] Booch G., Rumbaugh J., and Jacobson I., *the Unified Modeling Language User Guide*, Addison-Wesley Professional, 2005.

[10] Bordbar B. and Anastasakis K., "UML2ALLOY: A Tool for Lightweight Modelling of Discrete Event Systems," *in Proceedings of the IADIS International Conference on Applied Computing*, Algarve, pp. 209-216, 2005.

[11] Cabot J. and Clarisó R., "UML-OCL Verification in Practice," *in Proceedings of International Workshop on Challenges in Model-Driven Software Engineering*, Toulouse, pp. 31-35, 2008.

[12] Cabot J. and Teniente E., "Incremental Integrity Checking of UML/OCL Conceptual Schemas," *Journal of Systems and Software*, vol. 82, no. 9, pp. 1459-1478, 2009.

[13] Cabot J., Claris R., and Riera D., "On the Verification of UML/OCL Class Diagrams Using Constraint Programming," *Journal of Systems and Software*, vol. 1, no. 93, pp. 1-23, 2014.

[14] Cadoli M., Calvanese D., De Giacomo G., and Mancini T., "Finite Satisfiability of UML Class Diagrams by Constraint Programming" *in Proceedings of Workshop on CSP Techniques with Immediate Application*, pp. 1-17, 2004.

[15] Calero C., Ruiz F., and Piattini M., *Ontologies for Software Engineering and Software Technology*, Springer Science and Business Media, 2006.

[16] Clarisó R., González C., and Cabot J., "Towards Domain Refinement for UML/OCL Bounded Verification," *in Proceedings of Software Engineering and Formal Methods*, York, pp. 108-114, 2015.

[17] Dambal V., https://www.ibm.com/developerworks/library/wsRESTservices/index.html, Last Visited, 2016.

[18] Somenath Mukhopadhyay., https://dzone.com/articles/designing-software-system, Last Visited, 2018.

[19] Fish A., Howse J., Taentzer G., and Winkelmann J., "Two Visualizations of OCL: A Comparison," [Online]. Available: http://www.cmis.brighton.ac.uk/research/vmg/VOCLTR.htm, Technical Report, University of Brighton, 2005.

[20] France R., Evans A., Lano K., and Rumpe B., "The UML as a Formal Modeling Notation," *Computer Standards and Interfaces*, vol. 19, no. 7, pp. 325-334, 1998.

[21] Hilken F. and Gogolla M, "User Assistance Characteristics of the USE Model Checking Tool," *in Proceedings of the 3rd Workshop on Formal Integrated Development Environment*, Limassol, pp. 91-97, 2017.

[22] Kim S. and Carrington D., "A Formal V&V Framework for UML Models Based on Model Transformation Techniques," *in Proceedings of Model Validation (MoDeVa) Workshop*, Montego Bay, 2005.

[23] Kim S. and Carrington D.,"A Formal Mapping between UML Models and Object-Z Specifications," *in Proceedings of International Conference of B and Z Users*, Monte Verità, pp. 2-21, 2000.

[24] Kjetil M., "A Pratical Application of the Object Constraint Language OCL," Agder University College, 2002.

[25] Lahrouni M., Cariou E., and Fazziki1 A., "A Black-Box and Contract-Based Verification of Model Transformations," *The International Arab Journal of Information Technology*, vol. 16, no. 4, pp. 651-659, 2019.

[26] Larman C., *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*, Prentice Hall, 2004.

[27] Ledang H. and Souquières J., "Integrating UML and B Specification Techniques," *in Proceedings of Informatik Workshop on Integrating Diagrammatic and Formal Specification Techniques*, Vienna, 2001.

[28] Ledang H., "Automatic Translation from UML Specifications to B," *in Proceedings of Workshop on Refinement of Critical Systems: Methods, Tools and Experience*, San Diego, pp. 23-25, 2002.

[29] Malgouyres H. and Motet G., "A UML Model Consistency Verification Approach Based on Meta-Modeling Formalization," *in Proceedings of ACM symposium on Applied computing*, Dijon, pp. 1804-1809, 2006.

[30] Maraee A. and Balaban M., "Efficient Recognition of Finite Satisfiability in UML Class Diagrams: Strengthening by Propagation of Disjoint Constraints," *in Proceedings of International Conference on Model-Based Systems Engineering*, Haifa, pp.1-8, 2009.

[31] Maraee A., Makarenkov V., and Balaban M., "Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagram," *in Proceedings of International Workshop on Model Co-Evolution and Consistency Management, (MoDELS'08)*, France, 2008.

[32] Mukhopadhyay S, https://dzone.com/articles/designing-software-system, Last Visited, 2018.

[33] OMG. "OMG Unified Modeling Language TM (OMG UML)" Superstructure v.2.3. 2010.

[34] Pandey R., "Object Constraint Language (OCL): Past, Present and Future," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 1, pp. 1-4, 2011

[35] Parreiras F. and Staab S., "Using Ontologies with UML Class-Based Modeling: The TwoUse Approach," *Data and Knowledge Engineering*, vol. 69, no. 11, pp.1194-1207, 2010.

[36] "Patterns, Anti-Patterns and Inference Rules Catalog for UML Class Diagrams," https://www.cs.bgu.ac.il/~cd-patterns/?page_id=392, Last Visited, 2018.

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

[38] Ruijters E., Schivo S., Stoelinga M., and Rensink A., "Uniform Analysis of Fault Trees Through Model Transformations," *in Proceedings of Annual Reliability and Maintainability Symposium*, Orlando, pp. 1-7, 2017.

[39] Rumbaugh J., Jacobson I., and Booch G., *Unified Modeling Language Reference Manual*, The Pearson Higher Education, 2004.

[40] Seiter J. and Drechsler R., "Development of Consistent Formal Models," *in Proceedings of Formal Modeling and Verification of CyberPhysical Systems*, Bremen, pp. 302-304, 2015.

[41] Shaikh A. and Wiil U., "Efficient Verification-Driven Slicing of UML/OCL Class Diagrams," *Journal of Advanced Computer Science and Applications*, vol. 7, no. 5, pp. 530-547, 2016.

[42] Shaikh A. and Wiil U., "Feedback Technique for Unsatisfiable UML/OCL Class Diagrams," *Software Practice and Experience*, vol. 44, no. 11, pp. 1379-1393, 2014.

[43] Shaikh A., Wiil U., and Memon N., "Evaluation of Tools and Slicing Techniques for Efficient Verification of UML/OCL Class Diagrams," *Advances in Software Engineering*, vol. 2011, pp. 1-18, 2011.

[44] Sirin E., Parsia B., Grau B., Kalyanpur A., and Katz Y., "Pellet: A Practical OWL-DL Reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp.51-53, 2007.

[45] Straeten R., Inconsistency Management in Model-Driven Engineering An Approach using Description Logics, PhD, Thesis, Vrije Univesiteit Brussel, 2015.

[46] Sun W., Combemale B., and France R., "Towards the Use of Slicing Techniques for an Efficient Invariant Checking," *in Proceedings of 14th International Conference on Modularity*, Fort Collins, pp. 23-24, 2015.

[47] Thoughts about OCL - The Object Constraint. 2016. [Online]. Available: http://www.shiftedup.com/2016/02/05/thoughts-about-ocl-the-object-constraint-language. Last Visited, 2016.

[48] Truong J. and Souquieres N., "An Approach for the Verification of UML Models Using B," *in Proceedings of International Conference and Workshop on the Engineering of Computer Based Systems*, Brno, pp. 195-202, 2004.

[49] Truong N. and Souquieres J., "Verification of UML Model Elements Using B," *Journal of Information Science and Engineering*, vol. 22, no. 2, pp. 357-373, 2006.

[50] UML Constraint. http://www.uml-diagrams.org/constraint.html, Last Visited, 2016.

[51] Xu W., Dilo A., Zlatanova S., and Oosterom P., "Modelling Emergency Response Processes: Comparative Study on OWL and UML," *in Proceedings of Information Systems for Crisis and Response Management. Proceedings of the Third Joint ISCRAM-CHINA and Gi4DM*, Harbin, pp. 493-504, 2008.

Abdul Hafeez is Assistant Professor at department of Computer Science, SMI University, Karachi. he is associated with SMIU since last 6 years. he was engaged as HoD of Computer Science Department with Institute of Business and Technology, Karachi. He received MS degree in Software Engineering degree from the Hamdard University and PhD degree in Computer Science with specialization in Software Engineering from the Hamdard University. His current research interests include Software Engineering, Model Verification and Ontology-Based Software.

Syed Abbas Musavi received the B.E. degree in electronics engineering and the Ph.D. and M.E. degrees in telecommunication engineering under HEC Scholarship from the Mehran University of Engineering and Technology, Pakistan. He is currently serving as the Dean of the Faculty of Engineering Science and Technology with Indus University, Karachi. He is on review board of two impact factor international journals. He is a member of numerous national and international societies, including the IEEEP Karachi Local Council, the IEEE Computer Society, the IEEE Signal Processing Society, the IEEE Devices and Circuits Society, and the IEEE Communications Society. He was a General Chair at the IEEE ICIEECT 2017.

Aqeel-Ur-Rehman received the B.S. degree in electronic engineering from the Sir Syed University of Engineering and Technology, Karachi, Pakistan, in 1998, the M.S. degree in information technology from Hamdrad University, Karachi, in 2001, and the Ph.D. degree in computer science with specialization in ubiquitous computing from the National University of Computer and Emerging Sciences, Karachi, in 2012. He is a Professor, the Deputy Director (Admin)-HIET, and the Chairman of the Department of Computing, Faculty of Engineering Sciences and Technology, Hamdard Institute of Engineering and Technology, Hamdard University, Karachi. His current research interests include sensor networks, ubiquitous computing, computer networks, and smart agriculture.