Received November 2, 2017, accepted December 12, 2017, date of publication January 4, 2018, date of current version February 14, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2786781

Ontology-Based Finite Satisfiability of UML Class Model

ABDUL HAFEEZ KHAN[1], SAYED HYDER ABBAS MUSAVI[2], AQEEL-UR-REHMAN[3], AND ASADULLAH SHAIKH[4]

[1]Department of Computer Science, Hamdard Institute of Engineering and Technology , Karachi 74600, Pakistan
[2]Department of Computing, Faculty of Engineering Science and Technology, Indus University, Karachi 75300, Pakistan
[3]Department of Computing, Hamdard Institute of Engineering and Technology, Karachi 74600, Pakistan
[4]Department of Information Systems, College of Computer Science and Information Systems, Najran University, Najran 61441, Saudi Arabia

Corresponding author: Abdul Hafeez Khan (abdul.hafeez@hamdard.edu)

ABSTRACT Software models are core artifacts in model driven engineering (MDE) and processable by computer. They are automatically transformed into other models and in MDE, programming code is also produced by the models. The automatic transformation provides a systematic reuse of existing artifacts. However, sometimes models are developed with defects and the defects can implicitly shift into the code, which may be difficult to discover and repair. A promising solution to this problem is model verification. UML class model is a key ingredient of MDE. However, UML only offers graphical components without the support of reasoning, due to lack of the formal foundation. Therefore, the verification of formal properties, such as consistency and finite satisfiability is not possible in UML. This paper proposes an ontology-based optimized verification method for important correctness property ''finite satisfiability'' of UML class model.

INDEX TERMS Finite satisfiability, model satisfiability, ontology-based satisfiability, model checking, model verification.

I. INTRODUCTION

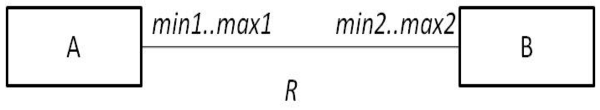
Software models present clear and precise description of the system [1]. In modern software development methodologies (e.g. MDE) they are considered as a nucleus and are recognized as a first-class elements instead of programming code [2]. As the core artifacts, they can be processable by computer and can be automatically transformed into other models (in MDE, programming code is also considered as the lowest level model) [2]. The automatic transformation gives the systematic reuse of existing artifacts. However, sometimes automatic transformation can cause some problems e.g. models are created in initial stages of software development and in the initial stages software development team does not well aware of the underdeveloped system and its constraints. Consequently, models are built with defects, and these defects can implicitly shift into the code [3]. Model verification can solve this problem [3]. However, the higher complexity of UML class model verification, current methods consumes a lot of computational resources (CPU, memory) and even in some cases verification result cannot be generated especially for the large and complex model [4], [5]. Cabot and Clarisó [6] pointed out many research directions for optimizing verification of the UML class model. This work mainly focuses on research direction search space reduction which identified in [6].

UML is a graphical modeling language and it is commonly used in MDE [7]. It offers various diagrams for dealing with different aspects of software and each describes the different views of software [8], [9]. The most important diagram of UML is a class diagram [10]–[15] and it describes static/structural aspects of the system [9]. UML only offers graphical components without the support of reasoning, due to lack of formal foundation [15], [16]. Therefore, verification of model correctness is not possible in UML.

The consistency and finite satisfiability are two major correctness facets of the UML class model. Both of them give surety of a non-empty and finite instance model. Consistency focuses on non-emptiness and finite satisfiability focuses on finiteness [14]. Contradicting constraints such as creating a subclass of two disjoint classes can cause emptiness and interaction among association multiplicity constraints which limit the number of links between instances of related classes can cause non-finiteness. A UML class model is considered consistent if and only if it has legitimate non-empty instances of all classes (maybe infinite), and finitely satisfiable if it has one legitimate finite instance where all classes are non-empty [17]. Hence, finding a single legitimate instance of the UML class model is adequate to guarantee consistency in which all classes are non-empty. Therefore, finite satisfiability can cover consistency [17]. This work proposes an optimized ontology-based method for verification of finite satisfiability of the UML class model.

II. BACKGROUND AND RELATED WORK

Verification of finite satisfiability of a static model (Entity Relationship Model similar to the UML class model) has been discussed in [18]–[20]. Mainly, there are two methods "linear inequalities method" and "detection graph method" which have been used for verification of various fragments of the UML class model. In the first method, finite satisfiability problem is solved through the finding a solution of linear inequalities. In the second method, a UML class model is represented through the directed graph and finite satisfiability is verified by the detection of the critical cycle in the graph.



$$A > 0;\ B > 0;\ R = 0;$$
$$\text{For } min1 \neq 0:\ R \geq min1{*}B;\ \text{for } max1 \neq \infty:\ R \leq max1{*}B$$
$$\text{For } min2 \neq 0:\ R \geq min2{*}A;\ \text{for } max2 \neq \infty:\ R \leq max2{*}A$$

FIGURE 1. Representation of association constraints in linear inequalities.

A. LINEAR INEQUALITY METHOD

The finite satisfiability of a conceptual schema (ERD model) was initially addressed by [19] and [20]. In the linear inequality method, cardinality constraints are transformed into the set of linear inequalities. Figure 1 shows, the UML class model and equivalent linear inequality representation.

According to Figure 1, there should be at least $min2 * B$ and $min1 * A$ and at most $max2 * B$ and $max1 * A$ links in the relationship R in order to satisfy association cardinality constraints, semantically, each instance of B should be connected to at least $min1$ and at most $max1$ instance(s) of A, and vice versa for A.

Many research works also used linear inequality system for representation and verification of other UML class model fragments such as Calvanese and Lenzerini [21] represented the class hierarchy constraint through the linear inequalities. Balaban and Maraee [10], [12], [13] also used the linear inequalities for representing class hierarchy constraints, generalization set, qualifier constraints, and association class constraints. Clarisó *et al.* [22] also used linear inequalities for bounded verification of UML Class/OCL model.

B. GRAPH-THEORETIC MODEL

The graph-theoretic model for detection of non-finite satisfiability of the UML class model addressed by [18] and [19].

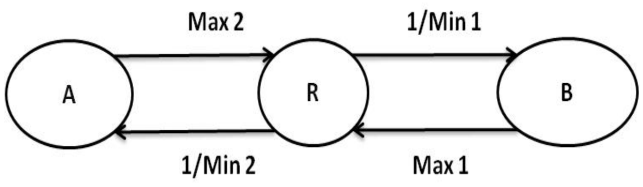


FIGURE 2. Graph theoretical model of UML class association constraints.

In this approach, a weighted directed graph is built, where nodes represent entities and relationships, edges join relationship nodes with their respective entities nodes and cardinality constraints are marked over the edges. Figure 2 shows the graph-theoretic model of the UML class model presented in Figure 1.

In this method, the weight of every cycle is calculated through the multiplication of weight of edges in the cycle. If the total weight of any cycle is calculated less than 1 then the cycle is considered as a critical cycle and indicates non-finitely satisfiable cardinality constraint. For detection of the critical cycle, the Bellman-Ford algorithm has been used with little alteration [18]. The Bellman-Ford is an efficient algorithm for obtaining the weight of the directed graph with the negative cycles. The complexity of the Bellman-Ford algorithm is $\Theta(|V||E|)$ from a single vertex. However, there is a possibility to miss negative cycle from a single vertex. Therefore, traversing is performed by all vertices which increases the complexity of the algorithm to $\Theta(|V^2||E|)$ and some paths are traversed many times. Queralt and Teniente [23] also used a graph-theoretic method for detecting dependency among OCL constraints. In the approach, cycles are searched for verifying the absence of an infinite model. Formica [24] used a graph-theoretic approach for checking finite satisfiability of object-oriented database schema and deals with cardinalities, navigation path, and comparison operators.

C. ONTOLOGY

Many researchers have used ontology for specification and verification of different software artifacts. Mahmud [25] presented an ontology-based domain specific language called ReSA for an embedded system. The ReSA uses ontology axioms for specification of the embedded system. The proposed approach checks refinements and requirement consistency and performs scalable formal verification of various Simulink models. Nguyen *et al.* [26] combined goal-oriented and use case modeling technique and proposed an ontology-based integrated framework for automated verification of incorrectness, incompleteness, and inconsistency. They also developed a prototype tool called GUITAR that takes requirements in the textual form and transforms them into the structured specification for performing automatic reasoning. The tool also generates comprehensive feedback for problems. The proposed approach also applied to six industrial case studies. Corea and Delfmann [27] combined the business rules with the ontology and proposed an ontology-based approach for verification of business processes against the business rules. In the approach, business rules are specified as a logic program. They used ontology reasoner to performed reasoning to detect model elements which violate the rules.

Liao *et al.* [28] discovered the challenges which faced by the legacy Enterprise Information System (EIS) in the fourth industrial revolution and proposed their potential solutions. They summarized seven main categories of requirements for future EIS and identified interoperability infrastructure issues related to the future EIS. They proposed a notification-based approach which derived from the notification-oriented paradigm. A notification-oriented paradigm is a new approach for software and hardware specification. Finally, they presented the ontology-based model-driven patterns for notification-oriented data intensive EIS.

Fellmann *et al.* [29] proposed ontology-based specification and verification of business process models. In this approach model elements are represented through the ontology and constraints through the rules then model and rules are verified by the inference engine. Sun *et al.* [30] presented ontology-based representation of software architecture components and transformed dynamic communication (constraints) into the SWRL. Mokos *et al.* [31] presented ontology-based transformation and verification of the dependable system and introduced ontology-based verification of safety-critical system. Kezadri and Pantel [32] proposed an ontology-driven knowledge base and related verification and validation technique for the behavioral model. Lapets *et al.* [33] used the ontology to represent the formal facts for improving the usability of the verification system. They proposed a lightweight verification system AARTIFACT that utilizes the ontology for the conceptualization of common mathematical concepts (e.g. numbers, sets, vectors, graphs).

Various research works also utilized an ontology for transformation and verification of UML models. He *et al.* [34] used an ontology for verification of UML behavioral model. They divided UML behavioral models into static semantics and dynamic semantics. The static semantics are represented by OWL DL and dynamic semantics are represented by DL-safe rules. Finally, the ontology is verified by pellet reasoner. Xu *et al.* [35] performed a comparison of UML Class diagram and web ontology language and specified that both have many similar elements e.g. classes, relationships, and attributes. They also pointed out the difference between these two languages such as UML has numerous relationships between classes (association, generalization, composition, etc) and OWL only has object property. Finally, they concluded that both are compatible with each other. Bahaj and Bakkas [36] proposed a different transformation technique from UML class model to ontology and considered encapsulation, aggregation, and composition as a special type of association. Belghiat and Bourahla [37] presented graph-based transformation of UML class model and transformed meta-model of the UML class model into the ontology. Parreiras and Staab [38] combined UML with OWL-DL for representing software model. They integrated MOF meta-model as the backbone for both UML and OWL. Guizzardi *et al.* [39] presented the ontological foundation to UML class diagram. They provide ontological adequacy to modeling language for efficiently representing the real-world semantics. They also argued that the ontological foundation of modeling language produces a conceptually cleaner, semantically unambiguous and ontologically well-founded version of the UML Class diagram. They also proposed a systematic evaluation method for comparing a meta-model of the UML class diagram with the reference ontology of the corresponding domain. They transformed the types and type taxonomies, roles, attributes, attribute values and attribute value spaces, relationships, and part-whole relations into the ontology [40]. They combined UML class diagram and Ontology into a language called ONTUML. the ONTUML meta-model reflects the ontological feature prescribed by Unified Foundational Ontology (UFO) and the formal constraint [41]. The ONTUML addresses the various conceptual modeling problems such as transitivity problem in the part-whole relationship, the collapse of cardinality constraints, association specialization, association sub-setting, and association redefinition.

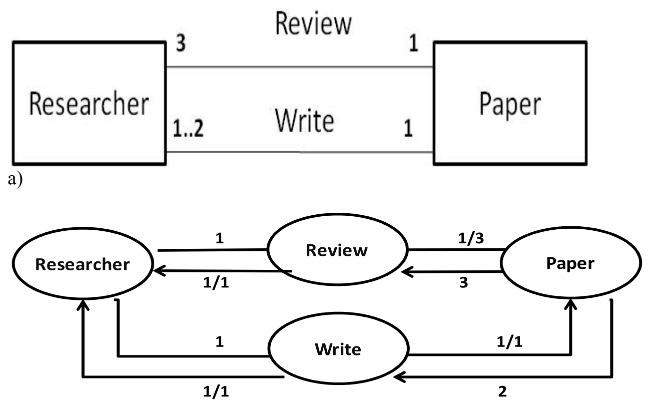


FIGURE 3. **Classical researcher and paper model with graph theoretic model [5]. a) Paper-Researcher UML class model. b) Graph theoretic representation of Paper-Researcher model.**

III. ANALYSIS OF CYCLES IN GRAPH-THEORETIC REPRESENTATION OF UML CLASS MODEL

The graph-theoretical representation of the UML class model has many cycles in which only a few cycles are significant for verification of finite satisfiability. The Figure 3a shows the classical benchmark "Paper-Researcher" and Figure 3b shows the graph-theoretical representation of it. Graph-theoretical representation of "Paper-Researcher" has 32 cycles and for each vertex, it has 8 cycles. Table 1 shows the complete 32 cycles and Figure 4 shows 8 cycles from Researcher vertex. It can be clearly seen in Figure 4 that the majority of cycles are not important due to they are balance or greater.

Definition 1 (Balance Cycle): The balance cycle is a cycle where the weight formula contains same quantities for the

TABLE 1. Total cycles of graph-theoretical representation of "paper-researcher" benchmark.

S#	Cycles	Weight
	From Researcher	
1	Researcher → Review → Researcher	1
2	Researcher → Write → Researcher	1
3	Researcher → Review → Paper → Review → Researcher	1
4	Researcher → Write → Paper → Write → Researcher	2
5	**Researcher → Rw → Pr → Wr → Researcher**	**.66**
6	Researcher → Write → Paper → Review → Researcher	3
7	Researcher → Review → Paper → Write → Paper → Review → Researcher	1.98
8	Researcher → Write → Paper → Review → Paper → Write → Researcher	2
	From Paper	
9	Paper → Review → Paper	1
10	Paper → Write → Paper	2
11	Paper → Review → Researcher → Review → Paper	1
12	Paper → Write → Researcher → Write → Paper	2
13	Paper → Review → Researcher → Write → Paper	3
14	**Paper → Write → Researcher → Review → Paper**	**.66**
15	Paper → Review → Researcher → Write → Researcher → Review → Paper	1.98
16	Paper → Write → Researcher → Review → Researcher → Write → Paper	2
	From Writer	
17	Write → Paper → Write	2
18	Write → Researcher → Write	1
19	Write → Paper → Review → Paper → Write	2
20	Write → Researcher → Review → Researcher → Write	1
21	Write → Paper → Review → Researcher → Write	3
22	**Write → Researcher → Review → Paper → Write**	**.66**
23	Write → Paper → Review → Researcher → Review → Paper → Write	2
24	Write → Researcher → Review → Paper → Review → Researcher → Write	1.98
	From Reviewer	
25	Review → Researcher → Review	1
26	Review → Paper → Review	3
27	Review → Researcher → Write → Researcher → Review	1
28	Review → Paper → Write → Paper → Review	1
29	Review → Researcher → Write → Paper → Review	3
30	**Review → Paper → Write → Researcher → Review**	**.66**
31	Review → Paper → Write → Researcher → Write → Paper → Review	1
32	Review → Researcher → Write → Paper → Write → Researcher → Review	1

division and multiplication due to the same Min and Max cardinalities. Therefore, it always produces 1.

Example 1: The cycle 2 in Figure 4 is a balance cycle because the weight from *Researcher* to *Review* is 1, *Review* to *Paper* is $\frac{1}{3}$, *Paper* to *Review* is 3, and *Review* to *Researcher* is $\frac{1}{1}$. The weight calculation formula of the cycle 2 will be $\left\{1 * \left(\frac{1}{3}\right) * 3 * \left(\frac{1}{1}\right)\right\}$.

Definition 2 (Greater Cycle): The greater cycle is a cycle where the weight formula uses a smaller quantity for the division and a larger quantity for multiplication and produces a value greater than 1.

Example 2: The cycle 6 in Figure 4 is a greater cycle because the weight from *Researcher* to *Write* is 1, *Write* to *paper* is $\frac{1}{1}$, *Paper* to *Write* is 2 and *Write* to *Researcher* is $\frac{1}{1}$ and the weight calculation formula of the cycle 6 will be $\left\{1 * \left(\frac{1}{1}\right) * 2 * \left(\frac{1}{1}\right)\right\}$.

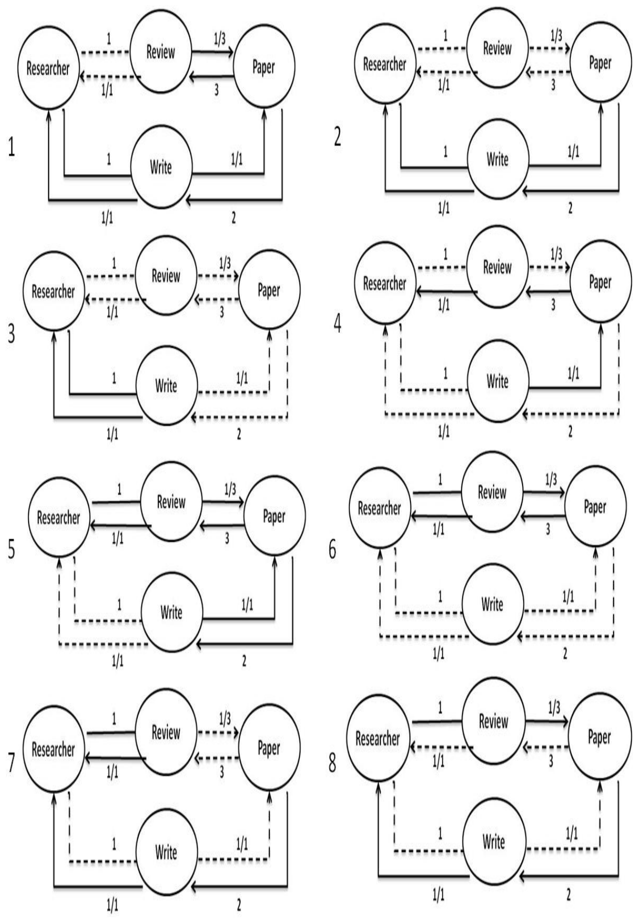


FIGURE 4. Cycles from node researcher (Rr).

The graph-theoretical representation of class association cardinality constraints has many balance and greater cycles. Balance and greater cycles are not important to determine finite-satisfiability therefore, they should not be traversed. The running example has 32 cycles in which 28 cycles are balance or greater and only 4 cycles are significant as shown in Table 1. Furthermore, if one critical cycle is detected in the graph, then it will be enough to prove that the UML class model is not finitely satisfiable. As per the above analysis, we propose an ontology-based method which reduces the search space for finding a critical cycle in the ontology-based representation of the UML class model.

IV. ONTOLOGY-BASED FINITE SATISFIABILITY

Ontology is also a graph-theoretic structure which has concepts (like vertices of the graph), relationships (like directed edges in the graph). The proposed work uses an ontology-based algorithm for finding critical cycles in the graph-theoretical representation of the UML class model. Figure 5 shows, how an ontology graph will be built and how classes, association, and cardinalities will be represented in the ontology. In the proposed method, weight distribution formula given by Hartmann is distributed among object

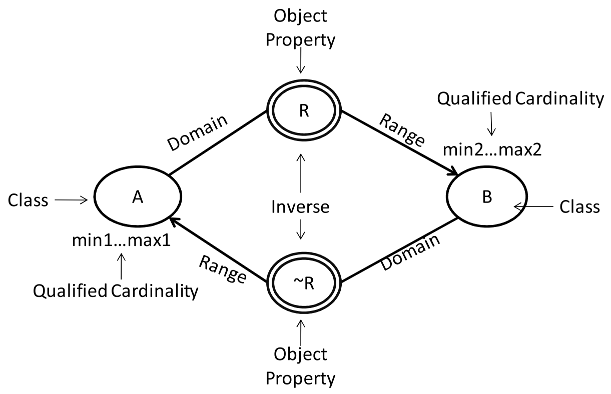


FIGURE 5. Ontology-based graph-theoretic of UML class diagram.

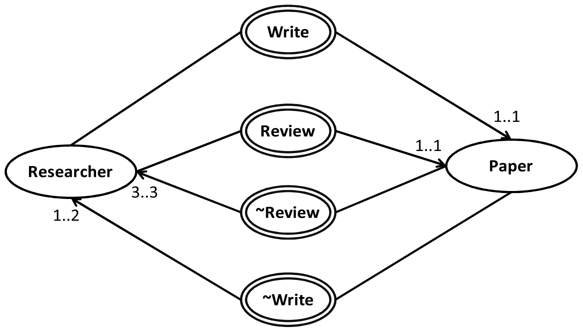


FIGURE 6. Ontology-based representation of "paper and researcher" benchmark.

properties and inverse properties as shown in figure 5.

In this work, UML classes are transformed into ontology's classes and associations into object properties. An inverse object property is also added for every association. The association cardinalities are transformed into ontology's qualified cardinalities. Figure 6 shows, the ontology graph of the Paper-Researcher class model. The proposed method greatly concerned about that, there are few critical cycles among many which cause unsatisfiability as discussed in the previous section.

In the proposed method, the ontology graph is only traversed by the object properties. The traversing method starts from the selection of an arbitrary object property and moves forward to the successor adjacent object property until the successor adjacent object property range is equal to the domain of initial object property (where the traversing was started).

Definition 3 (Adjacent Object Property): Object properties P_1 and P_2 are considered adjacent object property if and only if the P_1range is equal to the P_2domain (except the inverse property of $P_1$for avoiding unnecessary cycles discussed in section IV).

Example 3 in "Paper-Researcher" *Write* and ~*Review* are adjacent as shown in Figure 6. However, *Write* and ~*Write* are not considered adjacent even their range and domain are equal since they are inverse to each other and if it is considered as adjacent then balance and greater cycles will be traversed. Algorithm 1 describes the step-by-step ontology-based method for detecting critical cycles.

Algorithm 1 Ontology-Based Algorithm for Detecting Critical Cycle

```
1. $p \leftarrow$ Arbitrary Property
2. $s \leftarrow p$. Domain
3. $w \leftarrow 1$
4. While $s \neq p$. Range
   $\sim p \leftarrow$ Inverse ($p$)
   $w \leftarrow w * (p.\text{MaxRange}*(1/\sim p.\text{MinRange}))$
     $ap \leftarrow$ SearchProperties()
     (Where $ap$.Domain $= p$.Range and $ap \neq \sim p$)
     $p \leftarrow ap$
   End While
5. $\sim p \leftarrow$ Inverse ($p$)
6. $w \leftarrow w * (p.\text{MaxRange}*(1/\sim p.\text{MinRange}))$
```

In the algorithm 1, ontology graph is traversed by each object property only once. The running example "Paper-Researcher" has four object properties *Write, ~Write, Review,* and ~*Review*. For algorithm demonstration, we start from object property *Review*. The following steps are repeated until successor adjacent object property range is not equal to the domain of initial property.

1) Start from object property *Review* and it's become selected property.
2) Select the inverse property of selected property (~*Review*).
3) After getting the inverse property, the weight of the path is calculated and is added to the total weight. According to running example, the *MaxCradinality* of *Review* is 1 and the *MinCardinality* of ~*Review* is 3 then the weight of current path will be .33 and it will be added to the total weight.
4) Then adjacent properties of the selected property will be searched and according to running example, two properties will return as successor adjacent object property ~*Write* and ~*Reviews*. ~*Review* is the inverse property of selected property (*Review*). Therefore, it will be ignored and ~*Write* will become a selected property. The *range* of ~*Write* is *Researcher* which is equal to the *domain* of *Review* (the initial property). Therefore the algorithm finishes here (when the successor adjacent property *range* will be equal to the *domain* of initial property).

Table 2 shows the complete iterations of the presented algorithm for object property *Review*. The running example has four object properties; therefore only four cycles will be traversed for verification of finite satisfiability. However, in the four cycles, some cycles are calculating the same weight due to traversing the same node but their order of calculation is different.

Table 3 shows the four unique cycles which are important for verification of finite satisfiability. If the table 3 is keenly

TABLE 2. Iterations of algorithm 1 for object property *review*.

ST	PR	DO	RA	MC	IP	1/M	WT	TW
Researcher	Review	Researcher	Paper	1	~Review	1/3=1	.33	.33
	~Write	Paper	Researcher	2	Write	1/1=1	2	.66

ST = start,PR = Property, DO = Domain; RA =Range, MC = Max Cardinality, IP = Inverse Property, 1/M = 1/Min, WT = Weight, TW = Total Weight

TABLE 3. Significant cycles in "paper and researcher" example.

Object Property	Path and Weight	Weight
Write	Researh → Write → Paper → ~Revieww → Researchr	$\left\{1*\left(\frac{1}{2}\right)*3*\left(\frac{1}{1}\right)\right\}$=1.5
~Write	Paper → ~Write → Researcher → Review → Paper	$\left\{2*\left(\frac{1}{1}\right)*\left(\frac{1}{1}\right)*\left(\frac{1}{3}\right)\right\}$ =.66
Review	Researcher → Review → Paper → ~Write → Researcher	$\left\{1*\left(\frac{1}{3}\right)*2*\left(\frac{1}{1}\right)\right\}$ =.66
~Review	Paper → ~Review → Researcher → ~Write → Paper	$\left\{3*\left(\frac{1}{1}\right)*1*\left(\frac{1}{2}\right)\right\}$=1.5

analyzed then it can be observed that *Write* and ∼*Review*, and *Review* and ∼*Write* traverse the same path and calculate the same weight. The only difference is their order of multiplication as shown in the last column of table 3. Therefore, if one object property and its inverse property are traversed then it will be enough for checking all unique cycles in the graph and there is no need of traversing the ontology graph by other object properties.

V. COMPLEX GRAPH

Algorithm 1 works fine on the simple graph where search property procedure only returns two adjacent object properties, (1) the inverse of the selected property and (2) other than inverse property. However, when the procedure returns two or more than two adjacent object properties except the inverse property then the algorithm can procced only with one of them and the path starts from the other adjacent object properties are missed. For example, Figure 7a shows a complex class model and Figure 7b shows equivalent ontology graph. If the selected property is *R1* then adjacent object properties will be ∼*R1*, *R4*, and *R2*. The ∼*R1* is the inverse of *R1* and it will be ignored but anyone from *R4* or *R2* will be selected and traversed according to algorithm 1. Therefore, the paths which lead by other property will be missed. Algorithm 2 introduces recursiveness in the algorithm 1 for dealing with the limitation mentioned above.

VI. COMPLEXITY

Table 4 shows the complexity of the proposed method and presents the comparison with Bellman-Ford. The complexity of Bellman-Fords is Θ (|V||E|) from the single vertex and from all vertex or for complete graph the complexity is Θ ($|V^2||E|$). The complex UML class diagram such as a shown in Figure 7 a. will be transformed into the complete graph and there is a chance to miss negative cycle from a single vertex. Therefore we consider both complexities for comparison.

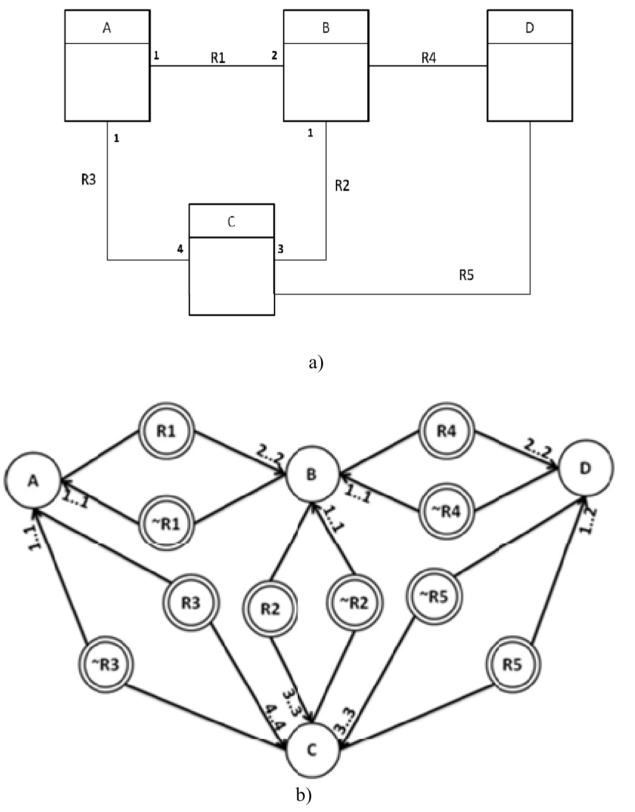


FIGURE 7. Class model with complex association. a) UML class model with complex associations. b) Graph theoretic representation of complex associations model.

Algorithm 2 Ontology-Based Recursive Algorithm for Detecting Critical Cycle

```
FindCycle($s, r, p, 1$)
1.  $visited[p] \leftarrow$ true
2.  $\sim p \leftarrow$ Inverse ($p$)
3.  **if** $s = r$ **then**
     **return** $w$
4.  **else**
     $ap \leftarrow$ SearchProperties ()
       (Where $ap$.Domain $= p$.Range **and** $ap \neq \sim p$)
       **foreach** $i \in$ Ap **do**
       $w \leftarrow p.Range.MaxRange*(1/\sim p.Range.MinRange)$
       **if** visited ($p$) $\neq$ true **then**
          FindCycle($s, i.Range, Class, i.Property, weight$)
5. $visited[p] \leftarrow$ false
```

We consider single vertex complexity as a best case and all vertex as a worst case. For the proposed algorithm best and worst case are same. The last fours columns of table 4 also show the number of paths will be traversed by each algorithm in the best case and worst case for "Paper-Researcher" and "Complex Association".

VII. SOUNDNESS AND COMPLETENESS

Theorem 1 (Soundness and Completeness): Let *G* be an ontology graph and suppose that proposed algorithm run on *G* from a given property *P* and its inverse property ∼*P* then, during

TABLE 4. Comparison bellman-ford and proposed algorithm.

Methods	Complexity (Single Vertex)	Complexity (All Vertex)	Paper-Researcher		Complex Associations	
			Best case	Worst case	Best case	Worst case
Bellman Ford	Θ (|V||E|)	Θ (|V²||E|)	16	64	180	810
Ontology Based (Proposed)	Θ (CO)	Θ (CO)	8	8	40	40

its execution, proposed algorithm discovers critical cycles in the ontology-based graph of UML class model.

Proof: utmost (n^{n-1}) object properties (in which half will be inverse) will be required when each class connect to other class by a single association in the ontology-based graph of UML class model. Therefore, there should be $((n^2 - n)/n)$ cycles which will unique and enough for processing all weight and detection of critical cycles.

- Let suppose we have $C = \{c_1, c_2, c_3\}$ for the classes and $P = \{p_1, p_2, p_3, p_4, p_6\}$ for object properties where p_4, p_5, p_6 are inverse object properties of p_1, p_2, p_3. The graph connectivity is as follows:

$$G = \{p_1(c_1, c_2), p_2(c_2, c_3), p_3(c_1, c_3), p_4(c_2, c_1), p_5(c_3, c_2), p_6(c_3, c_1)\}$$

- Let suppose, we traverse graph from p_1 and it inverse p_4 then we found the following subset of G:
  $p_1 = \{(c_1, c_2), (c_2, c_3), (c_3, c_1)\}$ and access the following adjacent object properties:
  $A = \{p_1, p_2, p_3\}$
  $p_4 = \{(c_2, c_1), (c_1, c_3), (c_3, c_2)\}$ and access the following adjacent object properties:
  $A^I = \{p_4, p_5, p_6\}$
- $A \cap A^I = \emptyset$, all cycles are unique.
- $A \cup A^I = P$, all object properties have been traversed and all weights have been calculated.
- For other object properties, the path will remain same just their order of traversing and order of multiplication will be different. According to the Associative law, in multiplication order does not matter. □

The discussion in section V has shown that algorithm 1, work with simple ontology graph which returns only two adjacent properties one is the inverse of the selected property and second is other than inverse property.

Algorithm 1 is thus incomplete; that is, it does not guarantee the traversing of all valid paths. The algorithm 2 which is improved version of algorithm 1 work with complex graph and guarantee the traversing of all valid paths and determines the existence of a critical cycle in the ontology graph. For example Figure 8 shows the recursive call of algorithm 2 for property R1. There are two paths move forward from R1 in which one path goes to R2 and other goes to R4. From R2, further two paths move forward, one path goes to R3 which completes the cycle and other path goes to R5. From R5, the path moves forward to R4 which completes the cycle at R1. Here all unique cycles which start from R2 complete their cycle. On the other side, R4 goes to R5 and R5 further move forward to R3 and R2. On R3 another cycle is completed and R2 further goes to R1 and complete the cycle.

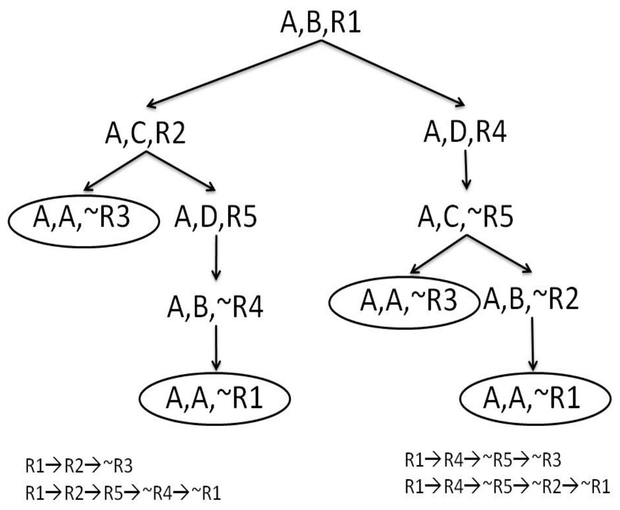


FIGURE 8. Recursive Call of Algorithm 2 from *R1*.

VIII. EMPIRICAL EVALUATION

The objective of this evaluation is to investigate the effectiveness and correctness of the proposed ontology-based verification method. Especially, the evaluation aims to answer the following questions:

— RQ1. Is it possible for a proposed approach to scale well at a practical extent?

— RQ2. How does the proposed approach compare to other UML class diagram verification methods?

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

A. EXPERIMENT SETUP AND SUBJECT OF THE STUDY

We implemented our approach as a Java prototype 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 RDF, and OWL graph. Jena provides a layer of abstraction for transforming statement into java artifacts. The abstraction layer reduces the effects required for accessing and managing ontology. It has excellent documentation which includes powerful resources such as description of artifacts and tutorial. It also supports all variant of OWL: Full, Description Logic and Lite. The input UML class model of the prototype tool is represented in XMI (XML Metadata Interchange). XMI is an OMG standard for exchanging metadata information by Extendable Markup language (XML). Specifically, the XMI is intended to provide a common format for UML diagrams for sharing among different CASE tools and it is supported by the majority of modern UML CASE Tools. So the input models of our tool must be specified in XMI version 2.4.1. The version 2.4.1 of XMI is backward compatible. However, it includes some extra elements for supporting new features of UML version 2.

As the subject of our study, we considered 5 UML class diagrams. The list and brief detail of class models used in the evaluation are as follows:

Paper-Researcher: The Paper-Researcher class model describes the concepts of article writing and reviewing process. This model took from class model benchmark paper [5].

Coach: The coach model describes the information of Bus Company Ticking system, and this model took from class model slicing paper [3].

Point of Sale (POS): The POS describe the set of classes and associations of point of sale and took from [42]

Script 1: The script 1 model is a programmatically generated model, for verification of large model.

Script 2: It is also a programmatically generated model but has more elements.

We used the bellman-ford algorithm for comparison with the proposed method because of the bellman-ford also a graph-based algorithm and used for detecting critical cycles with a negative weight. Various existing methods have been used it with little modification for verification of finite satisfiability of UML class diagram association constraint. Furthermore, there are many other verification methods which used different techniques for verification of UML class model with different features. We used UMLtoCSP and Alloy for comparison 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 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.

B. RQ1: IS IT POSSIBLE FOR A PROPOSED APPROACH TO SCALE WELL AT A PRACTICAL EXTENT?

1) MEASUREMENTS AND SETUP

The proposed approach should be fast enough and scale effectively as classes and associations increase and minimized the verification time as compared to the bellman-ford algorithm. For this reason, to respond to RQ1, we applied the proposed method to various class models as shown in Table 5. The size of these models ranges from 4 to 2,000 classes and associations. For example, Paper-Researcher only has two classes and two associations and script 2 has 1000 classes and 1000 associations. For each UML class model, we measured the verification time with the proposed approach and bellman-ford approach. Especially, we analyzed the relationship between execution time and model size for each model.

TABLE 5. Size of class models used in the evaluation.

Class Model Name	Classes	Associations
Paper-Researcher	2	4
Coach	10	10
Point Of Sale	11	15
Script1	100	100
Script2	1000	1000

TABLE 6. Description of experimental results.

Method	Paper-Researcher	Coach	POS	Script1	Script 2
Bellman	0.0009s	0.00212s	0.00315s	0.0413s	0.354s
UML2ONTO	0.00006s	0.00014s	0.00027s	0.00261s	0.034s
Improvement	92.46%	93.01%	91.40%	93.50%	90.40%

2) RESULTS

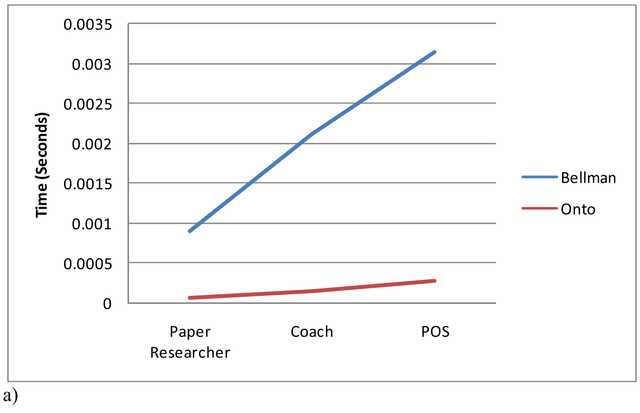
Table 6 and Figure 9 shows the average execution time (in seconds) required to verify the UML association constraints through proposed method and bellman-ford. Table 6 and Figure 9 also shows that the approach scales to a practical extent. For first model Paper-Researcher, the proposed approach requires on average 0.00006 seconds with 2 classes and 2 associations and bellman-ford takes 0.0009 seconds. In the case of Coach model which have 10 classes and 10 associations, the proposed approach requires on average 0.00014 seconds and bellman-ford takes 0.00212 seconds. In the case of POS model which have 11 classes and 15 associations, the proposed approach required on average 0.00027 seconds and bellman takes 0.00315. In the case of script 1 which has 100 classes and 100 associations the proposed approach takes on average 0.00261 seconds and bellman-ford takes 0.041. Finally, for checking the performance of the proposed method on a large model experiment performed on Script II which has 1000 classes, 1000 associations the proposed approach takes on average 0.034 seconds and bellman-ford takes 0.354 seconds.

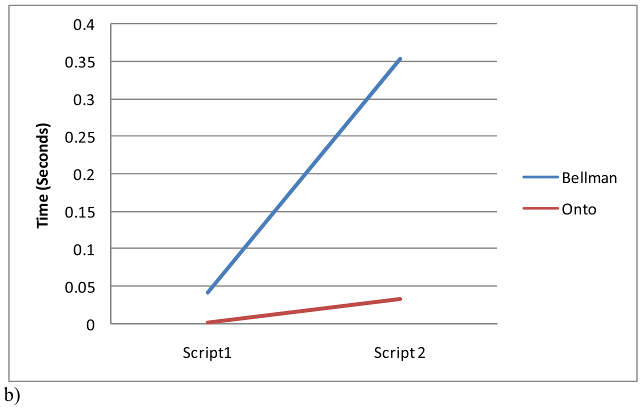
A model containing thousands of classes and associations are particularly complex to verify and there is very less chance that a single model contains a number of elements like Script II, which highlighted the scalability and efficiency of our approach. It can also be observed that the proposed method is efficient than the bellman-ford in the verification of UML class model association cardinality constraints. The last row of Table 5 also shows the improvement achieved by the proposed method in percentage which is approximately 90%.

C. RQ 2: HOW DOES THE PROPOSED APPROACH COMPARE TO OTHER VERIFICATION METHODS?

1) MEASUREMENTS AND SETUP

To respond to RQ 2, we thus compared the performance of the approach proposed with UMLtoCSP and UML2Alloy. In the comparison, we used all five UML class models (1)





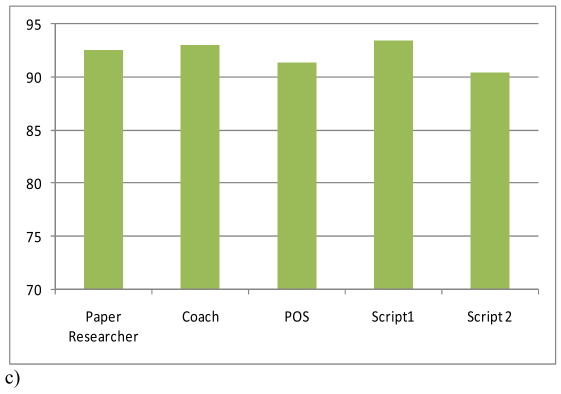


FIGURE 9. Analyzing UML class model benchmarks. a) Analyzing Paper-Researcher, Coach, and POS UML class model. b) Analyzing Script 1 and Script 2 UML class model. c) Improvement achieved by the proposed method.

Paper-Researcher (2) Point of Sale (3) Coach (4) Script I and (5) Script II. All Experiments are conducted on Intel Core 2 Duo Processor 1.3 GHz with 2GB of RAM. All times are measured in seconds and a time-out limit has been set at 2 hours (7200 seconds).

2) RESULTS

Figure 10 and Table 7 show the comparison results between existing methods (UMLtoCSP and UMLtoAlloy) and the proposed method. The x-axis of Figure 10 reports models and the y-axis reports the execution time taken in a second. As the results show proposed approach is efficient from the existing methods. The comparison also shows that the for the small model the verification time of proposed method and other method have a small difference but when the size of model increase the proposed methods is more efficient than the existing methods. Furthermore, in some cases, the exiting tools unable to verify the complex and large model such as in the evaluation the UMLtoCSP tool found unable to verify Script II as shown in Table 7.

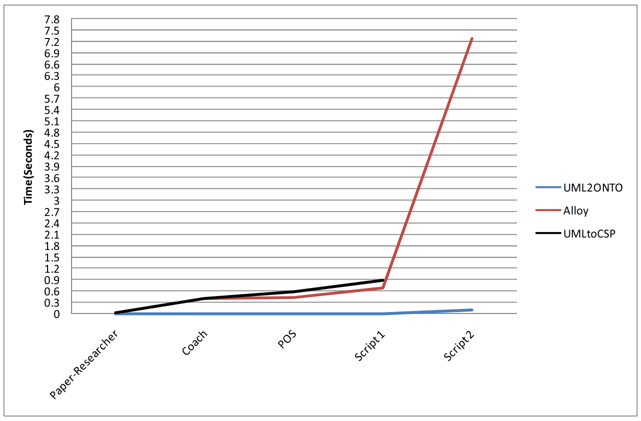


FIGURE 10. Analyzing UML class model benchmarks.

TABLE 7. Description of experimental results.

Tools/Models	Paper-Researcher	Cocah	POS	Script 1	Script 2
UML2ONTO	0.00098s	0.0019s	0.003s	0.00396	0.090s
Alloy	0.023s	0.41s	0.42s	0.68s	7.28s
UMLtoCSP	0.022s	0.41s	0.58s	0.89s	Timeout

IX. CONCLUSION AND FUTURE WORK

This paper presents an innovative ontology-based method for verification of finite satisfiability of UML class model to reduce the verification time in order to improve the efficiency. In this method efficiency of the verification process has been achieved by a reduction of search space. This work also presented an analysis of the graph-theoretic representation of the UML class model. In the analysis, it has been identified that the graph-theoretic representation of the UML class model has many cycles which are balance or greater and they do not impact on verification. Therefore, they should not be considered as a part of the search space. Furthermore, we demonstrated the proposed method on real-world case studies such as Paper-Researcher, Coach, and POS to analyze the improvement and also applied the proposed method on programmatically generated large models which have thousands of classes and associations in order to demonstrate that, the proposed method efficiently support large and complex model.

As our future work, we plan to explore two research directions. Firstly, we plan to investigate more efficient method which more speeds up the verification. Secondly, provide the support of other UML class model constraints and OCL constraints.

REFERENCES

[1] D. Cuadra, P. Martínez, E. Castro, and H. Al-Jumaily, "Guidelines for representing complex cardinality constraints in binary and ternary relationships," *Softw. Syst. Model.*, vol. 12, no. 4, pp. 871–889, Oct. 2013.

[2] R. Van Der Straeten, "Inconsistency management in model-driven engineering an approach using description logics," Ph.D. dissertation, Dept. Comput. Sci., Vrije Univ. Brussel, Brussels, Belgium, 2005.

[3] A. Shaikh and U. K. Wiil, "Efficient verification-driven slicing of UML/OCL class diagrams," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 5, pp. 530–547, 2016.

[4] A. Shaikh and U. K. Wiil, "A feedback technique for unsatisfiable UML/OCL class diagrams," *Softw., Pract. Exper.*, vol. 44, no. 11, pp. 1379–1393, 2014.

[5] A. Shaikh, U. K. Wiil, and N. Memon, "Evaluation of tools and slicing techniques for efficient verification of UML/OCL class diagrams," *Adv. Softw. Eng.*, vol. 2011, Jun. 2011, Art. no. 370198.

[6] J. Cabot and R. Clarisó, "UML-OCL verification in practice," in *Proc. MoDELS Workshops*, vol. 5421. 2008, pp. 31–35.

[7] N. Przigoda, J. G. Filho, P. Niemann, R. Wille, and R. Drechsler, "Frame conditions in symbolic representations of UML/OCL models," in *Proc. ACM/IEEE Int. Conf. Formal Methods Models Syst. Design (MEMOCODE)*, Nov. 2016, pp. 65–70.

[8] M. Cadoli, D. Calvanese, G. De Giacomo, and T. Mancini, "Finite satisfiability of UML class diagrams by constraint programming," in *Proc. Workshop CSP Techn. Immediate Appl.*, Sep. 2004, pp. 1–17.

[9] H. Malgouyres and G. Motet, "A UML model consistency verification approach based on meta-modeling formalization," in *Proc. ACM Symp. Appl. Comput.*, 2006, pp. 1804–1809.

[10] M. Balaban and A. Maraee, "Finite satisfiability of UML class diagrams with constrained class hierarchy," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, 2013, Art. no. 24.

[11] A. Artale, D. Calvanese, and A. Ibáñez-García, "Full Satisfiability of UML class diagrams," in *Proc. 29th Int. Conf. Conceptual Model.*, Vancouver, BC, Canada, Nov. 2010, pp. 317–331.

[12] A. Maraee, V. Makarenkov, and B. Balaban, "Efficient recognition and detection of finite satisfiability problems in UML class diagram," in *Proc. Int. Workshop Model Co-Evol. Consistency Manage. (MoDELS)*, Toulouse, France, 2008.

[13] A. Maraee and M. Balaban, "Efficient recognition of finite satisfiability in UML class diagrams: Strengthening by propagation of disjoint constraints," in *Proc. Int. Conf. Model-Based Syst. Eng. (MBSE)*, 2009 pp. 1–8.

[14] M. Balaban, A. Maraee, A. Sturm, and P. Jelnov, "A pattern-based approach for improving model quality," *Softw. Syst. Model.*, vol. 14, no. 4, pp. 1527–1555, 2015.

[15] H. Ledang and J. Souquières, "Integration of UML and B specification techniques: Systematic transformation from OCL expressions into B," in *Proc. Asia–Pacific Softw. Eng. Conf. (APSEC)*, 2002, pp. 495–504.

[16] N.-T. Truong and J. Souquieres, "An approach for the verification of UML models using B," in *Proc. Int. Conf. Workshop Eng. Comput.-Based Syst.*, Brno, Czech Republic, 2004, pp. 195–202.

[17] M. Balaban, A. Maraee, and A. Sturm, "Management of correctness problems in UML class diagrams towards a pattern-based approach," *Int. J. Inf. Syst. Model. Des.*, vol. 1, no. 4, pp. 24–47, 2010.

[18] S. Hartmann, "Coping with inconsistent constraint specifications," in *Proc. ER*, vol. 2224. 2001, pp. 241–255.

[19] M. Lenzerini and P. Nobili, "On the satisfiability of dependency constraints in entity-relationship schemata," *Inf. Syst.*, vol. 15, no. 4, pp. 453–461, 1990.

[20] B. Thalheim, *Fundamentals of Entity-Relationship Modeling*. New York, NY, USA: Springer, 2000.

[21] D. Calvanese and M. Lenzerini, "On the interaction between ISA and cardinality constraints," in *Proc. 10th Int. Conf. Data Eng.*, 1994, pp. 204–213.

[22] R. Clarisó, C. A. González, and J. Cabot, "Towards domain refinement for UML/OCL bounded verification," in *Software Engineering and Formal Methods* (Lecture Notes in Computer Science), vol. 9276. U.K.: Springer, 2015, pp. 108–114.

[23] A. Queralt and E. Teniente, "Reasoning on UML class diagrams with OCL constraints," in *Conceptual Modeling—ER*, vol. 4215. Tucson, AZ, USA: Springer, 2006, pp. 497–512.

[24] A. Formica, "Finite satisfiability of integrity constraints in object-oriented database schemas," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 1, pp. 123–139, Jan. 2002.

[25] N. Mahmud, "Ontology-based analysis and scalable model checking of embedded systems models," Ph.D. dissertation, Dept. Comput. Sci. Eng., Mälardalen Univ., Västerås, Sweden, 2017.

[26] T. H. Nguyen, J. C. Grundy, and M. Almorsy, "Ontology-based automated support for goal–use case model analysis," *Softw. Qual. J.*, vol. 24, no. 3, pp. 635–673, 2016.

[27] C. Corea and P. Delfmann, "Detecting compliance with business rules in ontology-based process modeling," in *Proc. Int. Tagung Wirtschaftsinformatik (WI)*, 2017, pp. 226–240.

[28] Y. Liao, H. Panetto, J. M. Simão, and P. C. Stadzisz, "Ontology-based model-driven patterns for notification-oriented data-intensive enterprise information systems," in *Proc. 7th Int. Conf. Inf. Soc. Technol. (ICIST)*, 2017, pp. 148–153.

[29] M. Fellmann, F. Hogrebe, O. Thomas, and M. Nüttgens, "An ontology-driven approach to support semantic verification in business process modeling," in *Proc. MobIS*, 2010, pp. 99–110.

[30] J. Sun, H. H. Wang, and T. Hu, "Design software architecture models using ontology," in *Proc. SEKE*, 2011, pp. 191–196.

[31] K. Mokos, G. Meditskos, P. Katsaros, N. Bassiliades, and V. Vasiliades, "Ontology-based model driven engineering for safety verification," in *Proc. 36th EUROMICRO Conf. Softw. Eng. Adv. Appl. (SEAA)*, 2010, pp. 47–54.

[32] M. Kezadri and M. Pantel, "First steps toward a verification and validation ontology," in *Proc. KEOD*, 2010, pp. 440–444.

[33] A. Lapets, P. Lalwani, and A. Kfoury, "Ontology support for a lightweight formal verification system," Dept. CS, Boston Univ., Tech. Rep. BUCS-TR-2010-012, 2010.

[34] H. He, Z. Wang, Q. Dong, W. Zhang, and W. Zhu, "Ontology-based semantic verification for UML behavioral models," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 2, pp. 117–145, 2013.

[35] W. Xu, A. Dilo, S. Zlatanova, and P. van Oosterom, "Modelling emergency response processes: Comparative study on OWL and UML," in *Proc. Inf. Syst. Crisis Response Manage. Harbin Eng. Univ.*, 2008, pp. 493–504.

[36] M. Bahaj and J. Bakkas, "Automatic conversion method of class diagrams to ontologies maintaining their semantic features," *Int. J. Soft Comput. Eng.*, vol. 2, no. 6, pp. 65–69, 2013.

[37] A. Belghiat and M. Bourahla, "From UML class diagrams to OWL ontologies: A graph transformation based approach," in *Proc. ICWIT*, 2012, pp. 330–335.

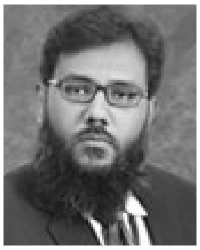
[38] F. S. Parreiras and S. Staab, "Using ontologies with UML class-based modeling: The TwoUse approach," *Data Knowl. Eng.*, vol. 69, no. 11, pp. 1194–1207, 2010.

[39] G. Guizzardi, H. Herre, and G. Wagner, "Towards ontological foundations for UML conceptual models," in *On the Move to Meaningful Internet Systems: CoopIS, DOA, and ODBASE*. Irvine, CA, USA: Springer, 2002, pp. 1100–1117.

[40] G. Guizzardi, H. Herre, and G. Wagner, "On the general ontological foundations of conceptual modeling," in *Conceptual Modeling—ER* (Lecture Notes in Computer Science), vol. 2503. Tampere, Finland: Springer, 2002, pp. 65–78.

[41] A. B. Benevides and G. Guizzardi, "A model-based tool for conceptual modeling and domain ontology engineering in OntoUML," in *Enterprise Information Systems* (Lecture Notes in Business Information Processing), vol. 24. Milan, Italy: Springer, 2009, pp. 528–538.

[42] C. Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development*. New Delhi, India: Pearson Education, 2012.



ABDUL HAFEEZ KHAN received the M.S. degree in software engineering from Hamdard University, where he is currently pursuing the Ph.D. degree in computer science. His current research interests include software engineering, model verification, and ontology-based software.

SAYED HYDER 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. Previously, he was engaged as the Chairman of the Department of Electrical and Electronics Engineering with Hamdard University, Karachi. In past, he has served as a Professor and the Principal at Petroman–an Institute of Ministry of Information Technology and Telecommunications, Government of Pakistan at its various campuses for over ten years and had also remained Executive District Officer IT (EDO-IT) District Government, Larkana. To his credit are over 30 research publications in national and international journals. He has attended numerous international conferences as an invited speaker. 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. He is associated with Hamdard University since last 18 years. He has over 19 years of teaching, research, and academic administration experience. He is the author of 47 research articles in journals, conferences, and book chapters of international repute. His current research interests include sensor networks, ubiquitous computing, computer networks, and smart agriculture.

ASADULLAH SHAIKH received the B.Sc. degree in software development from the University of Huddersfield, England, the M.Sc. degree in software engineering from the University of Gothenburg, Sweden, and the Ph.D. degree in software engineering from the University of Southern Denmark, Denmark. He is currently an Assistant Professor, the Head of research, and the Coordinator of seminars and training at the College of Computer Science and Information Systems, Najran University, Najran, Saudi Arabia. He has vast teaching and research experience. He has written and published more than 70 research papers in top class conferences and impact factor journals. He is also an editorial board member and a reviewer of several national and international conferences/journals. He is also a member of Technical Committee of Software Engineering of the IEEE Computer Society. Previously, he was a member of the Software Engineering Group with the Universitat Oberta de Catalunya, Barcelona, Spain.