IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 26, 2021, accepted October 13, 2021, date of publication October 18, 2021, date of current version October 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3121222

More Than Two Decades of Research on Verification of UML Class Models: A Systematic Literature Review

ASADULLAH SHAIKH[1], ABDUL HAFEEZ[2], ASIF ALI WAGAN[2], MESFER ALRIZQ[1], ABDULLAH ALGHAMDI[1], AND MANA SALEH AL RESHAN[1]

[1]College of Computer Science and Information Systems, Najran University, Najran 61441, Saudi Arabia
[2]Department of Software Engineering, SMI University, Karachi 75190, Pakistan

Corresponding author: Abdul Hafeez (ahkhan@smiu.edu.pk)

This work was supported by the Ministry of Education and the Deanship of Scientific Research, Najran University, Saudi Arabia, under Grant NU/-/SERC/10/521.

ABSTRACT Error checking is easy and inexpensive in the initial stages as compared to later stages due to when the development cycle precedes the development cost and efforts also increase. UML class model is a key element of modern software methodologies and creates in the initial stage of software development. Therefore, error detection and rectification of the UML class model may save software development costs and time. This paper presents an overview of UML Class model verification approaches and identifies open issues, current research trends, and other improvement areas. This study uses a systematic literature review as an investigation method with six research questions and assesses 65 papers dated January 1997 to December 2020. From 2124 published research papers, 65 papers are selected and distributed into 7 studies. This work provides an analysis of verification approaches and the automation level of proposed approaches. As a result, it is found that the existing UML class model verification methods provide great efforts to check correctness. However, in some situations (when dealing with large and complex models), they consume a significant amount of time and do not support many important features of the UML class model.

INDEX TERMS Class model, UML, model formalisation, model verification, UML-OCL models.

I. INTRODUCTION

Model Derive Engineering (MDE) is a software development methodology where the software model is considered the nucleus of software development activities. In the MDE, software specifications are stated through abstract models. These models are initially built platform-neutral and are known as Platform Independent Model (PIM) [1]. Then PIM models are gradually refined and transformed into the Platform Specific Model (PSM). Finally, the programming code is generated (semi)automatically by the PSM models [1]. MDE approach saves time and cost, and the industry extensively adopts it. However, the MDE approach has some limitations, such as the PIM model is developed in the preliminary phases of software development. In the early stages, the development team is not fully aware of the system and its constraints. Therefore, there is a possibility that a PIM model will be built with errors, and these errors will be move implicitly into the PSM model and code. Model verification is a promising solution to the problem.

Model verification makes sure the model is bug-free. Mainly, it checks the model's correctness and ensures that the model under consideration is consistent and satisfactory. Unified Modeling Language (UML) is an industry adopted graphical modeling language, and it is recurrently used in MDE [2], [3]. It has many models for dealing with different aspects of software [4], [5]. The class model is an essential part of UML and performs a key role in software analysis and design [6]. UML only provides graphical elements for designing models without reasoning support due to a lack of formal foundation [7], [8]. Therefore, many researchers have used various formal/semi-formal approaches to verify the UML class model, such as Z Notation, B method, Alloy, CSP. This paper reviews many research articles on UML class model verification to evaluate the progress and direct future research on this MDE problem. In [9], authors performed

The associate editor coordinating the review of this manuscript and approving it for publication was Haider Abbas.

a systematic study on verification of conceptual models in which they also discussed verification of UML class model verification. However, the study presented in [9] differs from this study in the following directions:

- The study presented in [9] focuses on verification of almost all conceptual models, including the Entity Relationship Diagram (ERD) model, domain-specific model, and UML class model. The presented study only focuses on the UML class model.
- In [9], authors also reviewed UML class model validation methods such as the USE method. In this study, the presented work only considered verification approaches.
- Authors of [9] also reviewed approaches that only support transformation and verification of OCL without a class model such as OCLtoFOL. The presented study only focuses on verification approaches that performed formalization of OCL constraints with a class model.
- They reviewed from 2002 to 2014. due to this range, authors of [9] missed initial works such as Z and B-based approaches, which provide a foundation for modern verification methods. The presented study reviewed the period from 1997 to 2020, covering a suitable range of papers that include both foundation and up-to-date verification approaches.

The rest of the paper is structured as follows. Section II presents methodology for the conducted Systematic Literature Review (SLR). Section III shows the results obtained from SLR. Discussions are explored in Section IV. Finally, Section VI provides the conclusions.

II. METHODOLOGY

This section discusses the applied methodology for the conducted SLR. The following sections, describe the process to be followed during this work, followed by stating the Research Questions (RQs), strategy regarding search and selection, then defining inclusion and exclusion criteria, and finally specified the method to follow for data extraction.

A. PROCESS

The procedure of the systematic literature review was developed according to the guidelines specified by Kitchenham and Charters. SLR has three key stages; planning, execution, and reporting [10].

1 **Planning Stage:**
   a Protocol development: Scope of research and review protocol is developed; the protocol is iteratively improving in later stages.
   b Research Questions: Research questions are prepared through the NHMR guidelines.

2 **Execution Stage:**
   a Collecting studies: Search keywords are formalized, and then, studies are collected.
   b Data analysis: extracted information from studies is examined to respond to the research questions. studies: Search keywords are formalized, and then, studies are collected.

3 **Reporting phase:** Systematic discussion and present the outcomes.

B. RESEARCH QUESTIONS

The presented work provides a comprehensive survey and analysis of formal and semi-formal verification methods. More significantly, focus on the following research questions:
RQ1: What are the importance of UML and UML class model in modeling and MDE?
RQ2: What methods or techniques have been adopted for verification of the UML class model?
RQ3: Is the verification approach is based on a metamodel?
RQ4: Which model defects have been undertaken in each method, and which defect has been examined in most cases?
RQ5: Is an automatic or semi-automatic tool developed for the verification method, and what are their strengths and limitations?

RQ1 Is designed to describe UML and UML class model role in the model-driven engineering and software industry. The expected outcome will be a comprehensive view about the use of UML and UML class models in the industry. The RQ2 is formulated to identify different formal and semi-formal methods that have been used in the verification of the UML class model. RQ3 is specially designed to find the answer that current verification methods still support the metamodel's formalization. RQ4 is designed to determine the state-of-the-art of research on different correctness properties which are verified in different methods. RQ5 is formulated to examine that the proposed approaches practically demonstrated through the software. It also helps us to identifies the limitations and strengths of the method on which the tool is based.

C. SEARCH AND SELECTION STRATEGY

This is the most critical and most important stage of SLR. It must be specified very carefully as the search for studies should ensure complete coverage of the subject under consideration. The search strategy developed in this work has 4 stages. (1) Automatically searched on the most relevant scientific digital libraries. (2) removed all duplicate studies (3) ensuing pre-set criteria of inclusion, only related studies were considered. (4) more papers were explored by snowballing. Figure 1 shows the search and selection strategy employed in this work.

1) **STAGE 1:** PERFORMING AUTOMATIC SEARCH

To obtain as many related studies as possible, an automatic search was performed on the scientific digital libraries (list of scientific libraries shown in Table 1). These scientific libraries have a massive amount of software engineering

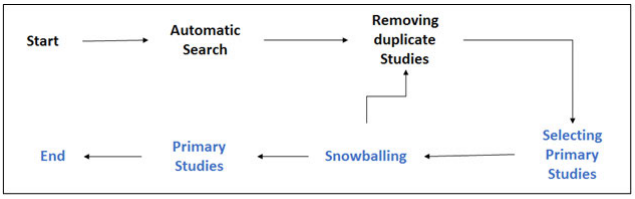

FIGURE 1. Search and selection strategy.

TABLE 1. Scientific libraries included in the search.

S.No	Scientific Digital Library	URL
01	ACM Digital Library	https://dl.acm.org/
02	IEEE Explore	https://ieeexplore.ieee.org/
03	Science Direct	https://www.sciencedirect.com/
04	Springer Link	https://link.springer.com/
05	Google Scholar	https://scholar.google.com/
06	Microsoft Academic Search	https://academic.microsoft.com/home
07	Scopus	https://www.scopus.com

TABLE 2. Keywords definition based on PICOC criteria.

Population	"UML class model" OR "UML class diagram" OR "UML Model" OR "UML/OCL"
Intervention	"verification" OR "consistency" OR "analysis" OR "reasoning" OR "satisfiability" OR "corrections" OR "checking" OR "unsatisfiable" OR "formalisation"
Comparison	Not Applicable
Outcome	Report on the current state-of-the-art method, technique, tools and challenges of UML class model verification
Context	Peer-reviewed publications

publications PICOC (**P**opulation, **I**ntervention, **C**omparison, **O**utcome, **C**ontext) criteria [10] were employed to specify the keywords as shown in Table 2, which help to build good search strings. These searching terms are used when performing searches in the above-mentioned scientific digital libraries.

The overall search string is as follows:

1) ((''verification'' OR ''consistency'' OR ''analysis'' OR ''reasoning'' OR ''satisfiability'' OR ''corrections'' OR ''checking'' OR ''unsatisfiable'' OR ''formalization'')) AND ((''UML class model'') OR (''UML class diagram'') OR (''UML Model'') OR (''UML/OCL''))
2) Scientific Digital libraries have different syntax, a specialized search string for each digital library was created. This is to make sure to include as appropriate studies as possible.

2) **STAGE 2:** REMOVING DUPLICATE STUDIES

Initially, Mendeley's reference manager was used to a kept a pool of primary studies. This repository has also helped the process of finding duplicate papers. Two or more papers are treated duplicate if and only if:

1) Study title, date of publications, location, and authors are the same. The most recent paper is kept when different versions of the same paper were found.
2) In case of paper is published in a different location, one of them is selected (the most recent).
3) In the case of a study published in a journal and conference, the journal publication is counted because the journal publication comprises the extended study and provides more information.

3) **STAGE 3:** SELECTING STUDIES

In this stage, on the inclusion and exclusion criteria, primary studies are selected. Only those studies are included which matching the criteria. The criteria were implemented by reading the article title, abstract/summary, keywords, and introduction sections. However, if this is not adequate to reach an absolute decision, other sections such as methodology and conclusion are considered.

The search strings specified in the previous section were very wide and returned papers that are not directly related to the goal of the systematic literature review. After collecting 2124 papers, the pruning stage was started, which kept out unrelated studies.

In the title pruning stage, papers title and keywords were read. After carefully read, 40 papers were excluded because they were focusing on the verification of other UML models. The remaining 82 papers were finalized for further analysis. In the last, the abstracts of the remaining papers were read, and 48 papers were excluded because they were focused on inter-model consistency. In the end, only 37 articles were selected that were directly related to the purpose of this review. The list of papers is given in Table 3. This list was complemented with papers that are found after the snowballing process.

4) **STAGE 4:** SNOWBALLING

To ensure not miss potential studies, articles that could not be automatically searched or published in the issues of scientific digital libraries were also searched. The snowballing process is done by discovering citing of the primary studies. Google Scholar is used to discover those articles. Subsequently, newly discovered and selected articles are included to the pool.

In the snowballing process, references of all selected papers were carefully read. After applying inclusion and exclusion criteria mentioned in section 4 and pruning stages, 24 more papers added to the study, as listed in Table 4. Finally, after the pruning and snowballing stages total of 61 papers were selected for the study. The year-wise distribution of selected papers is shown in Figure 2.

This work includes related work published from January 1997 to December 2020 to answer the above research questions. 61 research papers were analyzed and combined into 7 studies, as shown in Table 5. These studies are further divided into sub-studies that indicate similar approaches by different authors. However, the final study is not further divided because it is a combination of controversial confirmation approaches. In this SLR, most of the papers are part of a single research project.

D. INCLUSION AND EXCLUSION CRITERIA

In the systematic literature review, some criterion is defined for incorporating only related research works. The inclusion criteria focus only on the verification of the UML class model with or without OCL. The exclusion criteria also defined,

TABLE 3. Relevant work obtained after pruning Stage.

S#	Title	Pub. Year
01	Continuing a Benchmark for UML and OCL Design and Analysis Tools	2016
02	Efficient Verification-Driven Slicing of UML/OCL Class Diagrams	2016
03	Towards Domain Refinement for UML/OCL Bounded Verification	2015
04	On the Verification of UML/OCL Class Diagrams using Constraint Programming.	2014
05	A Feedback Technique for Unsatisfiable UML/OCL Class Diagrams	2014
06	Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy	2013
07	Initiating a Benchmark for UML and OCL Analysis Tools	2013
08	Automated Reasoning on UML Conceptual Schemas with Derived Information and Queries	2013
09	UMLtoCSP (UOST) A Tool for Efficient Verification of UML/OCL Class Diagrams Through Model Slicing	2012
10	CD2Alloy Class Diagrams Analysis Using Alloy Revisited	2011
11	Evaluation of Tools and Slicing Techniques for Efficient Verification of UML/OCL Class	2011
12	AuRUS: Automated Reasoning on UML/OCL Schemas	2010
13	UOST UML/OCL Aggressive Slicing Technique For Efficient Verification of Models	2010
14	Formalization of UML Class Diagram Using Description Logics	2010
15	Full Satisfiability of UML Class Diagrams	2010
16	Verification-Driven Slicing of UML/OCL Models	2010
17	Efficient Recognition of Finite Satisfiability in UML Class Diagrams Strengthening by Propagation of Disjoint Constraints	2009
18	Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagram	2008
19	Verification of UML/OCL Class Diagrams using Constraint Programming	2008
20	Efficient reasoning about finite satisfiability of UML Class Diagram With Constrained Generalization sets	2007
21	UMLtoCSP A Tool for the Formal Verification of UML/OCL Models Using Constraint Programming	2007
22	A UML Model Consistency Verification Approach Based on Meta-Modeling Formalization	2006
23	Reasoning on UML Class Diagrams with OCL Constraints	2006
24	Verification of UML Model Elements Using B	2006
25	Reasoning on UML Class Diagrams	2005
26	Reasoning on UML Class Diagrams is EXPTIME-hard	2005
27	A Formal V&V Framework for UML Models Based on Model Transformation Techniques	2005
28	An Approach for the Verification of UML Models using B	2004
29	A Formal Framework for Reasoning on UML Class Diagrams	2002
30	Using B Formal Specifications for Analysis and Verification of UML/OCL Models	2002
31	Reasoning on UML Class Diagrams in Description Logics	2001
32	Reasoning on UML Class Diagrams using Description Logic Based Systems	2001
33	The UML as a Formal Modeling Notation	1999
34	Reasoning with UML Class Diagrams	1998
35	Formalizing UML/OCL structural features with FoCaLiZe	2019
36	Incremental Verification of UML/OCL Models.	2020
37	Ontology-Based Transformation and Verification of UML Class Model	2020
38	Formalizing the UML class diagram using Object-Z	1999
39	Finite model reasoning on UML class diagrams via constraint programming	2007
40	Finite satisfiability of class diagrams: Practical occurrence and scalability of the finite sat algorithm	2009
41	Reasoning about UML/OCL class diagrams using constraint logic programming and formula	2019

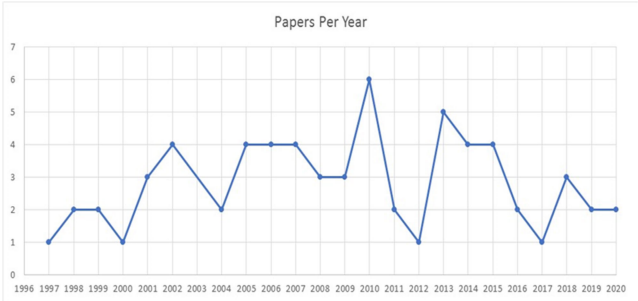

FIGURE 2. Paper included in the review year-wise distribution.

such as inter-model consistency and the UML class model validation did not consider. This study only considers work published after 1996 in English. Table 6 is showing the inclusion and exclusion criteria.

E. IDENTIFICATION OF PAPERS

This review includes papers published between January 1997 and December 2020. The paper searching process was completed in December 2020. Therefore, papers published over the time period as a result of the publication time lags. The search strategy mainly consists of the following:

1) The first strategy was based on research repositories: ACM Digital Library, IEEE Explore, Science Direct, Springer Link, Google Scholar, and Microsoft Academic Search. These libraries have a massive amount of software engineering publications, and the following search strings were used ((“verification” OR “consistency” OR “analysis” OR “reasoning” OR “satisfiability” OR “corrections” OR “checking” OR “unsatisfiable” OR “formalisation”)) AND ((“UML class model”) OR (“UML class diagram”) OR (“UML Model”) OR (“UML/OCL”)).
2) In the last, references of crucial papers were searched manually.

F. STUDY SELECTION

The study (papers) selection phases mainly focus on identifying papers related to the systematic literature review’s goal. The search strings specified in the previous section were

TABLE 4. Relevant work obtained after snowballing.

S#	Title	Pub. Year
01	Towards the Use of Slicing Techniques for an Efficient Invariant Checking	2015
02	Using Slicing to Improve the Performance of Model Invariant Checking	2015
03	Development of Consistent Formal Models	2015
04	Automatic Refinement Checking for Formal System Models	2014
05	Removing Redundancies and Deducing Equivalences in UML Class Diagrams	2014
06	Determining Relevant Model Elements for the Verification of UML/OCL Specification	2013
07	Contract-Aware Slicing of UML class models	2013
08	On challenges of model transformation from UML to Alloy	2010
09	Incremental Integrity Checking of UML/OCL Conceptual Schemas	2009
10	A UML-Based Method for Deciding Finite Satisfiability in Description Logic	2008
11	UML2Alloy: A challenging model transformation	2007
12	Incremental Evaluation of OCL Constraints	2006
13	UML2Alloy: A Tool for Lightweight Modeling of Discrete Event Systems	2005
14	Finite Satisfiability of UML Class Diagram by Constraint Programming	2004
15	Integration of UML and B Specification Techniques Systematic Transformation from OCL Expressions into B	2002
16	Automatic translation from UML specifications to B	2002
17	Integrating UML and B Specification Techniques	2001
18	A Formal Mapping between UML Models and Object-Z Specifications	2000
19	The UML as a Formal Modeling Notation	1998
20	Foundations of the Unified Modeling Language	1997
21	Smart Bound Selection for the Verification of UML/OCL Class Diagrams	2017
22	Ontology-Based Finite Satisfiability of UML Class Model	2018
23	Ontology-Based Verification of UML Class/OCL Model	2018
24	Overview of Slicing and Feedback Techniques for Efficient Verification of UML/OCL Class Diagrams	2018

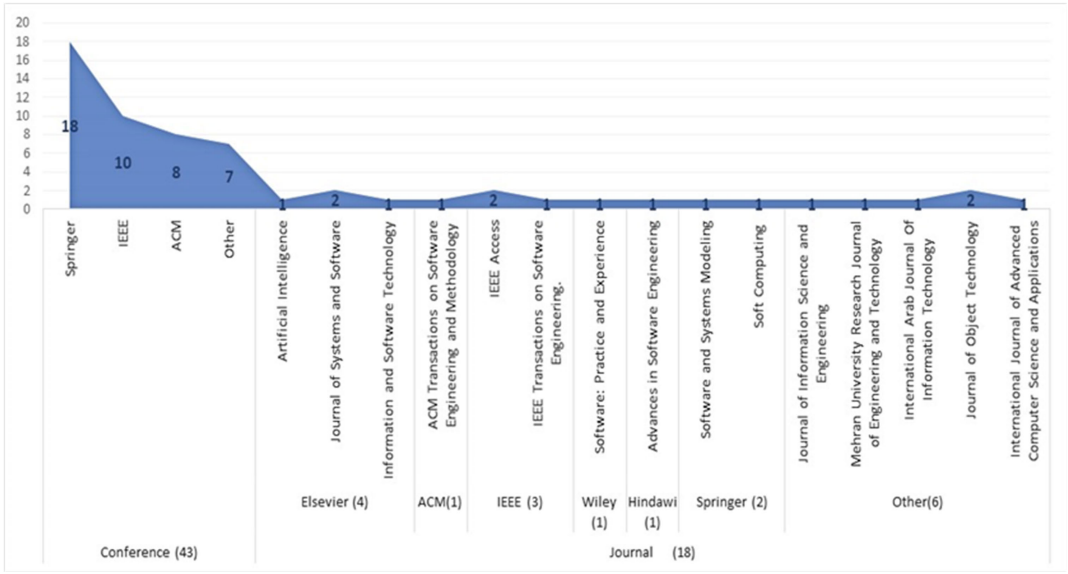

FIGURE 3. Distribution of paper according to the source.

TABLE 5. List of selected studies.

Study	Method Name	References
S1	Z and Object Z	[11]–[16]
S2	B Method	[7], [8], [17]–[20]
S3	Description Logic	[21]–[36]
S4	Constraint Programming	[4], [5], [37]–[46]
S5	Model Slicing	[6], [47]–[55]
S6	Alloy	[56]–[59]
S7	Other	[60]–[66]

very wide and returned papers that are not directly related to the systematic literature review's goal. After collecting 125 papers, the pruning stage was started, which kept out unrelated studies.

In the title pruning stage, the title of papers and keywords were read. After carefully read, 41 papers were excluded because they focused on the verification of other UML models. The remaining 84 papers were finalized for further analysis. In the last, the abstracts of the remaining papers were read, and 47 papers were excluded because they were focused on inter-model consistency. In the end, only 37 articles were selected that were directly related to the purpose of this review. The list of papers is given in Table 3. This list was complemented with papers that are found after the snowballing process.

G. DATA SOURCES

Figure 3 gives a distribution of paper according to the source, which shows that most of the included papers have been

TABLE 6. Inclusion and exclusion criteria.

Inclusion	Exclusion
Paper published in English	Editorials, keynotes, reviews, tutorial summaries, and panel discussions.
Journal papers or conference/ workshop proceedings.	Validation methods of UML class model.
Published after 1996	Inter-model consistency verification.
Focus on verification methods (formal, semi-formal, and non-formal) of the UML class model.	

TABLE 7. General information collected from each study.

Data Item	Description
ID	A unique ID for each paper
Title	Paper Title
Authors Name	Name of all authors
Published years	Publishing year of paper
Citation (Google Scholar)	Number of citations

published in high esteem journals and conferences such as Elsevier, Springer, IEEE, and ACM.

H. DATA EXTRACTION AND SYNTHESIS

This phase extracted related information from each research work. For this purpose, an Excel sheet was prepared, which was used for storing essential information. Mainly, information about each work was recorded in two groups in the following way. As shown in Table 7, general information about each work was collected in the first group, such as title, author name, published year, no of citation. As shown in Table 8, the second group collected the information directly corresponding to the research questions, such as the importance of UML and class model, the name of verification method, and transformation level.

III. RESULTS

A. RQ1—WHAT ARE THE IMPORTANCE OF UML AND UML CLASS MODEL IN MODELING AND MDE?

More or less, all studies (S1 to S7) reported that the UML is an industry adapted visual modeling language. It hides the complexity of the system and provides an appropriate level of abstraction [59]. It is used in the software industry for software specification, analysis, design, documentation, and nowadays, it is also used for code generation [44]. UML has various models that deal with various aspects of software modeling [4], [5]. It uses simple diagrammatic notations for describing software [7]. It is currently also used in engineering, ontology, DBMS, and other areas [32]. UML class model is an integral part of UML [4], [22] [23], [27]–[32], and it is widely used in the industry [27]. It describes the system through concepts, their relationship, and constraint [6]. It is also a key ingredient of the MDE process [6], [47] [49].

B. RQ2—WHAT METHODS OR TECHNIQUES HAVE BEEN ADOPTED FOR VERIFICATION OF THE UML CLASS MODEL?

1) Z NOTATION

The first study uses the Z notation for formalization and verification of the UML class model. This study is divided into 2 sub-studies (SS1.1 and SS1.2). The SS1.1 compiles the work of Evans *et al.* [11]–[14], and SS1.2 collects the work of Kim *et al.* [15], [16].

In SS1.1, the authors used Z notation to provide the formal foundation of the UML core metamodel. They argued that the formal foundation provides various benefits such as clarity, consistency checking, refinement, and proof [12, 13]. They represented the UML core metamodel (also a class model) through a compositional schema that contains several sub-schemes. The sub-schemes formalize UML model elements such as type, instance, values, operation, associations, etc. The authors also describe three ways to formalize the UML model: They also describe following three ways to formalize UML model:

1) Supplemented: In this way, informally defined elements of the UML model are defined formally.
2) OO-Extended: In this approach, the existing formal method is extended with object-oriented concepts such as Object-Z.
3) Method Integration: In this method, the entire UML model is transformed into the formal model.

They also argued that only formal UML analysis is not enough for the discovery of semantic correctness [14]. Moreover, they suggested that industry experts' feedback is also essential for the UML model semantic correctness. They developed a formal reference manual that precisely and formally describes the semantics of UML concepts. The reference manual provides inference rules for analyzing different models properties [14]. In [11], the authors extended their work and proposed a diagrammatic transformation of the UML class model. They further proposed five steps roadmap for model formalization and verification [11]. The roadmap is as follows:

1) A mature formal language must be chosen that should be expressive and well supported by the tools.
2) The abstract syntax of modeling notation (metamodel) should be formalized.
3) Infrastructure parts of modeling notation should be formalized.
4) The meaning function should be defined, which maps the syntax and semantic of the model into the formal notation.
5) Finally, analysis techniques should be developed.

In SS1.2, the authors used the object-z for the core UML model's formal specification and provided a sound mechanism for reasoning [15], [16]. They formalized core UML class model constructs in Object-Z, verified inconsistency and errors through the Z proof technique [15]. They

TABLE 8. Information collected according to the research goal.

Data Item	Description	Research Questions
UML Importance	Record importance of UML	RQ1
Class Model Importance	Record importance of UML Class Model	RQ1
Method	Record information about verification approaches	RQ2
Metamodel level Support	Record transformation of the metamodel	RQ3
Correctness problem	Record correctness problem undertaken	RQ4
Tool Support	Record verification method which complemented with any tool	RQ5

formalized associations, generalization, aggregation, composition, and association classes. [67]. They defined the UML class model's hidden semantics through the schema's invariant of the Z machine. For example, domain and range of association must be classes. The public and protected attributes of the class must be included in the child class. In the aggregation relationship, the whole class should be the range, and the part class should be the domain. They also defined abstract syntax of UML (meta-model) in object-z. In [16], they argued that a single formal method could not cover all analysis tasks of UML class model verification because different formal methods have different strengths and limitations.

Consequently, they presented an integrated verification and validation framework where different formalisms are available for different analysis tasks. It is a hybrid technique where an analyst can select appropriate formalism according to the requirement. Mainly, in this approach, a metamodel is automatically transformed from one formal notation to another.

2) B METHOD

The second study used the B formal method for verification of the UML class model. This study is divided into 2 sub-studies (SS2.1 and SS2.2). The SS2.1 collects the work of Ledang and Souqieres [7], [8] [17]–[19], and SS2.2 presents the work of Marcano and Levy [20].

In SS2.1, the authors presented the transformation of UML classes, attributes, and operations into B Machine [8, 18]. They also presented the transformation of OCL constraint into the B method. They transformed OCL basic types (integer, float, etc.) into B method basic types and operations such as $+$, $-$, etc., into B method basic operations [17]. They used B prover to verify the consistency of UML class model [7]. Further, they integrated all their previous works and proposed the UML class model and metamodel transformation with well-formedness rules in B specification for verification [19]. In this approach, elements of the metamodel's core package are transformed into B abstract machine.

The authors in SS2.2 also used the B method to verify the UML class model [20]. In the approach, the transformation process is done in two stages. Firstly, a core abstract machine (interface) is declared, which describes the complete system structure and the associations. Finally, abstract machines are declared for classes.

3) DESCRIPTION LOGIC

The third study used description logic for verification of the UML class model. This study is mainly divided into 3 sub-studies (SS3.1, SS3.2, and SS3.3). The SS3.1 compiles the work of Cal'I *et al.* and D. Calvanese [21]–[26], SS3.2 collected the work of Maraee and Balaban [27]–[32], and SS3.3 presented the work of Efiizoni *et al.* [33]. The SS3.4 compiles the work of Abdul *et al.* [34]–[36].

In SS3.1, the authors analyzed the UML class model through description logic and verified the model correctness properties, such as redundancies and inconsistencies. They argued that the description logic could adequately deal with the UML class model expressiveness. They mainly verified the UML class model satisfiability and class equivalence [24], [25]. In [26], authors extended their work and performed various reasoning experiments (consistency and satisfiability) on the UML class model through two well-known reasoners, Fact and Racer. They further investigated the computational complexity of UML class model reasoning [21]–[23]. The investigation found that reasoning on the UML class model with minimum supporting features (binary association, minimal multiplicity, and generalization) is ExpTime-hard. They also argued, the reasoning task of checking correctness properties (class subsumption, class equivalence, class consistency) can be mutually reducible into each other.

In SS3.2, the authors proposed bounded verification of constraint generalization set through linear inequalities. They analyzed different generalization types such as tree structure, acyclic structure, and graph structure [32]. In the tree structure generalization, subclasses only inherit from the single superclass. In the acyclic structure generalization, a subclass inherits from multiple superclasses, but an inheritance from a common ancestor is not allowed. In the graph structure generalization, multiple inheritances with different ancestors are allowed. They also proposed two algorithms that reduce the generalization Set (GS) complexity in polynomial time. The first algorithm deals with GS without constraints. The algorithm replaces all GS constructs with binary associations and represents the associations through linear inequalities. The second algorithm supports GS with constraints and does not support disjoint and complete constraint within the cyclic class hierarchy. In [30], the authors proposed another algorithm that supports disjoint and complete constraints over cyclic class hierarchy and introduced a more compact version of linear inequalities. In [31], authors extended their work and added the support of qualifier, association classes in the algorithm. The author also identifies that currently there are no accepted benchmarks for class diagrams for checking the scalability of verification methods. Therefore, they developed a set of metrics for measuring the size and complexity of the

class diagram. Furthermore, they also statistically measure the scalability of finiteSat algorithm [68].

They further extended their work and added the support of qualifier, association classes in the algorithm [31].

Moreover, they also introduced a constraints detecting method that detects the causes of unsatisfiability. The extended algorithm replaces the qualifier constraint with associations and a new class. In [29], the authors formalized their approach, which they presented in [31], [32] through description logic, and built a DL formula for the multiplicity constraint. In [27], they further expanded the work and presented methods for eliminating redundancy in wider constraints (Universal and Extensional).

In SS3.3, the authors also used description logic for the formalization of a class model. They transformed many elements of the class, such as attributes (visibility, type multiplicity, etc.), operations (visibility, parameter list, return types), composition, aggregation, and generalization [33].

In SS3.4, the authors proposed an ontology-based method for verification of the UML class model. In [34], they argued that the UML class model and OCL could be formally represented through the Web Ontology Language (OWL). In [35], the authors presented ontology-based algorithms for checking the satisfiability of association cardinality constraints. The proposed algorithms use the ontology graph and ontology constraints to find the optimal solution. In [36], they also transformed xor constraints and dependency relationship into the ontology.

4) CONSTRAINT SATISFACTION PROBLEM

In the fourth study, constraints programming used for verification of the UML class model. This study is mainly divided into 3 sub-studies (SS4.1, SS4.2, and SS4.3). The SS4.1 focuses on the work of Cadoli *et al.* [4], [69], SS4.2 presents the work of Pérez and Porres [70], SS4.3 presents the work of Malgouyres and Motet [5], and SS4.3 compiles the work of Cabot *et al.* [37]–[46].

In SS4.1, the authors presented a linear inequality-based method for finite model verification through constraint programming. They represented the UML class model through the Constraint Satisfaction Problem (CSP), and the satisfiability of the UML class model is checked by the ILOG's Solver [4]. Furthermore, two correctness problems of the class model were addressed and encoded into CSP. In the first problem, they verify that all the model's classes are completely satisfied at the same time. In the second problem, they verify a finite non-empty model can be obtained from the class model.

In SS4.2 authors presented a framework for checking the satisfiability of UML class model through CLP. The proposed framework detects the design defects in UML class model annotated with OCL. They used the bounded verification approach and performed the reasoning on finite bounds for the number of instances of the model through model-finding tool Formula. The proposed approach checks the predefined correctness properties such as satisfiability, lack of redundant constraints. Furthermore, it can be used to examine complex models for finding best fit object model for the domain. They also implemented the proposed framework through an eclipse plug in called CD-to-Formula.

In SS4.3, the authors used Constraint Logic Programming (CLP) to verify the UML class model. They transformed the UML class model, metamodel, and meta-meta-model into CLP clauses (fact and rules) [5]. In this approach, meta-meta-model, metamodel, and UML class model concrete elements (which have instances) are transformed into CLP facts, and abstract elements and constraints are transformed into the rules. Additionally, CLP's goals are also declared, which negate the consistency rules. In the end, a unified checker finds the solution to the goal, and if the goals are resolved, then the UML class model considers inconsistent.

In SS4.3, the authors proposed incremental verification of the OCL integrity constraint of the class model. They introduced the term Potential Structure Even (PSE) and argued that checking integrity constraints after every structure event (Insert Entity, Update Attribute, Delete Entity, Specialised Entity, etc.) can be costly [44]. On the other side, The PSEs represent only those events that can violate the constraint. Therefore, PSEs for every integrity constraint are recorded. Moreover, only those instances of entity types and relationship types that are the victim of PSEs are verified. They implemented the proposed approach through CSP and presented a fully automatic, decidable solution for binding the UML class model with OCL [38]. Mainly, classes and associations of the UML class model are transformed into a set of variables, domains, and CSP constraints. If the generated CSP is solvable, the model is considered satisfiable otherwise is considered unsatisfiable. They also reported different class models (*CivilStatus, WriterReview, DisjointSubclasses, and ObjectASInteger*) as a benchmark along with various analysis questions such as consistency, invariant independence, consequences [41]. They further introduced two more benchmarks, which were not included in the previous work [42]. The proposed benchmarks target different UML class model features and specified various computational challenges for verification tools. In [37], they pointed out bounded verification limitation and argued that an inadequate bound could miss defects in the model due to small search space or maybe inefficient if set too large. In the proposed solution, large initial bounds are set. Then bounds are tightened up as much as possible, and unnecessary value from the bounds is eliminated through the interval constraint propagation technique. This technique further improved in a way that verification bounds are automatically set whenever possible, then bounds are tightened through user assistance and guide the uses regarding bound setting.

Furthermore, they proposed techniques for the incremental verification of the UML class and OCL model. The proposed techniques merge the slicing technique (which will be discussed in the next study) and a model as a certificate approach presented in the work. In the certificate approach, the model's instance is considered a certificate of satisfiability [46].

5) ALLOY

In the fifth study, a lightweight formal method Alloy has been used to verify the UML class model. This study is mainly divided into two sub-studies (SS5.1 and SS5.2). The SS5.1 compiles the work of Anastasakis *et al.* [56], [58] [59], and SS5.2 presented the work of Maozi *et al.* [57].

In SS5.1, the authors transformed the UML class model and UML metamodel into Alloy specification [58]. They reported different challenges faced by authors during transformation [58]. For example, both approaches support inheritance, but in UML, child classes can redefine the parent class's properties and methods. However, Alloy's sub-signature does not support this feature (overriding is impossible in the Alloy). In [58], they further discussed challenges such as object identifier, multiple inheritances, and collection constraints in more detail. OCL mapping rules also presented in [56]. According to this work, many OCL operations can be directly mapped into the Alloy's elements and operations. Operations that do not have direct corresponding elements in the Alloy are represented by the combination of other elements.

In SS5.2, the authors claimed that the previous UML verification works through Alloy only support limited features and only analyze the model's consistency [57]. This work mapped advanced features (multiple inheritances, interface) of the class model through a combination of Alloy's basic construct (fact, functions, and predicate). This work also supports various analyses on the class model, such as the intersection and refinement analysis.

6) MODEL SLICING

In the sixth study, the authors proposed a technique for minimizing UML class model verification's complexity through model slicing. This study is divided into 2 sub-studies (SS6.1 and SS6.2). The SS6.1 focused on the work of Shaikh *et al.* [6], [47]–[52], and SS6.2 presented the work of Sun *et al.* [53]–[55].

In SS6.1, the authors proposed a model slicing technique for solving UML class model verification's scalability problem. In the proposed approach class model is partitioned into sub-models (Slice); each slice only contains elements that have some impact on model verification and constraints that are not trivially satisfiable such as key unique value assignment, drive value constraints, etc. Elements and constraints which apply to the same model elements are combined into the same slice. The correctness of each slice is checked separately, and finally, the verification result of all slices is combined to determine the correctness of the entire model. The slicing process is initiated from the identification of local and global constraints. The local constraints are verified separately because they do not affect more than one instance of the class. Mainly, a dependency graph is used for analyzing the dependency among class elements, and slices are made from connecting components of the dependency graph. The authors also reported that slicing techniques minimize the verification time of a large model with fewer constraints. However, the slicing technique does not work with complex models. In [50], the authors extended the work and provided the support of non-disjoint sub-models (where a class is used in several constraints). They also reported that if a class model has a large number of disjoint sub-models, then minimum slices will be created, and efficacy cannot be gained. The proposed slicing technique has been applied on two verification methods UMLtoCSP and Alloy, and reported improved results [47]. In [6], they introduced the feedback technique for the unsatisfied UML class model.

Moreover, they reported a slice could be unsatisfiable due to various reasons, e.g. constraints specified an inconsistent condition. Furthermore, they demonstrated the slicing technique on a real-world case study, ''DBLP'' [51]. In [52] they extended the work with the support of both disjoint and non-disjoint slicing. Furthermore, they proposed an overall concept of disjoint and non-disjoint slicing techniques that can split UML/OCL class diagrams into several independent submodels to reduce the complexity. Addtionally, the reasons for unsatisfiability is also discussed through feedback technique.

In the study SS6.2, the authors presented model slicing techniques to optimize two class model analysis techniques. The first technique checks conformance between the object model and class model. The latter one verifies the order of operation invocation to discover the invariant violation. In the first task, they verify the consistency between the object model and the invariant specified in the UML class model and verifies whether the object model is a valid instance of the class model. The proposed approach takes two inputs, object model and class model. The second task presented a rigorous slicing technique that works with both OCL invariants and operations contracts.

7) OTHERS

The last study S7, is a combination of heterogeneous approaches. This study is mainly divided into 3 sub-studies (SS7.1, SS7.2, and SS7.3). The SS7.1 compiles the work of Queralt *et al.* [63], [64] [65], SS7.2 presented the work of Seiter *et al.* [60]–[62]. SS7.3 focused on the work of Abbas *et al.* [66].

In SS7.1, the authors formalized the UML class model through the first-order logic and used the CQC method for verification [62]. They proposed automatic verification of various correctness properties (class liveliness, satisfiability, and redundancy) of OCL integrity constraint. The proposed technique automatically generated tests that verify the association's cardinalities, disjointness, and generalization constraints. Furthermore, it provides a query box reasoning facility for checking reachability for a particular state [64]. In SS7.2, the authors proposed an algorithm for automatically reducing the UML class model's irrelevant elements. The proposed method only considers relevant elements, which are necessary for verification. It takes the verification task and model as input. An empty model is then initialized, which

has only classes without attributes, operations, and invariants. After that, other elements that are related to verification are added for verification. At last, associations are added, which affect verification [62]. In [60], [61], the authors pointed out that current verification techniques only focus on single model verification. They further argued that in Model-Driven Engineering (MDE) abstract model is constructed first, and then subsequent models are built. Focusing on a single model without considering the previous model can sometimes lead to consistency issues between models. It is desirable that the refined model consistent with all preceding models. Finally, they proposed a framework for managing consistency in the verified models.

In SS7.1, the authors presented the UML class model's formal transformation into FoCaLiZe, a proof-based formal approach. It supports various UML class model elements, such as multiple inheritances, dependency, templates, template bindings, and OCL constraints navigation. They convert UML classes in the FoCaLiZe species, properties of classes as a getter function, and OCL constraint as species' properties. Furthermore, in the proposed method, the multiple inheritances are converted into species hierarchy, UML template, template binding, and dependency relationships into species parameters substitution. Species properties that represent OCL constraints converted into FoCaLiZe inheritance and parameterization relationships. They used Zenon for verification [66].

C. RQ3—IS THE VERIFICATION APPROACH IS BASED ON A METAMODEL?

Numerous studies have focused not only on formalizing the UML class model but also on the UML class model meta model partially or completely. In the study [SS1.1], the UML core metamodel was formalized in Z-notation and represented through compositional schemes. In SS1.2, the authors transformed the core UML metamodel in Object-Z meta-Schema. In SS2.1, the authors transformed the UML core metamodel and its well-formedness rules into B abstract machine. They transformed classes, data types, associations, attributes, operations, association ends, and operation parameters of the UML core metamodel into the B method. They also transformed semantic rules of the UML core metamodel into the B method. In SS4.3, the authors transformed the UML class model's metamodel basic elements into the constraint logic program. The metamodel elements such as namespace, class, property, features, named element, and static features were transformed into a CLP meta-fact. In SS5.1, the authors constructed the UML class model's metamodel equivalent metamodel in Alloy. They transformed most UML metamodel constructs into Alloy, such as type, class, package, association, attributes, etc.

D. RQ4—WHICH MODEL DEFECTS HAVE BEEN UNDERTAKEN IN EACH METHOD, AND WHICH DEFECT HAS BEEN EXAMINED IN MOST CASES?

Model validation ensures that the model is error-free and has satisfied correctness features such as satisfactory (strong and weak), consistency, and accuracy such as a class system. The model is considered incorrect when it does not justify any single correctness feature and is considered correct when it meets all the correctness features.

In logic theories, satisfiability means checking the validity of formulas. If the formula is found correct, it is called satisfiable; otherwise, it is called unsatisfiable. Satisfiability is the most crucial correctness property, and it makes sure that a non-empty instance of a model may be created without violation of any constraint. Furthermore, other correctness properties can be covered under it. Some authors also divide satisfiability into two types: weak satisfiability and strong satisfiability. Strong satisfiability checks whether an instances model of the UML class model may be instantiated successfully in which at least one instance of each element successfully populates. Weak satisfiability checks whether at least one or more elements of the model can be instantiated successfully. Class liveliness checks whether a class can be populated successfully. Redundancy of OCL integrity constraint checks the duplication of OCL constraints.

In SS1.1 and SS1.2, the authors verified the UML class model's consistency against the well-formedness rules. In SS1.1, the authors also performed a diagrammatic transformation of the UML class model where one model infer from another through transformation rules. In SS2.1 and SS2.2, the authors checked the consistency of the UML class model against well-formedness rules. They also checked the OCL syntax error, incompatible types of expression, use of the illegal operation. In SS3.1, the authors verified the satisfiability of the UML class model and individual classes. Additionally, they checked class equivalence, class subsumption, and logical consequence.

In SS3.2, the authors verified the finite satisfiability of the unconstrained generalized set, constrained generalized set, and association qualifier of the UML class model. Finite satisfiability (strong satisfiability and weak satisfiability) is checked in SS4.1 and SS4.1. The consistency of the UML class model against the metamodel is verified in SS4.2. In SS4.3, the authors mainly checked finite satisfiability (Weak and Strong Satisfiability) of the UML class model along with OCL constraints. They also verified other correctness features: lack of constraint subsumptions, strong class liveliness, and constraint redundancies. In SS5.1, the authors checked the satisfiability of the UML class model in conjunction with OCL constraints. In SS5.2, the authors checked the intersection of different class models and refinements. The SS7.1 verified satisfiability, class liveliness, redundancy of OCL integrity constraint, and state reachability. In SS7.2, the authors test the state reachability of the model. Figure 4 shows the correctness properties verified by different sub-studies. As shown in Figure 4 satisfiability and consistency have been checked in most of the sub-studies. The correctness property ''well-formedness'' only checked by the initial works, and only two sub-studies verified the UML class model's well-formedness against the metamodel.

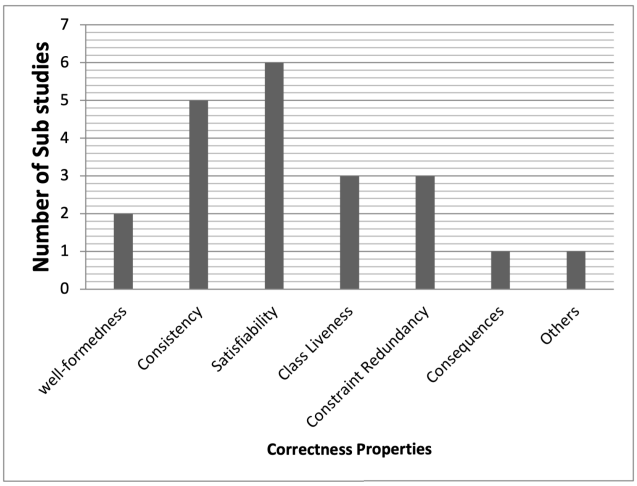

FIGURE 4. Correctness properties verified in different sub-studies.

In most cases, class liveness and constraints redundancy are verified by the works that focus on satisfiability, and 3 sub-studies have verified these properties. Only 1 sub-study also checked the consequences/state reachability. One sub-study checks that the deduced model does not contain any contradiction.

E. RQ5—IS AN AUTOMATIC OR SEMI-AUTOMATIC TOOL DEVELOPED FOR THE VERIFICATION METHOD, AND WHAT ARE THEIR STRENGTHS AND LIMITATIONS?

In SS2.1, the authors reported a prototype tool called ArgoUML+B based on the ArgoUML platform. This tool takes the UML class model in XMI format and automatically transforms it into the B specification. The authors of SS2.2 also developed a prototype tool called OCL2B, which takes the UML class model and OCL file for analysis. This tool used OCAML higher-order language for mapping transformation rules. In SS3.2, the authors presented a tool called FiniteSatUSE, which uses a FiniteSat algorithm for verification of class model with constraints generalization. In SS4.3 authors presented the tool called UMLtoCSP. It takes the class model in the XMI format and OCL as a separate text file. The model and OCL are transformed into the CSP and then checked by the CSP solver. Metadata Repository API parses the XMI file, and the Dresden OCL Toolkit processes OCL constraints. In SS5.1, the authors presented the tool UML2Alloy, which transformed the UML class model and OCL constraint into the Alloy specification. In SS5.2, the authors developed a plug-in CD2Alloy for ECLIPSE. In CD2Alloy, the transformation rules are specified through FreeMarker templates. It provides the facility of class model editing and analyzing. In SS7.1, the authors developed a Java-based prototype tool called AuRUS for Automated Reasoning on UML/OCL Schemas.

IV. DISCUSSIONS

In modern software development, software design models perform critical roles. They are not only used for documentation but also used for analysis, design, testing, and even for code generation through automatic transformation technique. The transformation technique provides automatic reuse of existing software artifacts. However, it has some problems, such as through the transformation, models' defects are automatically transferred in the transformed model. These defects are difficult to discover and repair. Model verification is a promising solution for the problem.

Verification of the UML class model performs a vital role in ensuring the model quality before the transformation. Verification of the UML class model through formal notation has been discussed in several works. In the existing literature, different facets of UML class model correctness have been discussed by the researchers according to different factors such as static vs. dynamic and inter vs. intra-model [66]. Under the static factor, only the UML class model's structural elements are considered for verification, such as association multiplicities and generalization. In the dynamic category, the behavior part, such as operation, is considered for analysis. In the inter-model verification, consistency between two different models is verified, and in the intra-model verification, consistency and satisfiability of the model against constraints are verified.

Consistency and well-formedness are the most fundamental correctness features [48]. The consistency verifies whether the model elements are consistent with the declaration, whereas well-formedness verifies whether a model is a correct instance of its metamodel [66]. However, well-formedness only verifies the initial level of syntax weaknesses and is not concerned with the model's semantic correctness. Semantic correctness concerns the constraints that are graphically specified in the model, such as associations, dependencies, generalizations, or textually defined through constraint languages OCL.

The most fundamental semantic correctness property of the static model is satisfiability [66]. It checks the possibility of the creation of an instance model without violation of any constraint. Other important correctness properties which are verified and come under the umbrella of satisfiability are strong satisfiability, weak satisfiability, and class liveliness [38]. The UML class model provides graphical modeling notation without any formal foundation [7]. The well-formedness rules are specified through metamodel and OCL without any formal proof facility.

Consistency and finite satisfiability are two major correctness features of the UML class model, and both guarantee a non-empty and finite instance model. The consistency focuses on non-emptiness, and finite satisfiability focuses on finiteness [71]. Contradicting constraints such as creating a subclass of two disjoint classes can cause an emptiness, and little mistake in association multiplicity constraints specification can cause non-finiteness. A class model is considered consistent 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 [71]. Hence, finding a single legitimate instance of

TABLE 9. Limitation of verification methods.

UML Features	Study 1 (Z)		Study 2 (B)		Study 3 (DL)			Study 4 (CFP)				Study 5 (Alloy)		Study 6 (Model Slicing)		Study 7 (Other)		
	1	2	1	2	1	2	3	1	2	3	4	1	2	1	2	1	2	3
Association	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Association Classes		✓			✓	✓		✓			✓	✓						✓
Aggregation		✓		✓	✓		✓	✓				✓	✓					✓
Composition							✓	✓					✓					✓
Generalization	✓	✓			✓	✓	✓	✓		✓	✓	✓	✓	✓			✓	✓
N-ary Association						✓	✓				✓							✓
Association Qualifier							✓											
Xor Association		Not Supported																
Dependency																		
Stereotype																		

TABLE 10. Efficiency analysis for some verification tools.

Tool Name	Classes / scope	No of Constraints	Efficiency
UMLtoCSP	Small (2-50)	Small (5-10)	in few minutes
	Large (100- Above)	Small (5-10)	Timeout
UML2Alloy	Small (1-7)	Small (5-10)	In few minutes
	Large (100-1000)	Small (5-10)	Timeout
USOT	Large (above 100)	Small (5-10)	Less time required as compared to CSP and Alloy (efficiency gain)
	Large (above 100)	Large (100-above)	Timeout

TABLE 11. Prototype tools availability on the internet.

S#	Name of Tool	Status	Web Address
01	ArgoUML +B	Not Available	N/A
02	FiniteSatUSE	Available	https://sourceforge.net/projects/usefsverif/
03	UMLtoCSP	Available	http://gres.uoc.edu/UMLtoCSP/
04	UML2Alloy	Available	http://www.cs.bham.ac.uk/∼bxb/UML2Alloy/
05	CD2Alloy	Available	https://sourceforge.net/projects/alloymda/files/
06	AuRUS	No Available	N/A
07	UOST	Available	http://asadshaikh.com/UMLtoCSP_UOST/

the class model is adequate to guarantee consistency in which all classes are non-empty. Therefore, finite satisfiability can cover consistency [71]. Many other correctness properties such as class liveliness, constraint redundancy, and subsumption can be verified under satisfiability [66]. However, another important aspect of model verification is consequences, which infer new properties from the existing properties [41].

Current UML class model verification methods efficiently verify the correctness of the model. However, in some cases, their performance goes down mostly when they deal with the large and complex model or consume a lot of computational resources (CPU, Memory). Furthermore, they do not support some essential elements of the UML class model. The major limitations of each verification method regarding the UML class model's supporting features are shown in Table 9. The majority of the methods support association, aggregation, and generalization relationships and do not support the xor constraint, dependency relationships, and stereotypes.

Another analysis is presented in Table 9 based on 7 studies, which describes some verification methods' efficiency. As shown in Table 10, current methods work well on a small model with few constraints, such as the performance of UMLtoCSP goes down when it performs verification on the large and complex model. Table 10 also depicts the execution time of UML2Alloy, which depends upon the scope, and its efficiency goes down when deals with a larger scope. Additionally, some improvement can be achieved by the slicing techniques on both approaches. However, UOST also has some limitations, such as it does not work with the complex (constraints rich) UML class model. Approximately all existing verification methods supported their method through prototype tools. Table 11 shows the availability of the tools on the internet.

Therefore, there are two open issues. First, some important UML class model elements have not been supported by existing verification methods, such as xor associations, dependency relationships, and stereotypes. Second, there is a need for an optimized technique that reduces the verification time.

V. THREAD TO VALIDITY

It is very important for every SLR to identify all relevant studies. To deal with this threat, the research protocol was formed and validated rigorously to reduce the risk of eliminating relevant studies. The Searching terms were designed in a way that only a trivial number of relevant studies could be missed, and very few irrelevant studies could be included. Manual search and snowballing were also performed besides the automatic search. The protocol was carefully designed to be reusable by others for reproducing the same study. This review has a few limitations. Like all reviews, it was limited by the search conditions used, the journals and conferences included, and the time period. However, the research works presented in this literature review provide a snapshot of UML class model verification methods. This review excluded non-English papers. Although many aspects of UML class model verification are discussed here, there may be other important

aspects in model verification. The review studies are coded with respect to their strengths and weaknesses, although the verification methods' challenges during transformation and verification have not been discussed.

VI. CONCLUSION

This study is based on a systematic review of UML class model verification literature with and without OCL constraints. This work identifies numerous challenges the researcher faces during the model transformation process and presents potential strategies to tackle these challenges. This work presents the findings in two phases: The first phase provides quantitative information on the number of works published each year since 1996, the type of methods reported in reviewed work, and the reported studies' contextual factor. In the second phase, studies are analyzed and extracted data interpreted to answer research questions.

Our findings suggest that existing UML class model verification methods provide great efforts in verifying the UML class model. However, there are some limitations. Firstly, almost all works only focus on core UML class elements such as classes, binary associations, generalization, and aggregation. They do not focus on the other elements such as xor constraint over the associations, dependency relationships, and qualified associations, n-ary associations, and stereotypes. Most of the initial studies only focus on the UML class model's well-formedness due to the unavailability of a formal foundation in UML. Modern studies mainly work on consistency and satisfiability. Some other correctness properties such as class liveliness, constraint redundancy, and subsumption have also been verified in different research studies. However, less attention has been paid to them than consistency and satisfaction.

REFERENCES

[1] M. Kardoš and M. Drozdová, "Analytical method of CIM to PIM transformation in model driven architecture (MDA)," *J. Inf. Organizational Sci.*, vol. 34, no. 1, pp. 89–99, 2010.

[2] F. Hilken and M. Gogolla, "User assistance characteristics of the USE model checking tool," 2017, *arXiv:1701.08471*. [Online]. Available: http://arxiv.org/abs/1701.08471

[3] 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.

[4] M. Cadoli, D. Calvanese, G. De Giacomo, and T. Mancini, "Finite satisfiability of UML class diagrams by constraint programming," *CSP Techn. Immediate Appl.*, vol. 2, pp. 2–16, Sep. 2004.

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

[6] 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, Nov. 2014.

[7] N.-T. Truong and J. Souquières, "An approach for the verification of UML models using B," in *Proc. 11th IEEE Int. Conf. Workshop Eng. Comput. Syst.*, May 2004, pp. 195–202.

[8] H. Ledang and J. Souquières, "Integrating UML and B specification techniques," in *Proc. Informatik Workshop Integrating Diagrammatic Formal Specification Techn.*, 2001, p. 8.

[9] C. A. González and J. Cabot, "Formal verification of static software models in MDE: A systematic review," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 821–838, Aug. 2014.

[10] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Softw. Eng. Group School Comput. Sci. Math., Keele Univ., Keele, U.K., EBSE Tech. Rep. EBSE-2007-01, 2007.

[11] R. France, A. Evans, K. Lano, and B. Rumpe, "The UML as a formal modeling notation," *Comput. Standards Interfaces*, vol. 19, no. 7, pp. 325–334, Nov. 1998.

[12] A. S. Evans, "Reasoning with UML class diagrams," in *Proc. 2nd IEEE Workshop Ind. Strength Formal Specification Techn.*, Oct. 1998, pp. 102–113.

[13] T. Clark and A. Evans, "Foundations of the unified modeling language," in *Proc. 2nd Northern Formal Methods Workshop*. Ilkley, U.K.: Springer, Jul. 1997, pp. 1–15.

[14] A. Evans, R. France, K. Lano, and B. Rumpe, "The UML as a formal modeling notation," in *Proc. Int. Conf. Unified Modeling Lang.* Berlin, Germany: Springer, 1998, pp. 336–348.

[15] S.-K. Kim and D. Carrington, "A formal mapping between UML models and Object-Z specifications," in *Proc. Int. Conf. B Z Users*. London, U.K.: Springer-Verlag, 2000, pp. 2–21.

[16] S.-K. Kim and D. Carrington, "A formal V&V framework for UML models based on model transformation techniques," in *Proc. 2nd MoDeVa Workshop-Model Design Validation*, Inria, France, 2005, pp. 1–7.

[17] H. Ledang and J. Souquieres, "Integration of UML and B specification techniques: Systematic transformation from OCL expressions into B," in *Proc. 9th Asia–Pacific Softw. Eng. Conf.*, 2002, pp. 495–504.

[18] H. Ledang, "Automatic translation from UML specifications to B," in *Proc. 16th Annu. Int. Conf. Automated Softw. Eng. (ASE)*, 2001, p. 436.

[19] N. T. Truong and J. Souquieres, "Verification of UML model elements using B," *J. Inf. Sci. Eng.*, vol. 22, pp. 357–373, Oct. 2007.

[20] R. Marcano and N. Levy, "Using B formal specifications for analysis and verification of UML/OCL models," in *Proc. Workshop Consistency Problems UML Softw. Develop., 5th Int. Conf. Unified Modeling Lang.*, 2002, pp. 91–105.

[21] A. Artale, D. Calvanese, and A. Ibáñez-García, "Full satisfiability of UML class diagrams," in *Proc. Int. Conf. Conceptual Model.* Berlin, Germany: Springer-Verlag, 2010, pp. 317–331.

[22] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artif. Intell.*, vol. 168, nos. 1–2, pp. 70–118, Oct. 2005.

[23] D. Berardi, D. Calvanese, and G. D. Giacomo, "Reasoning on UML class diagrams is EXPTIME-hard," in *Proc. Int. Workshop Description Logics, DL, CEUR Workshop*, vol. 81. Rome, Italy: CEUR-WS.org, 2003.

[24] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini, "A formal framework for reasoning on UML class diagrams," in *Proc. Int. Symp. Methodol. Intell. Syst.* Berlin, Germany: Springer, 2002, pp. 503–513.

[25] A. Calı, D. Calvanese, G. De Giacomo, and M. Lenzerini, "Reasoning on UML class diagrams in description logics," in *Proc. Workshop Precise Modeling Deduction Object Softw. Develop. (PMD IJCAR)*, 2001, pp. 77–86.

[26] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams using description logic based systems," in *Proc. Workshop Appl. Description Logics*, vol. 44, 2001, pp. 1–12.

[27] A. Maraee and M. Balaban, "Removing redundancies and deducing equivalences in UML class diagrams," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.* Cham, Switzerland: Springer, 2014, pp. 235–251.

[28] M. Balaban and A. Maraee, "Finite satisfiability of UML class diagrams with constrained class hierarchy," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 1–42, Jul. 2013.

[29] M. Balaban and A. Maraee, "A UML-based method for deciding finite satisfiability in description logics," in *Description Logics*. Berlin, Germany: Springer, 2008, pp. 1–11.

[30] 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.*, Mar. 2009, pp. 1–8.

[31] A. Maraee, V. Makarenkov, and B. Balaban, "Efficient recognition and detection of finite satisfiability problems in uml class diagrams: Handling constrained generalization sets, qualifiers and association class constraints," in *MCCM*. Berlin, Germany: Springer, 2008.

[32] A. Maraee and M. Balaban, "Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets," in *Proc. Eur. Conf. Model Driven Archit.-Found. Appl.* Berlin, Germany: Springer, 2007, pp. 17–31.

[33] L. Efrizoni, T. Informatika, W. M. N. Wan-Kadir, and R. Mohamad, "Formalization of UML class diagram using description logics," in *Proc. Int. Symp. Inf. Technol.*, vol. 3, Jun. 2010, pp. 1168–1173.

[34] A. Hafeez, S. H. A. Musavi, and A. U. Rehman, "Ontology-based verification of UML class/OCL model," *Mehran Univ. Res. J. Eng. Technol.*, vol. 37, no. 4, pp. 521–534, 2018.
[35] A. H. Khan, S. H. A. Musavi, A.-U. Rehman, and A. Shaikh, "Ontology-based finite satisfiability of UML class model," *IEEE Access*, vol. 6, pp. 3040–3050, 2018.
[36] A. Hafeez, S. Abbas, and A.-U. Rehman, "Ontology-based transformation and verification of UML class model," *Int. Arab J. Inf. Technol.*, vol. 17, no. 5, pp. 758–768, Sep. 2020.
[37] R. Clarisó, C. A. González, and J. Cabot, "Towards domain refinement for UML/OCL bounded verification," in *Proc. Collocated Workshops (SEFM)*. Berlin, Germany: Springer-Verlag, 2015, pp. 108–114.
[38] J. Cabot, R. Clariso, and D. Riera, "Verification of UML/OCL class diagrams using constraint programming," in *Proc. IEEE Int. Conf. Softw. Test. Verification Validation Workshop*, Apr. 2008, pp. 73–80.
[39] J. Cabot and E. Teniente, "Incremental integrity checking of UML/OCL conceptual schemas," *J. Syst. Softw.*, vol. 82, no. 9, pp. 1459–1478, Sep. 2009.
[40] J. Cabot, R. Clarisó, and D. Riera, "On the verification of UML/OCL class diagrams using constraint programming," *J. Syst. Softw.*, vol. 93, pp. 1–23, Jul. 2014.
[41] M. Gogolla, F. Büttner, and J. Cabot, "Initiating a benchmark for uml and ocl analysis tools," in *Proc. Int. Conf. Tests Proofs*. Berlin, Germany: Springer-Verlag, 2013, pp. 115–132.
[42] J. Cabot, R. Clarisó, and D. Riera, "UMLtoCSP: A tool for the formal verification of UML/OCL models using constraint programming," in *Proc. 22nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2007, pp. 547–548.
[43] M. Gogolla and J. Cabot, "Continuing a benchmark for UML and OCL design and analysis tools," in *Proc. Fed. Int. Conf. Softw. Technol., Appl. Found.* Cham, Switzerland: Springer, 2016, pp. 289–302.
[44] J. Cabot and E. Teniente, "Incremental evaluation of OCL constraints," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Cham, Switzerland: Springer, 2006, pp. 81–95.
[45] R. Clarisó, C. A. González, and J. Cabot, "Smart bound selection for the verification of UML/OCL class diagrams," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 412–426, Apr. 2019.
[46] R. Clariso, C. A. Gonzalez, and J. Cabot, "Incremental verification of UML/OCL models," *J. Object Technol.*, vol. 19, no. 3, pp. 1–3, 2020.
[47] 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, pp. 1–18, Sep. 2011.
[48] A. Shaikh, R. Clarisó, U. K. Wiil, and N. Memon, "Verification-driven slicing of UML/OCL models," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2010, pp. 185–194.
[49] A. Shaikh and U. K. Wiil, "UMLtoCSP (UOST): A tool for efficient verification of UML/OCL class diagrams through model slicing," in *Proc. 20th Int. Symp. Found. Softw. Eng. (FSE ACM SIGSOFT)*, 2012, pp. 1–4.
[50] A. Shaikh, U. K. Wiil, and N. Memon, "UOST: UML/OCL aggressive slicing technique for efficient verification of models," in *Proc. Int. Workshop Syst. Anal. Modeling*. Berlin, Germany: Springer-Verlag, 2010, pp. 173–192.
[51] A. Shaikh and U. Kock, "Efficient verification-driven slicing of UML/OCL class diagrams," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 5, pp. 530–547, 2016.
[52] A. Shaikh and U. K. Wiil, "Overview of slicing and feedback techniques for efficient verification of UML/OCL class diagrams," *IEEE Access*, vol. 6, pp. 23864–23882, 2018.
[53] W. Sun, R. B. France, and I. Ray, "Contract-aware slicing of UML class models," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.* Miami, FL, USA: Springer, 2013, pp. 724–739.
[54] W. Sun, B. Combemale, R. B. France, A. Blouin, B. Baudry, and I. Ray, "Using slicing to improve the performance of model invariant checking," *J. Object Technol.*, vol. 28, p. 28, Jul. 2015.
[55] W. Sun, B. Combemale, and R. B. France, "Towards the use of slicing techniques for an efficient invariant checking," in *Proc. Companion 14th Int. Conf. Modularity*, Mar. 2015, pp. 23–24.
[56] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "On challenges of model transformation from UML to alloy," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 69–86, Jan. 2010.
[57] S. Maoz, J. O. Ringert, and B. Rumpe, "CD2Alloy: Class diagrams analysis using Alloy revisited," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.* Berlin, Germany: Springer-Verlag, 2011, pp. 592–607.
[58] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A challenging model transformation," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.* Springer, 2007, pp. 436–450.
[59] B. Bordbar and K. Anastasakis, "UML2ALLOY: A tool for lightweight modelling of discrete event systems," in *Proc. IADIS AC*, 2005, pp. 209–216.
[60] R. Drechsler and U. Kühne, *Formal Modeling and Verification of Cyber-Physical Systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems*. Bremen, Germany, Sep. 2015.
[61] J. Seiter, R. Wille, U. Kühne, and R. Drechsler, "Automatic refinement checking for formal system models," in *Proc. Forum Specification Design Lang. (FDL)*. Cham, Switzerland: Springer, Oct. 2014, pp. 3–22.
[62] J. Seiter, R. Wille, M. Soeken, and R. Drechsler, "Determining relevant model elements for the verification of UML/OCL specifications," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 1189–1192.
[63] A. Queralt and E. Teniente, "Reasoning on UML class diagrams with OCL constraints," in *Proc. Int. Conf. Conceptual Modeling*. Berlin, Germany: Springer-Verlag, 2006, pp. 497–512.
[64] C. Farré, A. Queralt, G. Rull, E. Teniente, and T. Urpí, "Automated reasoning on UML conceptual schemas with derived information and queries," *Inf. Softw. Technol.*, vol. 55, no. 9, pp. 1529–1550, Sep. 2013.
[65] A. Queralt, G. Rull, E. Teniente, C. Farré, and T. Urpí, "AuRUS: Automated reasoning on UML/OCL schemas," in *Proc. Int. Conf. Conceptual Modeling*. Berlin, Germany: Springer-Verlag, 2010, pp. 438–444.
[66] M. Abbas, C.-B. Ben-Yelles, and R. Rioboo, "Formalizing UML/OCL structural features with FoCaLiZe," *Soft Comput.*, vol. 24, no. 6, pp. 4149–4164, Mar. 2020.
[67] S.-K. Kim and C. David, "Formalizing the UML class diagram using Object-Z," in *Proc. Int. Conf. Unified Modeling Lang.* Berlin, Germany: Springer-Verlag, 1999, pp. 83–98.
[68] V. Makarenkov, P. Jelnov, A. Maraee, and M. Balaban, "Finite satisfiability of class diagrams: Practical occurrence and scalability of the *FiniteSat* algorithm," in *Proc. 6th Int. Workshop Model-Driven Eng., Verification Validation (MoDeVVa)*, 2009, pp. 1–10.
[69] M. Cadoli, D. Calvanese, G. De Giacomo, and T. Mancini, "Finite model reasoning on UML class diagrams via constraint programming," in *Proc. Congr. Italian Assoc. Artif. Intell.* Berlin, Germany: Springer-Verlag, 2007, pp. 36–47.
[70] B. Pérez and I. Porres, "Reasoning about UML/OCL class diagrams using constraint logic programming and formula," *Inf. Syst.*, vol. 81, pp. 152–177, Mar. 2019.
[71] J. Cabot and R. Clarisó, "UML/OCL verification in practice," in *Proc. Workshop Int. Workshop Challenges Model-Driven Softw. Eng. (ChaMDE)*, 2008, pp. 31–35.