Managing Inconsistencies in UML Models: A Systematic Literature Review

Driss Allaki*, Mohamed Dahchour, Abdeslam En-nouaary
National Institute of Posts and Telecommunications, Rabat, Morocco.

* Corresponding author. Tel.: +212 673 331 551; email: d.allaki@inpt.ac.ma
Manuscript submitted March 13, 2017; accepted May 24, 2017.
doi: 10.17706/jsw.12.6.454-471

Abstract: Software systems are often modeled as a set of related UML diagrams. Due to the overlapping multi-view nature of UML and due to the incremental and iterative nature of the used software development process, these diagrams might contain inconsistencies. Thus, it is of utmost importance to detect, analyze and fix these inconsistencies before implementing the system. In this paper, we elaborate a transverse view of the aforementioned activities. More explicitly, this work proposes a Systematic Literature Review (SLR) that evaluates the existing techniques for managing inconsistencies using different research questions. The ultimate objective of this review is to come up with new recommendations that should be taken into consideration when conceiving a new proposal for checking inconsistencies.

Key words: UML diagrams, model validation, inconsistency management, SLR..

1. Introduction

Software models provide an abstraction or an approximate representation of a software system using sufficient graphical information. They are used to specify, visualize, document and communicate the developed system. The Unified Modeling Language (UML) is the standard advocated by the software industry to be used for designing software models and related artifacts. UML is seen as a graphical and textual modeling language composed of a set of related diagrams that unifies both notations and object-oriented concepts to design all the aspects of a software system.

However, the overlapping multi-view nature of the modeling artifacts in UML, in addition to the incremental and iterative nature of the used software development process may be the source of several inconsistencies [1]. An *inconsistency* roughly means that overlapping elements of different model aspects don't match each other. In other words, the whole system is not represented in a harmonized way in different views of its model being syntactically or semantically in contradiction, incomplete, or reflecting ambiguities or anomalies [2].

Thus, particular attention should concern managing inconsistencies to undergo changes during a software life cycle, correct errors, accommodate new requirements, and so on. Indeed, inconsistency management has been defined in [3] as "the process by which inconsistencies between software models are handled so as to support the goals of the stakeholders concerned". More specifically, the process of inconsistency management includes activities of *detecting*, *diagnosing*, and *handling* inconsistencies.

Detection of inconsistencies is the activity of checking for inconsistencies in software models [4]. In literature, different proposals have been suggested for this purpose (for instance [5], [20], [31] and [34]).

Diagnosis of inconsistencies is concerned with "the identification of the source, the cause and the impact of an inconsistency" [11]. In addition to that, different kinds of inconsistencies can be distinguished. In fact, the classification of inconsistencies must be carefully defined to help diagnosing them.

On the other hand, the activity of handling inconsistencies consists of the identification of the possible actions for dealing with each of which, the selection of one of the actions to be executed and the evaluation of the possible risks and benefits that can arise when applying a resolution action.

In the last two decades, many works have been proposed to address the UML model inconsistency problem. Among them some are dedicated to the detection of inconsistencies, while others focus on the best ways to resolve them. Whatever the motivation behind or the approach used, the benefit aimed by ensuring the models' consistency is to minimize costs, time and effort upstream in software development process. This can be ensured by producing consistent models that are able to offer reliable and error free code; which can be an important step in taking a model-driven approach to software development. Otherwise, the model inconsistencies can lead to the development of an improper design that may lead to project termination since the designed system cannot guarantee to meet customer needs.

Thereby, this work was carried out to complete other reviews and surveys ([11]-[16]) about UML model consistency management in order to come up with more knowledge, open issues and trends within this scope. Hence, the aim of this paper is to evaluate the current status of the UML model inconsistency management and to propose some guides to help researchers, modelers and developers devise new consistency checking proposals that satisfy the needed requisites of software industrialization. From this perspective, the main contribution of this paper is to propose a Systematic Literature Review (SLR) of the different existing techniques on managing inconsistencies. This study takes into consideration new parameters regarding related work and differently discusses the obtained results.

The remainder of this paper is organized as follows. Section II proposes a running example to illustrate the problem and introduces the motivation behind this study. Section III presents related work on this field. Section IV explains the research methodology followed to carry out the SLR. Section V presents the obtained results. Section VI provides a discussion about the aforementioned results. Section VII concludes the paper and presents future work.

2. Problem Statement and Motivation

In this section, we introduce some examples to illustrate the inconsistency problem and we explain the reasons why we adopt a Systematic Literature Review to study and evaluate the existing inconsistency management techniques.

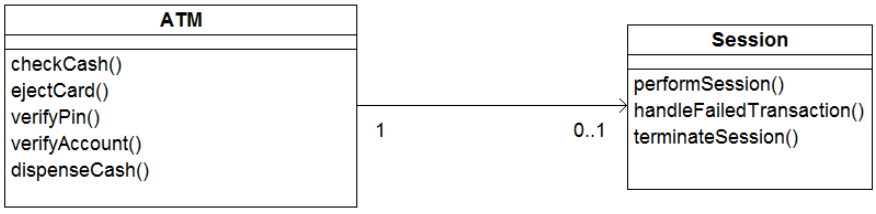

Fig. 1. A part of a class diagram for ATM.

2.1. Illustrative Example

To illustrate an inconsistency, we present an extract of a UML model, inspired from [18], that expresses the functioning of an Automated Teller Machine (ATM). The UML model consists of three diagrams: a class diagram showing the entities *ATM* and *Session* as presented in Fig. 1, a sequence diagram outlining the

interaction between the two instances of these classes as illustrated in Fig. 2, and a statechart depicting the state of the *ATM* class as expressed in Fig. 3.

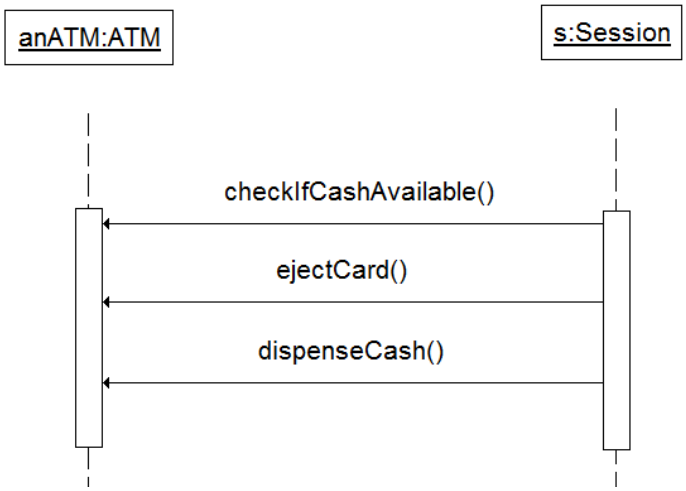

Fig. 2. A part of a sequence diagram for ATM.

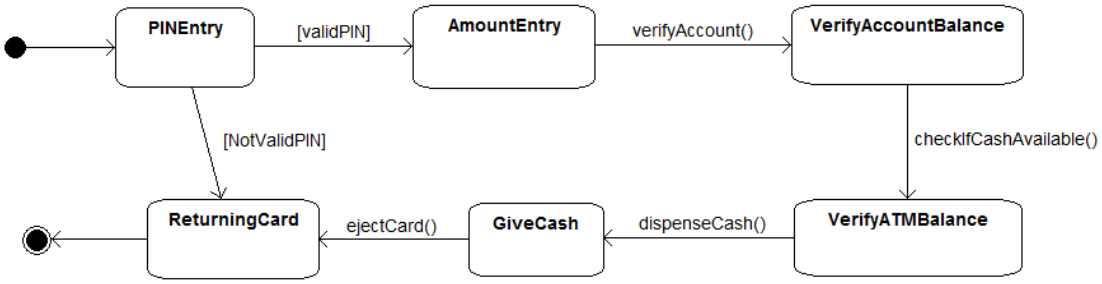

Fig. 3. A part of a statechart diagram for ATM.

We remark that the presented model contains the three following consistency violations:

1) The sequence diagram does not respect the order of operations imposed by the statechart since the operations “dispenseCash” and “ejectCard” appear in a different order in both diagrams.
2) The three messages of the sequence diagram “checkIfCashAvailable”, “ejectCard” and “dispenseCash” do not match the direction of the class association (only class ATM can access Session).
3) The ATM class has no operation that matches the “checkIfCashAvailable” name of both the message and the transition of respectively the sequence and the statechart diagrams.

For illustrative purposes, we limit ourselves to these examples. Further consistency rules can be found in the systematic study presented in [14].

The first inconsistency concerns the semantic behavior of different diagrams at the same level of abstraction. Since the succession of the statechart operations is in contradiction with the order of the messages in the sequence diagram. The second example neglects the existence of the navigation direction of the association between classes defined in the class diagram. While the third example presents a semantic incompleteness between the elements of the three diagrams.

This kind of inconsistencies can be the result of many software development attitudes. For instance, they can occur when rearrangements are carried out on an existing part of the system or when changing the modelled design or part of it.

The taxonomy presented in [6] provides more explanation and clarification about the different categories of inconsistencies in UML models and their classification.

2.2. Motivation

Due to the increasing number of publications in managing inconsistencies, there is a need for summarizing and classifying the existing findings in literature. Systematic studies can help researchers have a quick and efficient review on a research topic without the need of searching for all the publications in the field. As described by Kitchenham in [44], the Systematic review is a very effective method developed in a formal and systematic way to give a structured summary of the existing work. It is related to the outcomes of studies on specific research questions. The research is conducted using a process that follows a well-defined and strict sequence of methodological steps, according to a developed protocol.

In this work, we aim to provide an evaluation of existing consistency checking proposals by following the well-known trustworthy methodology, referred to as Systematic Literature Review (SLR) [44]. The first objective of this evaluation is to identify any gaps in current research in order to suggest areas for further investigation. The second objective of the evaluation is to provide a useful background to appropriately position new research activities.

3. Related Work

To give an overview of related work that deal with systematic review on managing UML inconsistencies, we considered that it would be important to propose the six following pieces of work that increase the body of knowledge on UML models' consistency. Then, we show on what our work differs from them.

Spanoudakis and Zisman [11] presented a literature review on managing inconsistencies in software models (not specifically UML models). They proposed a conceptual framework that organizes inconsistency management as a process and surveyed the research proposals that addresses one or more of the aspects of their proposed framework.

Usman *et al*. [12] analyzed the existing techniques on the basis of analysis parameters in a survey of consistency checking techniques for UML models. The authors provided an initial summary of their findings without adopting any specific search protocol. They argued that formalization of UML models is preferable to validate consistency.

Lucas, Molina and Toval [13] provided a Systematic Literature Review to identify and evaluate the different approaches for model consistency management. They concluded their work with a set of recommendations to improve forthcoming consistency checking proposals.

Torre *et al*. [14] focused on their SLR on creating a consolidated set of UML consistency rules and on identifying, for all the UML diagrams, which consistency rules have been proposed in literature.

Bashir *et al*. [15] provided an overview and a classification of current state-of-the-art UML-based model consistency management techniques. The classification is based on different criteria extracted from literature. The authors concluded their work with a set of research trends and challenges to provide a baseline for future research in this domain of study.

Knapp and Mossakowski [16] studied the question of multi-view models in UML/OCL by surveying the large amount of literature and by giving a classification of the used techniques for multi-view UML/OCL consistency. Also, they supported their work with a new approach to UML multi-view consistency, following a "heterogeneous transformation" paradigm.

Our work differs from the SLR studies presented above on three aspects: new research questions, qualitative evaluation, and new recommendations for research. In fact, the objective of our systematic literature review is to study the typological nature, the consistency strategy and the used technique and/or formalism of the consistency checking proposals. Furthermore, we adopt a set of qualitative features that we found interesting to evaluate the selected proposals. The aim of this study is to identify the strengths and the weaknesses of the used paradigms, techniques and strategies from different aspects in order to

come up with new recommendations that should be taken into consideration when conceiving new solutions for checking inconsistencies.

4. Research Methodology

The process of conducting a Systematic Literature Review must be precisely defined, documented, and include intermediate results. To this end, Kitchenham in [44] proposes guidelines that facilitate the planning and the execution of an SLR. In this paper, we have been inspired by the aforementioned guidelines to conduct out our SLR. Therefore, the activities carried out for this purpose were:

1) Research questions formulation
2) Search criteria selection
3) Inclusion and exclusion criteria definition
4) Search strategy execution
5) The following gives more details about each of these activities.

4.1. Research Questions

The underlying motivations for the research questions are to overview and analyze the state of the art regarding the techniques and their adequate formalisms used in the Consistency Checking Proposals (called hereafter CCPs). To this end, we consider four Research Questions (RQs) presented as follows:

RQ1) *What is the typology of the selected CCPs?*

This aims to precise the typological nature of the proposal. As explained in [17], these proposals may be classified into many categories, we mainly mention among them:

1) Tools: technologies and applications that can be used to identify and resolve UML model inconsistencies.
2) Processes that consist of a set of linked procedures which take one or more resources to convert inputs into outputs until gaining certain outcome on ensuring models' consistency.
3) Theories: or set of statements that describe logical facts on checking inconsistencies, especially those repeated several times in different environments.
4) Frameworks: models used to describe UML model validation entities in a hypothetical manner.
5) Methods: procedures and techniques used to check the consistency with accuracy and efficiency.
6) Approaches: strategies which help solving consistency problems in UML.
7) Extensions: extending existing technologies to provide new solutions that deal with additional aspects of the problem.
8) Algorithms: which help solving consistency problems in UML.

RQ2) *What is the consistency strategy of these CCPs?*

This aims to identify the consistency strategy used to detect UML model inconsistencies. As indicated in [12] we identify:

1) The consistency by monitoring (based on rules and constraints).
2) The consistency by construction (generates one artifact from another). In this transformation-based strategy, we propose, as identified in [7], a sub-classification according to the formal used paradigm (state transition, process algebra and logic).

RQ3) *What are the Techniques and Formalisms used in these CCPs?*

This aims to define the existing used techniques and their adequate formalisms to detect inconsistencies. This also helps researcher identify the differences and commonalities between the presented solutions.

RQ4) *Which of these CCPs are in compliance with the targeted quality features?*

This aims to surround almost all the essential aspects of an effective CCP. The proposed features include, but not limited to, the following:

- **Feature 1:** *Supporting all diagrams, properties and features offered by UML*

As known, a UML model can be constituted of several diagrams. Both, structural and behavioral diagrams contain many features and properties. Therefore, to provide complete solutions that cover all the aspects of the model without losing information, these proposals have to support all the existing, and the recently introduced, UML properties, diagrams and features. To ensure this, the used formalism for detecting and handling inconsistencies needs to be able to express the abstract syntax (e.g. classes, relationships and generalizations between classes, attributes, etc.) and semantics of the modelling language (e.g. call sequences, sequence diagram traces, etc.) such that well-formedness of the user-defined models can be guaranteed.

- **Feature 2:** *Proving the validity of the CCP*

The validity of the CCP should be provable since it adds more credibility to the solution. For this purpose, the proposed method should be formalized and implemented in a CASE tool for it to be used in industrial software development projects.

- **Feature 3:** *Supporting all kinds of inconsistencies*

A CCP that aims to completely solve the problem of inconsistencies in UML models must consider all kinds of inconsistencies as presented in [6]. To reach this aim, the used formalism for detecting and handling inconsistencies needs to define inconsistencies in a precise way and needs to detect them, preferably exhibiting some formal properties such as soundness, completeness and decidability [4].

- **Feature 4:** *Extensibility of the solution (containing extension mechanisms)*

The extensibility of a CCP concerns its ability to easily include new consistency checks required by new UML diagrams or new versions of existing ones. This feature makes a CCP more flexible to deal with any new arising inconsistency.

- **Feature 5:** *Alignment with the OMG standards*

To promote good modeling practices and fully exploit the advantages of models, the Object Management Group (OMG) proposed its own vision of MDE (Model Driven Engineering), under the name of MDA (Model Driven Architecture). This approach is based on several standards such as UML, MOF and XMI. The ultimate goal is to develop UML models across the phases of software development life cycle. To benefit from the advantages of the OMG standards, a CCP that aims to ensure the consistency of the UML models has to be in alignment with these standards. Being in conformance with the OMG standards makes the CCP more suitable for an MDA approach by checking models during the model transformation phases in more efficient and sustainable ways; which lets the CCP be more functional in the industrial software development.

- **Feature 6:** *Integration within a CASE tool*

To take a full advantage from the implemented checks, the CCP has to be suitably integrated within a CASE tool. This later can make the task of detecting inconsistencies easier for modelers by giving the possibility to use and add new checks, and by improving the feedback of models checks reported by the tool.

- **Feature 7:** *Allowing actions for fixing inconsistencies*

A CCP has to provide the modeler with the possibility to fix inconsistencies encountered in the model. For that, the used technique and its underlying formalism must enable defining resolution actions. These actions can depend on the nature of the inconsistency in question or on its causes. Furthermore, executing resolution actions is a highly interactive process. The technique to be used for the resolution has to cope with this interactivity. In addition to that, resolving inconsistencies can introduce new inconsistencies. The used technique must also deal with this dependency.

After carrying out this SLR, we recall that we expect to:

1) Obtain relevant information about the techniques used to manage inconsistencies and their adequate formalisms.

2) Identify strengths and weaknesses in current research concerning checking inconsistencies.
3) Come up with recommendations that should be taken into consideration when conceiving a new proposal for checking inconsistencies.

4.2. Search Criteria

Based on the defined research objectives, we choose search string criteria from well-known available digital libraries including IEEE eXplore, ScienceDirect – Elsevier and Google Scholar. The terms used to search papers were "UML", "model", "inconsistency" and "detection"; in addition to certain synonyms and terms related to the concept of the model within the scope of consistency detection such as "diagram", "consistency", "managing", "checking" and "verification". The logical operators "AND" and "OR" were used to formulate research queries such as ["UML" and "model" and ("inconsistency" or "consistency") and ("detection" or "checking" or "verification")].

We did not focus on the papers' period of publication (starting from the year 2000). Still, we consider the latest work when we found two or more papers adopting the same technique and/or the same formalism. We did the same, when an author has more than one work addressing the same issue being motivated by the following hypothesis:

(H) "We suppose that a recent work considers the limitations of the previous ones and tries to improve them."

4.3. Inclusion and Exclusion Criteria

Based on the aforementioned research questions, we present in what follows the inclusion criteria used for papers selection:

1) The papers only targeting inconsistencies in UML models, (no other models).
2) The papers published starting from the year 2000.
3) The papers written in English.

The exclusion criteria can be summarized as follows:

1) The papers which are not dedicated to UML model consistency were excluded.
2) Surveys, overviews and reviews (Mapping studies and systematic literature reviews) were excluded from the SLR since in general they do not provide new inconsistency detection techniques.
3) The papers targeting only fixing and handling techniques.
4) Duplicate papers and approaches using the same technique/formalism were excluded to avoid similar results.
5) The papers not written in English were excluded.
6) Papers published before 2000 were excluded.

4.4. Search Strategy

Initially, we started our search method based on more than 100 papers collected using the search criteria explained before. Then, we applied our exclusion criteria to avoid irrelevant and duplicated papers based on the title and the keywords. Additional papers did not meet the defined objectives and were excluded after reading the abstract, the introduction and sometimes the body of the paper (as illustrated in Fig. 4).

Finally, we selected 27 papers that almost cover the majority of the techniques/formalisms used to detect inconsistencies. These papers satisfied the defined objectives and inclusion/exclusion criteria. Our study is then based on the final selected papers.

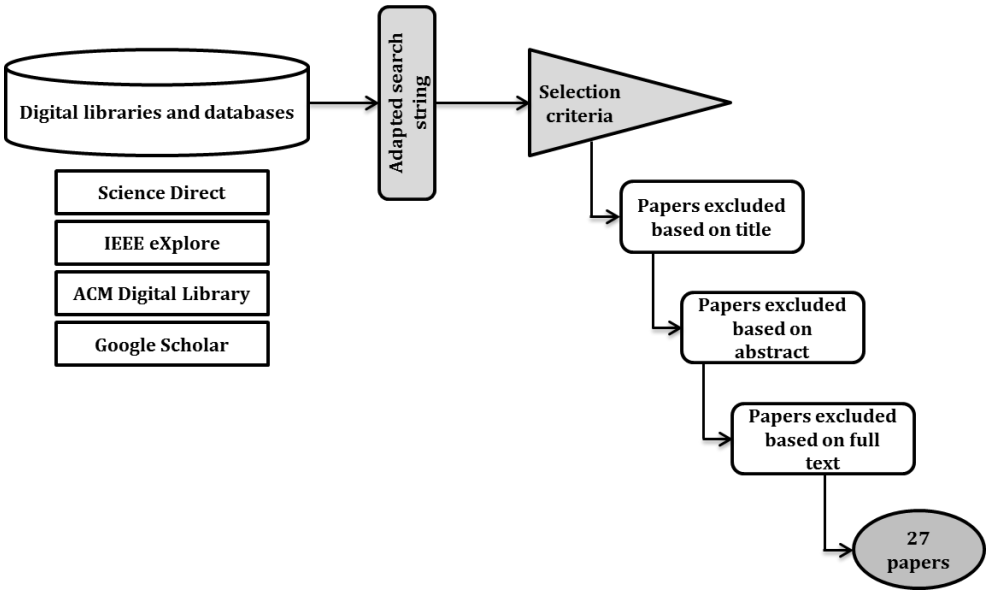

Fig. 4. The flow chart of the search strategy

Besides, we deem it necessary to mention that other digital libraries not listed above such as Scopus may contain some of the initially selected papers since a paper can be indexed in many databases.

5. Results

According to the four research questions, RQ1 through RQ4, the study provides information summarized and organized using tables and charts. Hereafter, we distinguish two collections of proposals, namely the *construction-based* and the *monitoring-based* solutions. The construction-based selected papers are presented in Table 1 referring to their general paradigm, their formalism and typology.

Table1. Construction-Based Techniques

Paradigm	Formalism	Typology	Authors & Ref.	Description
State-Transition	Z	Approach	Singh *et al.* [20]	A practical approach that transforms the use case diagram, class diagram and sequence diagram to Z Schema for capturing both the syntax and semantics for safety critical systems.
	B	Method	Laleau and Polack [21]	Checking inconsistencies between the different diagrams of an IS-UML specification by expressing consistency checks and mapping rules between metamodel concepts using B language.
	Graphs	Theory (+impl.)	Fryz and Kotulski [22]	A graph-based solution that relies on the concept of conjugated graphs with its implementation in the distributed environment using an aedNLC graph grammar.
	Petri Nets	Approach	Rajabi and Lee [23]	A mutual integration between UML diagrams and Colored Petri Nets modeling languages to support model changes. Set of rules are included to check the consistency and the integrity of the models.
	Automata / SPIN	Process	Knapp and Wuttke [19]	To ensure the consistency of the UML state machine diagrams, the authors translate the UML 2.0 interaction into interaction automata. The obtained result is in turn translated into SPIN using the HUGO/RT tool.
Process Algebra	CSP	Method	Cabot *et al.* [24]	A method to check UML class diagrams extended with OCL constraints. The method translates the UML/OCL model into a Constraint Satisfaction Problem (CSP) to deal with complex invariants which may include numerical constraints.
	π-calculus	Approach	Belghiat *et al.* [25]	Using the π-calculus formal theory and tools to capture and verify the dynamic behavior of systems represented through UML diagrams
	RT-LOTOS	Extension (profiling)	Apvrille *et al.* [26]	A UML 1.5 profile named TURTLE (Timed UML and RT-LOTOS Environment) endowed with a formal semantics given in terms of RT-LOTOS to solve logical errors and timing inconsistencies in the context of space-based embedded software.

	Maude	Approach	Lucas *et al.* [39]	A proposal based on the algebraic formalization of the UML metamodel diagrams in Maude. All the UML specifications are integrated within the same formalism to ensure the inter-diagram consistency.
Logic	SPIN	Approach	Zhao *et al.* [27]	Investigation of a SPIN model checker based approach to check the consistency between the sequence and statechart diagrams. They propose a variant of automata, called Split Automata, to efficiently translate the statechart diagrams to SPIN in order to deal with the hierarchy structure of statechart diagrams.
	NuSMV	Approach	Dos Santos *et al.* [28]	Using the NuSMV model checker to detect UML model inconsistencies. A technique that allows transforming up to three different UML behavioral diagrams (sequence, state machines, and activity) into a single Transition System to support Model Checking of software developed in accordance with UML.
	CLP	Method	Malgouyres and Motet [29]	A unified checker taking advantage of meta-modeling techniques to translate UML models to CLP (Constraint Logic Programming) clauses. Consistency rules are also expressed using CLP to let the CLP solver automatically detect inconsistencies.
	PVS	Approach	Paige *et al.* [30]	An approach for metamodeling, model conformance, and multi-view consistency checking, including contracts. They used for this aim the Prototype Verification System (PVS) to check the consistency of the UML models taking into account a trade-off between the completeness and the level of automation of the solution.
	DL	Tool	Straeten *et al.* [31]	Formalizing the behavior specified by UML 2.0 sequence and protocol state machine diagrams. They show how the reasoning capabilities of Description Logics (DL), a decidable fragment of first-order logic, can be used in a natural way to detect behavior inconsistencies of a refined model, and also detect the behavior preservation violations during model refactoring.
	rCOS	Algorithm	Long *et al.* [40]	An algorithm for checking the consistency of class and sequence diagrams based on rCos definitions. An OO language equipped with an observation-oriented semantics and a refinement calculus based on the Hoare.
	Alloy	Method	Nimiya *et al.* [41]	A method for detecting inconsistencies between state machine and communication diagrams using Alloy. This later is used to express system behaviors, message sequences and a consistency property.
	Action-LTL	Framework	Banerjee *et al.* [42]	A Dynamic Property Verification (DPV) framework that validates UML designs in accordance to Rhapsody simulation semantics. In this platform, assertions over data attributes and events of UML models are written using Action-LTL.
	Prolog	Approach	Khai *et al.* [43]	An approach for consistency checking of class and sequence diagrams based on Prolog language. The consistency checking rules and the UML models are represented in Prolog to be checked according to the Prolog's reasoning engine.

The monitoring-based works in Table 2 are presented according to the used technique and its typology.

Table 2. Monitoring-Based Techniques

Technique	Typology	Authors & Ref.	Description
OCL	Method	Przigoda *et al.* [32]	An automatic method to aid designers during debugging of their model. The method analyzes constraints causing contradictions and determines reasons explaining the contradiction in an inconsistent UML/OCL model.
	Extension	Berkenktter [33]	Reusing the existing OCL-based static semantics of UML and strengthens them by rectification and extension. The approach supports the usage of profiles as long as these specify their static semantics on OCL.
	Method	Kalibatiene *et al.* [34]	A rule-based method for consistency checking in IS models, which is implemented to check consistency in UML diagrams. The consistency rules expressed in this work associate meta-elements from different aspects of models despite the fact that they are directly associated in a metamodel.
	Extension (Profiling)	Ober and Dragomir [45]	A set of consistency rules formalized in OCL and defined within the OMEGA UML profile. These rules are applicable to hierarchical component models.
XMI	Method	Chen and Motet [35]	Expressing consistency rules using controlling grammar in XML format C-Control. These consistency rules and guidelines are formalized as controlling grammars that can be implemented as a parser, which can automatically verify the rules on a UML model in XMI format.
SQL triggers	Method	Sapna and Mohanty [36]	Dealing with structural inconsistencies between the use cases, activity, collaboration, statechart and class diagrams by using OCL rules converted to

			SQL triggers applicable across tables that store the UML diagrams.
Language/checker available in IBM rational Rose	Tool	Egyed [37]	Using the language/checker available in IBM rational Rose to define constraints in order to demonstrate that a tool can assist the modeler to detect unintentional side-effects, locating choices for fixing inconsistencies to change the design model.
EVL	Method	Allaki *et al.* [38]	A method for checking the consistency of UML models, based on formal constraints defined at the meta-model of UML. These constraints are described using Epsilon Validation Language (EVL) by matching related diagrams features at the meta-level. EVL also helps repair and correct the inconsistencies being detected.
USE tool	Approach	Gogolla [46]	An inspection technique that uses animation and certification of the USE tool to execute UML models and check OCL constraints.

In order to provide effective answers to the RQ4, we regroup in Table 3 all the selected papers and present, according to our understanding, their compliance with the seven features we specified above.

Table 3. Compliance of the Techniques with the Specified Quality Features

Features / **Authors & Ref.**	**Feature 1** Supporting all UML features	**Feature 2** Proving the validity of the CCP	**Feature 3** Supporting all kinds of inconsistencies	**Feature 4** Extensibility of the solution	**Feature 5** Alignment with the OMG standards	**Feature 6** Integration within a CASE tool	**Feature 7** Actions for fixing inconsistencies
Singh et al. [20]	×	✓	×	×	×	✓	×
Laleau and Polack [21]	✓	✓	×	✓	×	×	×
Fryz and Kotulski [22]	×	✓	×	×	×	×	×
Rajabi and Lee [23]	✓	✓	×	×	×	✓	×
Knapp and Wuttke [19]	×	✓	×	×	×	✓	×
Cabot et al. [24]	×	✓	×	×	×	✓	×
Belghiat et al. [25]	✓	✓	✓	×	×	✓	×
Apvrille et al. [26]	×	✓	×	✓	×	✓	×
Lucas et al. [39]	✓	✓	✓	✓	×	×	×
Zhao et al. [27]	×	✓	×	✓	×	✓	×
Dos Santos et al. [28]	×	✓	×	×	×	✓	×
Malgouyres and Motet [29]	✓	✓	✓	×	×	×	×
Paige et al. [30]	✓	✓	✓	✓	×	✓	×
Straeten et al. [31]	✓	✓	✓	×	×	✓	×
Long et al. [40]	×	✓	×	✓	×	×	×
Nimiya et al. [41]	×	✓	×	✓	×	×	×
Banerjee et al. [42]	×	✓	×	✓	×	✓	×
Khai et al. [43]	×	✓	×	✓	×	✓	×
Przigoda et al. [32]	×	×	×	✓	✓	✓	×
Berkenktter [33]	✓	×	×	✓	✓	✓	×
Kalibatiene et al. [34]	✓	×	✓	✓	✓	✓	×
Ober and Dragomir [45]	×	×	×	✓	✓	✓	×
Chen and Motet [35]	✓	×	✓	✓	✓	×	×
Sapna and Mohanty [36]	✓	×	×	✓	×	×	×
Egyed [37]	✓	×	✓	✓	×	✓	✓
Allaki et al. [38]	✓	×	✓	✓	×	✓	✓
Gogolla [46]	×	×	×	✓	×	✓	×

The presented results show that the two-thirds of the proposals' types were *methods* and *approaches* (66%).

On the other hand, almost half of the proposals (48%) support all UML diagrams and features but only (33%) cover all the kinds of inconsistencies. The two-thirds of the proposals are formally provable (66%) and almost the same ratio is extensible (66%) and automated using Case-tools (70%). However, only (7%)

propose fixing actions and only (19%) are in conformance to the OMG standards. Fig. 5 summarizes these findings.

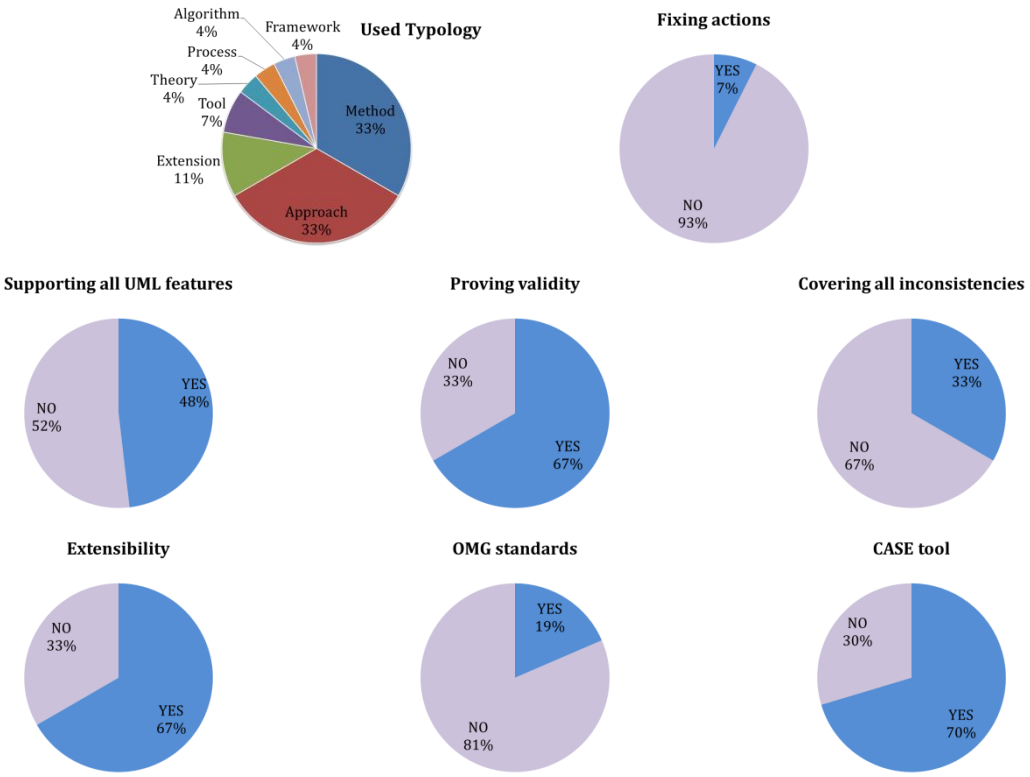

Fig. 5. Results in numbers of the RQ1 and RQ4

The Fig. 6 through 10 and 11 through 16 present additional detailed results on, respectively, the construction-based and the monitoring-based techniques.

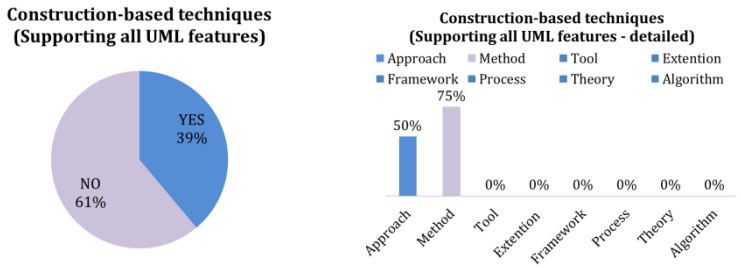

Fig. 6. Supporting all UML diagrams and features in construction-based techniques

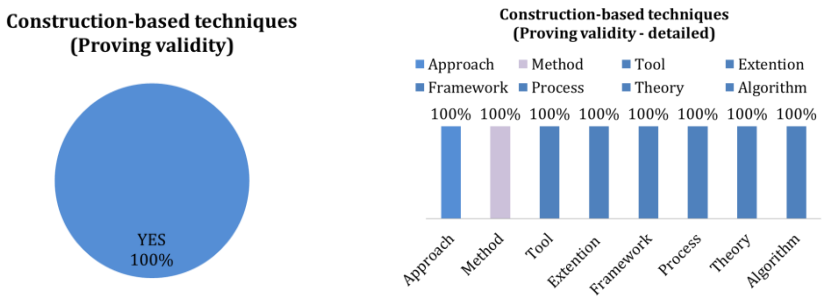

Fig. 7. Proving the validity of the proposals in construction-based techniques.

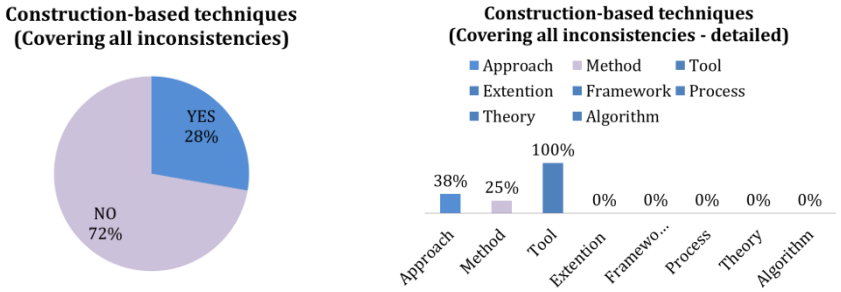

Fig. 8. Supporting all kinds of UML model inconsistencies in construction-based techniques.

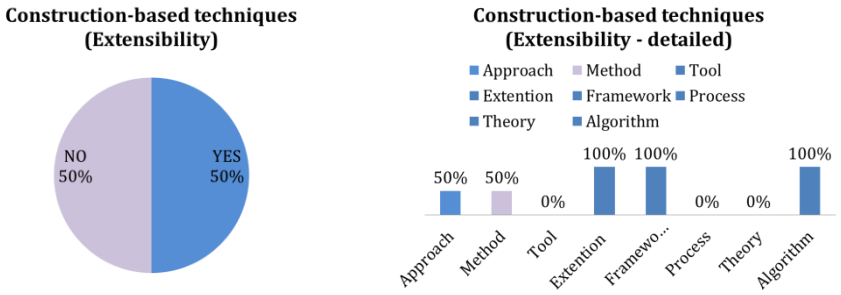

Fig. 9. Extensibility in construction-based techniques.

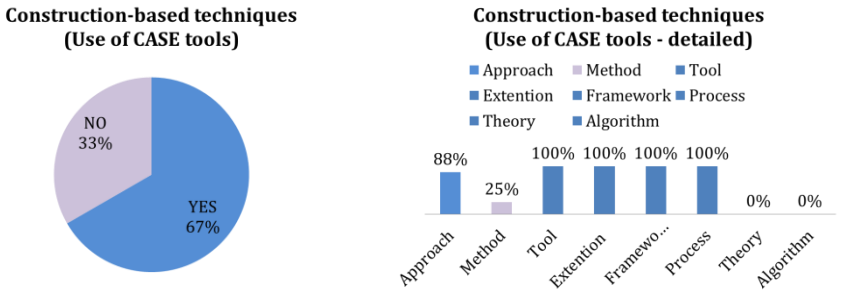

Fig. 10. Use of CASE tools in construction-based techniques.

As presented in the figures above, all the construction-based techniques are formalized (which is proof of validity of these solutions). However, few of them are complete in terms of extensibility and the coverage of both potential inconsistencies and the UML diagrams commonly used. Further, some proposals are not automated with case-tools and none of them proposed fixing actions to repair the detected inconsistencies.

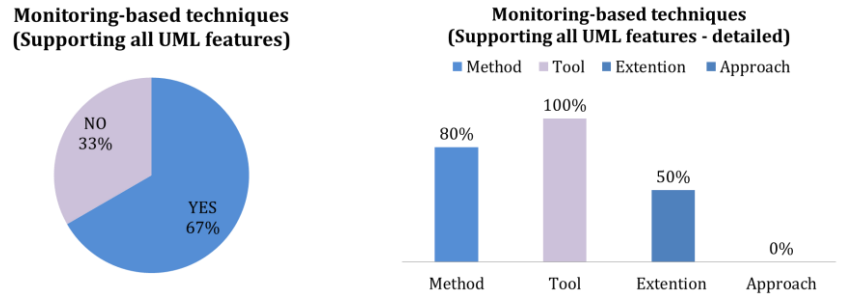

Fig. 11. Supporting all UML diagrams and features in monitoring-based techniques.

In contrast, the monitoring-based techniques are mostly automated and some of them support fixing actions. They are extensible and in accordance to the OMG standards. However, there is no formal guaranty for their validity and some of them may not support all the potential arising inconsistencies.

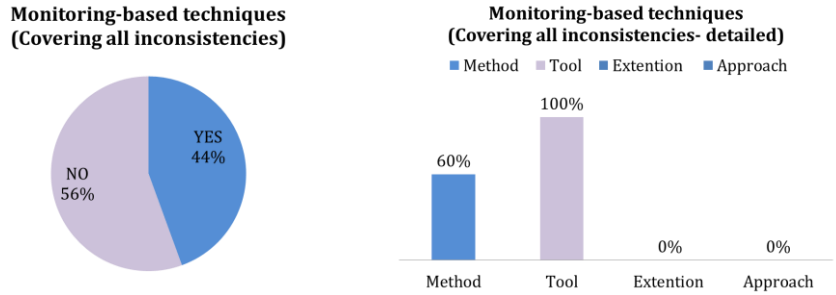

Fig. 12. Supporting all kinds of UML model inconsistencies in monitoring-based techniques.

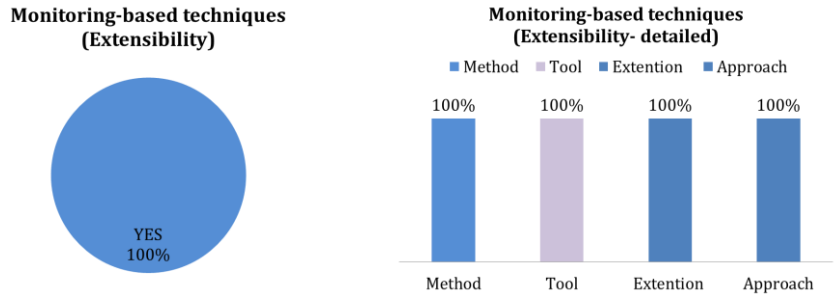

Fig. 13. Extensibility in monitoring-based techniques.

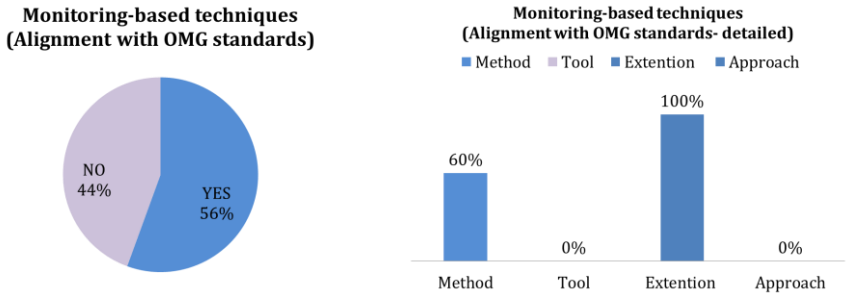

Fig. 14. Alignment with OMG standards in monitoring-based techniques.

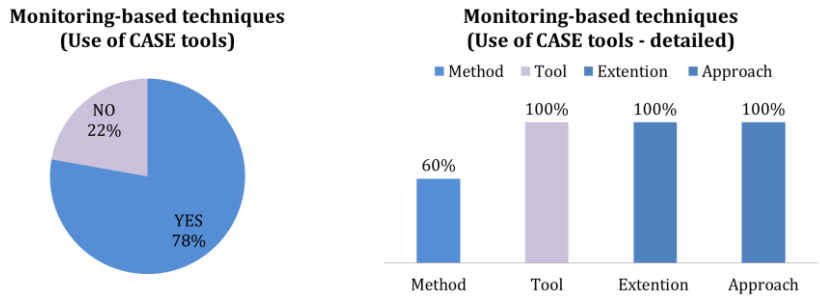

Fig. 15. Tool support in monitoring-based techniques.

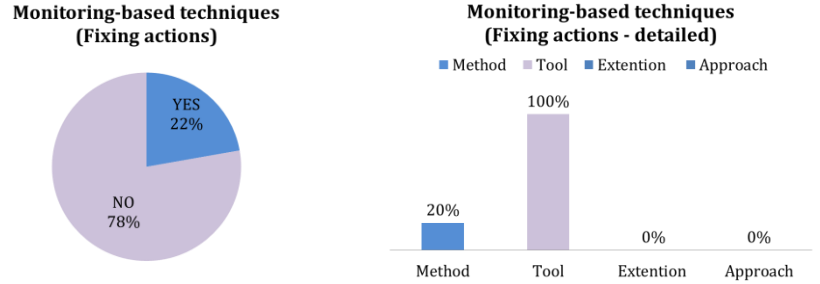

Fig. 16. Fixing actions in monitoring-based techniques.

6. Discussion

As mentioned before, several techniques have been proposed in literature to detect the inconsistencies in UML model. In our study, we classified these approaches into two main categories, according to the followed consistency strategy, namely *consistency by construction* and *consistency by monitoring*. Hereafter, a discussion about the main advantages and limitations of the techniques studied in each category is provided. The discussed findings are issued from the previous results.

6.1. Construction-Based Techniques

The main idea of this detecting inconsistency approach is first to transform the semi-formal UML models into a formal representation. Then, inconsistencies are detected using the inference mechanisms of the target formal language. These techniques are classified according to the underlying formal paradigm: State transitions, Process Algebra and Logic.

Overall, construction-based inconsistency checking techniques take advantages from the underlying formal languages that provide solid mathematical foundation, proof and tools such as theorem provers and model checkers. They also add more precision to UML models by avoiding ambiguities and misinterpretations in dealing with these models. In contrast, construction-based inconsistency checking techniques do not support all the properties of the UML models; they are limited only to those properties that can be expressed and analyzed by the target formal language of the translation, which means that we can have some features of some UML diagrams not able to be expressed in some formal languages. And then, this category of techniques is not complete due to the loss of information expressed in UML during the transformation process. For these reasons, these techniques do not have the ability to deal with all the categories of UML model inconsistencies. Furthermore, the used CASE tools, related to the formal techniques, generally suffer from the inability of tracing the detected inconsistencies back in the original UML diagrams. In other words, when a property is violated in the formal language, tools rarely report the elements of the UML models that might be the source of the encountered inconsistency. Thus, it is up to the designer to interpret and analyze the reported information in order to find out the source of the problem back in the input UML diagrams. Moreover, most of the construction-based techniques and languages are not commonly known among the industrial development communities. They are difficult to use due to the lack of documentation and the strong mathematical background required for applying them. This can lead to an updating non-regularity of these formal paradigms, which negatively affect the extensibility of the proposals that are based on them.

6.2. Monitoring-Based Techniques

Monitoring-based techniques are primarily based on constraints defined at the UML metamodel to deal with inconsistencies in UML models. These constraints are described in terms of rules expressed in OCL [8] or any other language such as XMI [9] or the language/checker available in IBM rational Rose [10]. Following these constraints, the inconsistencies in UML models are detected by support tools either when the models are being built or just before starting the implementation of these models.

Unlike construction-based inconsistency checking techniques, most of the monitoring-based proposals have the advantage of preserving all the information expressed in the UML models being studied. They also add more precision to the model by making it more expressive through the constraints defined at the metamodel. Moreover, they provide extensible solutions that take into account the possibility to include new consistency checks for new arising inconsistencies; which promotes the usability and the maintainability of these proposals. Nevertheless, constraint-based techniques suffer from a lack of formality, which limits their proof. Also, the quality of these techniques strongly depends on the quality and the coverage of the constraints defined on the models. Indeed, if the constraints used do not cover all the

potential inconsistencies some of these might have left undetectable in the models. Finally, existing monitoring-based techniques generally deal with static aspects of the UML models and are limited to checking inconsistencies in a single diagram, which compromise their efficiency. For instance, it would be hard, using some monitoring-based techniques, to check inconsistencies arising between two or more UML diagrams, especially when it concerns the behavioral aspects of the model.

7. Conclusion

A great number of papers have been proposed to detect inconsistencies among UML diagrams. Even if a considerable deal of quality work has been done in these works, yet some gaps and limitations still exist. Moreover, other studies have been dedicated to summarize and analyze these works according to different research perspectives. In this paper, we followed a systematic protocol using four research questions (RQs) to provide knowledge related to 27 selected proposals. We focused on their typology, research strategy, the used technique or formalism (and its adequate paradigm), in addition to their accordance to a set of quality features we proposed. The general results showed that there is a certain improvement's margin for the construction-based solutions concerning the effective use of support tools, adding fixing actions and making these solutions extensible and in accordance to the OMG standards. On the other hand, the techniques using a monitoring-based strategy lack of validity and the support of all the existing categories of inconsistencies.

Motivated by encouraging the researchers' community to produce industrial ready research solutions, we recommend considering the set of features we provide as requirements that should be met by every new consistency checking proposal. The aim is to provide extensible, provable, and automatized solutions that support all kinds of inconsistencies affecting the different features offered by the OMG standards in UML. More explicitly, we suggest using techniques/formalisms that are able to express the abstract syntax and the semantics of all the UML features such that well-formedness of the defined models can be guaranteed. Also, the inconsistency management technique/formalism needs to be able to precisely define, detect and fix all the existing inconsistencies using some formal properties in order to guarantee a formal proof concerning the validity of the solution. Moreover, the used technique/formalism has to enable management of interactive inconsistency resolutions. Additional tool support requirements can also be identified to provide useful feedback to the modelers or to add extensibility actions to deal with new arising inconsistencies. In another line of thought, the typology of the proposals has to be wisely chosen to help putting into action the aforementioned features and requirements.

Currently, we are working to provide new contributions on managing inconsistencies by focusing on the handling activities. In fact, the results of our study showed the need of including repairing actions to help the modeler correctly and wisely act when facing a detected inconsistency. Considering this as a starting point, we will work on generating a set of small, complete and correct repair actions by eliminating false and non-minimal repairs. Second, we plan to investigate mathematical techniques based on decision theory to help the designer select the more appropriate alternative from the generated repair actions. Third, the whole method will be initially evaluated and discussed using a comprehensive case study that contains patterns involving a set of tricky examples of inconsistencies and usage rules that cover a larger number of expressive UML diagrams.

References

[1] Huzar, Z., Kuzniarz, L., Reggio, G., & Sourrouille, J. L. (2004). Consistency problems in UML based software development. *UML Modeling Languages and Applications*, UML 2004 Satellite Activities, Lisbon, *3297*, 1-12.

[2] Allaki, D., Dahchour, M., & En-nouaary, A. (2014). A new taxonomy of inconsistencies in UML models: Towards better MDE. *Proceedings of the 9th International Conference on Intelligent Systems: Theories*

and Applications.

[3] Finkelstein, A., Spanoudakis, G., Till, D. (1996). Managing interference. Joint proceedings of second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops. ACM Press.

[4] Straeten, R. V. D. (2005). Inconsistency management in model-driven engineering: An approach using description logics. Ph.D. thesis, System and Software engineering lab, Department of Computer Science, Faculty of Science, Vrije Universiteit Brussel, Belgium.

[5] Allaki, D., Dahchour, M., & En-nouaary, A. (2016). Detecting and fixing UML model inconsistencies using constraints. *Proceedings of the 4th IEEE International Colloquium on Information Science and Technology*.

[6] Allaki, D., Dahchour, M., & En-nouaary, A. (2015). A new taxonomy of inconsistencies in UML models with their detection methods for better MDE. *International Journal of Computer Science and Applications, Technomathematics Research Foundation*, *12(1)*, 48-65.

[7] Habrias, H., & Frappier, M. (2006). Software specification methods..

[8] Object constraint language. Retrieved April 6, 2017, from the website: http://www.omg.org/spec/OCL/2.4/

[9] XML metadata interchange. Retrieved April 6, 2017, from the website: http://www.omg.org/spec/XMI/2.4.1/

[10] Egyed, A. (2007). Fixing inconsistencies in UML design models. *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*.

[11] Spanoudakis, G., & Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. *Handbook of Software Engineering and Knowledge Engineering*.

[12] Usman, M., Nadeem, A., Kim, T. H. *et al*. (2008). A survey of consistency checking techniques for UML models. *Advanced Software Engineering and Its Applications*.

[13] Lucas, F. J., Molina, F., & Toval, A. (2009). A systematic review of UML model consistency management. *Information and Software Technology*.

[14] Torre, D., Labiche, Y., Genero, M., & Elaasar, M. (2016). A systematic identification of consistency rules for UML diagrams. Carleton University, Technical Report SCE-15-01.

[15] Bashir, R. S., Lee, S. P., & Khan, S. U. R. *et al*. (2016). UML models consistency management: Guidelines for software quality manager. *International Journal of Information Management*.

[16] Knapp, A., & Mossakowski, T. (2016). Multi-view consistency in UML.

[17] Thalanki, P. K., & Maddukuri, V. K. (2013). Classifying research on UML model inconsistencies with systematic mapping. Unpublished Master thesis, School of Computing, Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden

[18] Mens, T., & Straeten. R. V. D. (2003). Simmonds, maintaining consistency between UML models with description logic tools. *ECOOP Workshop on Object-Oriented Reengineering*.

[19] Knapp, A., & Wuttke, J. (2007). Model checking of UML 2.0 interactions. *Reports Rev. Sel. Papers Ws.s Symp.s MoDELS 2006*.

[20] Singh, M., Sharma, A. K., & Saxena, R. (2016). Formal transformation of UML diagram: Use case, class, sequence diagram with Z notation for representing the static and dynamic perspectives of system. *Proceedings of the International Conference on ICT for Sustainable Development*.

[21] Laleau, R., & Polack, F. (2008). Using formal metamodels to check consistency of functional views in information systems specification. *Information & Software Technology*, *50*, 797-814.

[22] Fryz, L. & Kotulski, L. (2007). Assurance of system consistency during independent creation of UML diagrams. *Proceedings of the International Conference on Dependability of Computer Systems*.

[23] Rajabi, B., & Lee, S. P. (2014). Consistent integration between object oriented and coloured petri nets models. *Int. Arab J. Inf. Technol*.

[24] Cabot, J., Clariso, R., & Riera, D. (2014). On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*.

[25] Belghiat, A., Chaoui, A., & Beldjehem, M. (2016). Capturing and verifying dynamic systems behavior using UML and pi-calculus. *Theoretical Information Reuse and Integration*, Springer International Publishing, 59-84.

[26] Apvrille, L., Courtiat, J.-P., Lohr, C., deSaqui-Sannes, P. (2004). TURTLE: A real-time UML Profile supported by a formal validation toolkit. *IEEE Transactions on Software Engineering (TSE)*.

[27] Zhao, X., Long, Q., Qiu, Z. (2006). Model checking dynamic UML consistency. *Proceedings of the 8th International Conference on Formal Engineering Methods Formal Methods and Software Engineering.*

[28] Santos, L. B. R. D, Júnior, V. A. S., & Vijaykumar, N. L. (2014). Transformation of UML behavioral diagrams to support software model checking.

[29] Malgouyres, H., & Motet, G. (2006). A UML model consistency verification approach based on meta-modeling formalization. *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)* (pp. 1804-1809).

[30] Paige, R. F., Brooke, P. J., & Ostroff, J. S. (2007). Metamodel-based model conformance and multiview consistency checking. *ACM Transactions on Software Engineering and Methodology*.

[31] Straeten, R. V. D., Jonckers, V., & Mens, T. (2007). A formal approach to model refactoring and model refinement. *Software and System Modeling*, *6(2)*.

[32] Przigoda, N., Wille, R., & Drechsler, R. (2016). Analyzing inconsistencies in UML/OCL models. *Journal of Circuits, Systems and Computers*, *25(3)*.

[33] Berkenktter, K. (2008). Reliable UML models and Profiles. *Electronic Notes in Theoretical Computer Science*.

[34] Kalibatiene, D., Vasilecas, O., & Dubauskaite, R. (2013). Ensuring consistency in different IS models – UML case study. *Baltic Journal of Modern Computing*.

[35] Chen, Z., & Motet, G. (2009). A language-theoretic view on guidelines and consistency rules of UML. *Proceedings of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*.

[36] Sapna, P. G., & Mohanty, H. (2007). Ensuring consistency in relational repository of UML models. *Proceedings of the 10th International Conference in Information Technology*.

[37] Egyed, A. (2007). Fixing inconsistencies in UML design models. *Proceedings of the 29th International Conference on Software Engineerin*.

[38] Allaki, D., Dahchour, M., & En-Nouaary, A. (2016). A constraint-based approach for checking vertical inconsistencies between class and sequence UML diagrams. *Proceedings of the 18th International Conference on Enterprise Information Systems* (pp. 441-447).

[39] Martínez, F. J. L., & Álvarez, J. A. T. (2005). A precise approach for the analysis of the UML models consistency. *Proceedings of the Perspectives in Conceptual Modeling*.

[40] Long, Q., Liu, Z., Li, X., & He, J. (2005). Consistent code generation from UML models. *Proceedings of the 16th Austral. Software Engineering Conf. (ASWEC'05)*.

[41] Nimiya, A., Yokogawa, T., Miyazaki, H., Amasaki, S., Sato, Y., & Hayase, M. (2010). Model checking consistency of UML diagrams using alloy. *WASET Intl. J. Comp., Electr., Autom., Contr*.

[42] Banerjee, A., Ray, S., Dasgupta, P., Chakrabarti, P. P., Ramesh, S., & Ganesan, P. V. V. (2012). A dynamic assertion-based verification platform for validation of UML designs. *ACM SIGSOFT Softw. Eng. Notes*.

[43] Khai, Z., Nadeem, A., & Lee, G.-s. (2011). A prolog based approach to consistency checking of UML class

and sequence diagrams. *Proceedings of the Intl. Conf.s ASEA, DRBC, EL as part of Future Generation Information Technology Conf*.

[44] Kitchenham, B. (2007). Guidelines for performing systematic literature reviews in software engineering, EBSE technical report EBSE-2007-01, software engineering group, school of computer science and mathematics. Keele University, UK and Department of Computer Science, University of Durham, Durham, UK, 2007.

[45] Ober, I., & Dragomir, I. (2011). Unambiguous UML composite structures: The OMEGA2 experience. In: *Proceedings of the 37th Conf. Current Trends in Theory and Practice of Computer Science*.

[46] Gogolla, M., Büttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program*.

Driss Allaki received his master degree in computer science applied to offshore development in the Faculty of Sciences of Rabat, Morocco, in 2012. He is currently a PhD student in the doctoral center of INPT "Institut National des Postes et Télécommunications" Rabat, Morocco. His current research topics are on conceptual modeling and information systems management.

Mohamed Dahchour received the doctoral (2001) in computer science from Ecole polytechnique de Louvain, Université catholique de Louvain, Belgium. Currently, he is a full professor of computer science at the National Institute of Posts and Telecommunications (INPT), Morocco. He is also the head of the Department of Mathematics and Informatics at INPT and the leader of a research team on software systems engineering and information systems management. His scientific interests include software engineering, conceptual modeling, web semantic, distributed systems, and information systems management.

Abdeslam En-Nouaary received his engineer degree in computer engineering, option data communication and computer networks from ENSIAS (École Nationale Supérieure d'Informatique et d'Analyse des Systèmes), Rabat, Morocco, in 1996, the M. Sc. and Ph. D. degrees in computer science from the University of Montreal in 1998 and 2001 respectively. Dr. En-Nouaary is currently an associate professor at INPT (Institut National des Postes et Télécommunications), Rabat, Morocco. Before joining INPT in 2008, Dr. En-Nouaary has been an associate professor at the Electrical and Computer Engineering Department of Concordia University, Montreal, Canada, From 2001 to 2008. His main research interests are modeling and validation of distributed, realtime, and embedded systems.