A systematic identification of consistency rules for UML diagrams

Damiano Torre, Yvan Labiche, Marcela Genero, and Maged Elaasar

Abstract—UML diagrams describe different views of one piece of software. These diagrams strongly depend on each other and must therefore be consistent with one another, since inconsistencies between diagrams may be a source of faults during software development activities that rely on these diagrams. It is therefore paramount that consistency rules be defined and that inconsistencies be detected, analyzed and fixed. The relevant literature shows that authors typically define their own UML consistency rules, sometimes defining the same rules and sometimes defining rules that are already in the UML standard. The reason might be that no consolidated set of rules that are deemed relevant by authors can be found to date. The aim of our research is to provide a consolidated set of UML consistency rules and obtain a detailed overview of the current research in this area. We therefore followed a systematic procedure in order to collect and analyze UML consistency rules. We then consolidated a set of 116 UML consistency rules (avoiding redundant definitions or definitions already in the UML standard) that can be used as an important reference for UML-based software development activities, for teaching UML-based software development, and for further research.

Index Terms—Software/Program Verification, Model checking

1 INTRODUCTION

MODEL Driven Architecture (MDA) [1] is an approach to software development by the Object Management Group (OMG) that promotes the efficient use of models throughout software development phases, from requirements to analysis, design, implementation, and deployment [2]. Much attention has been paid to MDA by academia and industry in recent years [3], [4], [5], which has resulted in models gaining more importance in software development. The Unified Modeling Language (UML) [6] is the OMG specification that is most frequently used and is the de-facto standard modeling language for software modeling and documentation [7]. It is the prefered modeling language for implementing MDA, although it is not intended to be used in every single software development project [8]. The UML provides 14 diagram types [6] with which to describe a software system from different perspectives (e.g., structure and behavior) and abstraction levels (e.g., analysis and design), which, among other things, help deal with complexity and distribute responsibilities.

Since the various UML diagrams in a model typically describe different views of one, and only one, software system under development, they strongly depend on each other and hence need to be consistent with one another. As the UML is not a formal specification, inconsistencies between different diagrams may arise [9]. When UML diagrams communicate contradicting or conflicting syntax or semantics, the diagrams are said to be inconsistent [10]. Such inconsistencies may be a source of faults in software systems down the road [9], [10]. It is therefore of paramount importance that the consistency between diagrams be defined and that inconsistencies be routinely detected, analyzed and fixed [11].

In an attempt to obtain an accurate picture of current research in the area of UML diagram consistency, we conducted a systematic mapping study [8], which resulted in a number of findings concerning publications in the domain, such as: the majority of the publications discussing UML diagram consistency refer to consistency between the class diagram and sequence diagrams; only a very few publications discuss a consistency dimension other than horizontal consistency, also referred to as intra-model consistency, and the UML 2.0 is the most frequently used version in papers discussing consistency. Although the aim of this mapping study was to discover frequencies of publications in different categories and was not specifically to study published UML diagram consistency rules, the study provided anecdotal evidence that: (1) authors routinely define consistency rules for different UML based software engineering activities; (2) some authors tend to define similar, often identical rules, over and over again, and (3) some authors are not aware that the UML standard specification defines consistency rules (a.k.a., well-formedness rules). We also observed that even though many researchers have either explicitly or implicitly proposed rules with which to detect inconsistencies in the UML, no consolidated set of UML consistency rules had been published to date. By the word “consolidated”, we mean the process of combining a number of UML consistency rules that other authors feel are important for UML into a single,

- *D. Torre is with the Department of Systems and Computer Engineering, Carleton University, ON K1S5B6, Canada, and the Departamento de Tecnologías y Sistemas de Información de la Universidad de Castilla-La Mancha, CO 13071, Spain. E-mail:dctorre@sce.carleton.ca.*
- *Y. Labiche is with the Department of Systems and Computer Engineering, Carleton University, ON K1S5B6, Canada. E-mail: labiche@sce.carleton.ca.*
- *M. Genero iswiththe Departamento de Tecnologías y Sistemas de Información de la Universidad de Castilla-La Mancha, CO 13071, Spain. E-mail:marcela.genero@uclm.es.*
- *M. Elaasar is with Department of Systems and Computer Engineering, Carleton University, ON K1S5B6, Canada. E-mail:melaasar@gmail.com.*

coherent set. We believe that, even though the rules required may be domain, organization, or project specific, this lack of a consolidated list of UML consistency rules forces researchers to define the rules that they rely on for their own research [9], thus resulting in some rules being defined over and over again. This motivated our main research objective in this paper, which is to identify a consolidated set of consistency rules for UML diagrams. Identifying such a set of UML consistency rules has several benefits: it will be a reference for practitioners, educators, and researchers alike; it will provide guidance as to which rules to use in each context; and it will highlight which areas are more developed and which need further work. Moreover, the consolidated set of rules could be a good input to the UML revision task force for inclusion in a forthcoming revision of the standard.

In order to identify such a consolidated set of UML Consistency rules, we decided to revisit the primary studies obtained in our systematic mapping study [8], which led us to collect and analyze the rules from the publications in question. This was done by following a systematic procedure inspired by well-known guidelines for empirical research in software engineering [12]. These guidelines provide a systematic and rigorous procedure that is used to identify the quantity of existing research related to a specific research topic.

The research work presented in this paper is therefore an extension of the systematic mapping study published in [8]. Of the 95 primary studies identified during the mapping study, we collected 603 consistency rules, including duplicates and rules already in the UML standard, which we then analyzed in order to answer a number of research questions:

RQ1) What are the existing UML consistency rules?
RQ2) Which types of consistency problems are tackled in the existing rules?
RQ3) What types of UML diagrams are involved in UML consistency rules?
RQ4) For what software engineering activities are UML consistency rules used?
RQ5) Which UML diagram elements are involved in UML consistency rules?

We subsequently employed a systematic refinement process with the objective of removing duplicates and rules already in the UML standard, as explained later in this paper, and we eventually compiled a consolidated set of 116 rules. We believe that this is an important, long-awaited contribution that will be of interest to researchers, educators and practitioners alike. Another important contribution is a discussion, structured by means of our research questions, of the current state of the research and practice as regards UML diagram consistency and future directions in which further research is needed.

The remainder of this document is structured as follows: Section 2 provides a summary of the related work; the definition of the systematic procedure that we followed is presented in Section 3; Section 4 presents the execution of the study; the results are described in Section 5; a discussion of the threats to validity is presented in Section 6; and finally, our conclusions and outlines of future work are shown in Section 7.

2 Related Work

Since this paper concerns the systematic identification of UML consistency rules, related work pertains to systematic discussions of UML consistency. We have found five pieces of work that fit this description. We first discuss the only review [13] that presented UML consistency rules, and another four reviews regarding UML consistency which we considered that it would be important to include.

The work of Kalibatiene and colleagues [13] is, to the best of our knowledge, the closest piece of work to our research since the authors searched for, collected and presented UML diagram consistency rules. They obtained 50 rules by reviewing eight articles, which they did not select by following a systematic protocol (Systematic Literature Review or Systematic Mapping Study): no mention was made of the processes used to obtain the articles or to include/exclude these documents. After conducting our research, we confirmed that all of their 50 rules are included in our final set of 116 rules.

The following four pieces of work are reviews whose purpose is to increase the body of knowledge on UML diagram consistency, although not necessarily on UML consistency rules.

Spanoudakis and Zisman [11] presented a literature review on the problem of managing inconsistencies in software models in general, but not specifically UML models. The authors presented a conceptual framework that views inconsistency management as a process, which incorporates activities with which to detect overlaps and inconsistencies between software models, diagnose and handle inconsistencies, track the information generated along the way, and specify and monitor the exact means used to carry out each of these activities. They then surveyed all the research works published prior to 2001 (the year of publication of their work) that address one or more of the aspects of their conceptual framework/process.

Usman and colleagues [3] presented a literature review of consistency checking techniques for UML models. The authors argued that formalizing UML models is preferable to verifying consistency because this helps remove ambiguities and enforce consistency. They briefly reviewed 17 articles, which represent less than a quarter of the number of articles considered in our work (95): the two pieces of research have only ten studies in common since our search has a different objective. During this survey, the authors did not follow any Systematic Literature Review (SLR) or Systematic Mapping Study (SMS) protocols, and they simply provided an initial summary of their findings concerning UML consistency checking techniques (not consistency rules).

Lucas, Molina, and Toval [5] presented an SMS on UML consistency management in which they reviewed 44 papers published between 2001 and 2007, focusing on the consistency across two or more UML diagrams. This work is different from ours in several ways. The first important difference is that they did not present any UML consistency rules during their review. The second main difference is the purpose: they focused solely on the management of UML (in)consistencies, i.e., they focused on the techniques used to identify and fix inconsistencies, without providing a detailed discussion of which inconsistencies had to be identified and fixed. In contrast, our work focuses on those inconsistencies that need to be identified and fixed. A direct consequence of this difference is that we considered a broader number of articles in order to consolidate the set of UML consistency rules (95 articles rather than 44), and approximately half of their studies (24) are not considered in our work.

Ahmad and Nadeem [14] presented a literature review restricted to Description Logic (DL)-based consistency checking approaches. They briefly described the background to the DL formalism and reviewed three articles, which are also studied in our research. Their main finding is that only class diagram, sequence diagram and state machine diagram inconsistencies were covered in the papers surveyed and that a few common types of inconsistencies were discussed.

Our work differs from the last four reviews [5], [14], [3], [11] in three ways: a different goal, a more extensive and systematic review, and the presentation of UML consistency rules. Our goal is to identify which consistency rules for all the UML diagrams have been proposed in literature and to create a consolidated set of UML consistency rules. This contrasts with previous reviews that had very different goals, such as focusing on a small subset of UML diagrams, inconcistency management and consistency checking techniques.

Another difference is that all but one piece [5] of work performed an informal literature review or comparison with no defined research question, no precise, repeatable search process and no defined data extraction or data analysis process. Our work, however, follows a systematic procedure inspired by a strict, well-known protocol [12].

In summary, we were unable to find answers to the question posed in our main research objective (Section 1), which confirmed the need for the systematic construction of a consolidated set of UML consistency rules

3 Definition of the Systematic Procedure

As discussed earlier, we started this work using the 95 primary studies obtained in our systematic mapping study [8] as a basis (see Appendix B) for the detailed planning of the SMS [8]). Of an initial set of 1134 research papers, 95 primary studies were selected by following a precise selection protocol driven by seven research questions (see Appendix B). The primary studies were then classified according to several criteria that were also derived from the aforementioned research questions, after which an initial set of 603 rules was coalesced (see Section 4). Since our previous work was not meant to provide a list of UML consistency rules, we did not present the rules collected; our previous work presented only anecdotal evidence that justified this new contribution (please recall the introduction).

In this section, we present the procedure we followed, inspired by a well accepted systematic search and analysis process [12], to obtain our consolidated set of UML consistency rules, starting from the initial set of 603 rules collected from the 95 primary studies. From here on they will be refered to simply as rules. We present the main components of the planning of our work, which are: the specification of research questions that the study aim to answer (section 3.1); the procedure followed to select (or reject) rules to be part of our final consolidated set of rules (section 3.2); and the procedure used to extract data from the rules selected in order to answer the research questions (section 3.3).

3.1 Research questions

The underlying motivation for the research questions was to analyze the state-of-the-art regarding rules. In order to do this, we considered five research questions (RQs): TABLE 1.

Three of those five research questions (specifically, RQs 1, 2, and 3) had already been used in our previous work [8] (see Section 1 of Appendix B). The main differences between the three RQs in this paper and those in our previous work [8] lies in how and for what they are used:

TABLE 1
Research Questions

Research questions		Main motivation
RQ1)	What are the existing UML consistency rules?	To find the UML consistency rules used in literature in order to assess the state of the field.
RQ2)	Which types of consistency problems are tackled in the existing rules?	To find the types of consistency problems tackled in the UML consistency rules: 1) horizontal, vertical and evolution consistency; 2) syntactic and semantic consistency; 3) observation and invocation consistency; and assess the state of the field as regards that dimension of UML diagram consistency.
RQ3)	What types of UML diagrams are involved in UML consistency rules?	To discover the UML diagrams that research and practice has focused on, to reveal the UML diagrams that are considered more important than others, and to identify opportunities for further research.
RQ4)	For what software engineering activities are UML consistency rules used?	To find the different software engineering activities that require UML diagrams to be consistent according to specific sets of consistency rules, in order to understand what they should focus on.
RQ5)	Which UML diagram elements are involved in UML consistency rules?	To find the UML diagram elements that are most frequently tackled in the UML consistency rules.

RQ1: In our previous work we did not present any rule (we merely provided a link to a webpage with a list of rules) and solely discussed metadata about the rules, while in this work we present the consolidated set of rules and provide a detailed analysis of the rules.

RQ2: Unlike our previous work, we now filter out the rules we collected that are already in the UML standard.

RQ3: In our previous work, the focus of the study was a series of papers and we therefore studied how UML diagrams are involved in papers that present consistency rules, whereas in this work the focus of the study is rules and we therefore study how UML diagrams are involved in rules.

The other two research questions (RQs 4 and 5) are new to this paper.

3.2 Inclusion and exclusion criteria

In this section, we discuss the inclusion and exclusion criteria used to refine the set of rules. Since we used the 603 rules found in our SMS [8] as a starting point, and those rules had already been obtained through the use of inclusion and exclusion criteria, we only applied additional exclusion criteria:

- Inaccurate rules:
    - Rules that we considered wrong. For instance, one rule specified that each message in a sequence diagram should trigger a transition in a state machine diagram. We believe that this is incorrect since some of the classifiers receiving a message in a sequence diagram may not have a state-based behavior, and even if they have, not all messages need to trigger state changes;
    - Rules not focusing on UML diagram consistency. For instance, rules which discussed consistency between UML diagrams and other non-UML data, such as requirements or source code, were discarded;
    - Rules that were considered obsolete because they focus on UML characteristics that are no longer supported in the latest UML standard;
- Redundant rules, i.e., all those rules that can be implied by other rules;
- Rules that were already included in previous UML standard versions.

3.3 Data extraction strategy

We answered the research questions (TABLE 1) by extracting data from the collected rules, and recording that data on an Excel spreadsheet (our data extraction form). For each rule we collect the following data:

- Reason for excluding the rule, as per the exclusion criteria (see section 3.2)
- Precise definition of the rules (see Section 5 and Appendix A). When the rule, as originally specified by others, was not sufficiently clear or precise, we redefined it, whilst maintaining the original intent as we understood it, thus hopefully facilitating comparisons and the consolidation of rules. We also recorded the papers that introduced each rule, which can, for instance, be used to identify the rules that are the most important to users;
- Which of the 14 UML diagrams are involved in the rule: 1) Class Diagram (CD); 2) Communication Diagram (COMD) or Collaboration Diagram (COD); 3) Use Case Diagram (UCD); 4) State Machine Diagram (SMD) or Protocol State Machine Diagram (PSMD); 5) Sequence Diagram (SD); 6) Component Diagram (CTD); 7) Object Diagram (OD); 8) Profile Diagram(PD); 9) Activity Diagram (AD); 10) Composite Structure Diagram (CSD); 11) Interaction Overview Diagram (IOD); 12) Package Diagram (PD); 13) Timing Diagram (TD); 14) Deployment Diagram (DD). UML terminology has changed over the years, and the following pair of UML diagrams were therefore considered and counted (in the statistical reports presented in this research) as only one diagram:
    - COD and COMD (diagram 2) were counted as one diagram since COMD is the UML 2.x equivalent of the COD in UML 1.x [7].
    - PSMD is a kind of SMD (diagram number 4 in the list above) [15]. Nevertheless, we initially decided to collect the information for these two diagrams separately since they have different purposes during software development, but both diagrams were eventually reported as only one diagram.
- UML consistency dimensions and types (see Appendix B).
- The software engineering activity that required UML diagrams to be consistent with one another in some way as specified in rules. The following eight definitions describe activities in software engineering in which UML diagram consistency was required, as explained by some authors. Since this is a list of activities from the primary studies we found, this list is not meant to exhaustively represent all the software engineering activities that require diagrams to be consistent:
    - The verification of consistency (*Verification*) is the process of detecting inconsistencies between (or in a singular) UML diagram(s) during software development. (This definition is in line with the IEEE definition of verification [16].)
    - Consistency management (*Management*) includes the detection of inconsistencies in a software model, the diagnosis of the significance of inconsistencies, and the handling of detected inconsistencies [11].
    - Model Refinement and Transformation (*Ref. &Tra.*) are defined as follows [17]:
        - Refinement, which is a semantics-preserving transformation applied to a model, and which produces the same kind of model, e.g., refining a Plateform Independent Model (PIM) into a new PIM;
        - Transformation (also called mapping), which generates a new kind of model, e.g., transforming a Platform Independent Model into a Plat-

form Specific Model (PSM), or a PSM into executable code. These transformations generally add detail to the model.

- Safety and Security consistency (*Saf. & Sec.*): From the point of view of automated analysis, safety consistency implies that there are no conflicting requirements and unintentional (internal) non-determinism in a UML diagram [18]. Security consistency is, meanwhile, defined as the consistency of bounded values in non-abstract elements of a UML diagram [19].
- Impact Analysis (*Impact analysis*) is defined as the process of identifying the potential consequences (side-effects) of a change to the model, and estimating what needs to be modified to accomplish a change [20].
- Model formalization (*Formalization*) describes the formal process used to specify/develop a UML model [21, 22].
- Model understanding (*Understanding*) is the process employed to understand that a UML model is much more than a set of annotated boxes and lines as it reflects the meaning of a software, that is, the UML has semantics. The semantics of the UML is expressed through its metamodel and in particular through both the so-called well-formedness rules and context/domain dependent consistency rules, which describe in plain language and often using the Object Constraint Language (OCL) constraints that UML model elements have to satisfy [23].

4 Execution

In this section we report the results obtained, in terms of number of collected rules, after conducting the protocol discussed in Section 3. Please recall that we started from the primary studies collected in our systematic mapping study (SMS) [8], which began in September 2012 and was completed in October 2013: the primary studies selected span the period from 2000 to 2012; the list of primary studies can be found elsewhere [24]. The execution of the rule selection protocol of our SMS began in May 2014 and was completed in October 2014.

We then started to study the 95 primary study papers obtained from our SMS [8], which allowed us to identify 603 rules. In order to document the execution activity of the protocol described in Section 3 in sufficient detail, we describe the following multi-tasks process:

- First task (T1): 621 unique rules were eventually specified; we identified that, in order to obtain unique, we had to split 18 of the 603 rules into two rules each.
- Second task (T2): we obtained a reduced set of rules by deleting all the inaccurate ones, according to the exclusion criteria. We identified 197 innacurate rules and obtained a set of 424 accurate rules.
- Third task (T3): we further reduced the set of rules by deleting all the redundant ones, again according to the exclusion criteria. We identified 242 redundant rules and obtained a set of 182 accurate and non-redundant rules.
- Fourth task (T4): we further reduced the set of rules by deleting those already presented in previous UML standards, again according to the exclusion criteria. We identified 66 rules that were already in the standard (i.e., well-formedness rules) and obtained a set of 116 rules that are accurate, non-redundant and not already in the standard.

The set of 424 accurate rules (T2), and the final set of 116 consistency rules (T4) were then classified according to the data exctraction strategy discussed in section 3.2.

5 Results

The data recorded on our data extraction form (an excel file) permits the five research questions shown in section 3.1 to be answered. A quantitative summary of the results for research questions RQ2 to RQ5 is presented in TABLE 2. More details are provided in the following sub-sections, in which we shall refer to TABLE 2 for raw data.

Two different set of rules were used to answer our 5 research questions:

1) the final set of 116 consistency rules (T4 in Section 4) to answer RQ1, RQ2 and RQ3;
2) the set of 424 accurate rules (T2 in Section 4) for RQ4 and RQ5. We chose to include the 242 redundant rules (T3 in Section 4) and the 66 rules already presented in the UML standard (T4 in Section 4) to answer these two research questions, because we wished to show the whole picture of what researchers have focused on.

TABLE 2 (first raw) simply refers to Appendix A for research question RQ1 since the purpose of the question is to report on the consistency rules we have coalesced. The section of TABLE 2 that contains data for research question RQ3 shows all the combinations of UML diagrams for which we have found consistency rules in the primary studies, and the number of consistency rules for each combination after conducting the protocol discussed in section 3 (e.g., after applying inclusion/exclusion criteria): e.g., we have found 14 rules involving only the state machine diagram (SMD), 5 rules involving the sequence and use case diagrams at the same time (SD and UCD), and 1 rule involving the class, state machine and activity diagrams at the same time (CD, SMD and AD). Primary studies showed rules for 36 different diagram combinations, i.e., a small subset of all the possible combinations of UML diagrams. Note also that the table shows combinations with no rule (e.g., UCD and SMD): this means that we found rules in primary studies involving those diagrams but the rules were eventually discarded because of our include/exclusion criteria. The colored column shows the unique ID# we gave to each of the 36 UML diagram combinations; these unique IDs will later be used in figures to simplify the data analysis; the color code for combinations will later be used in section 5.3. The colored column is white when the number of rules for the corresponding combination of diagrams is zero. The 36 UML diagram combinations refer to: 1) rules involving only one diagram

TABLE 2
SUMMARY OF THE ANSWERS TO THE RQS

Research Question 1: What are the existing UML consistency rules? (see Appendix A)							
Research Question 2:Which types of consistency problems are tackled in the existing rules?							
Horizontal	97	Invocation	8	Vertical	7	Evolution	3
Observation	1	Syntactic	95	Semantic	21	-	-
Research Question 3: What types of UML diagrams are involved in UML consistency rules?							
1 SMD	14	**10** UCD and SMD	0	**19** COMD	1	**28** CD	33
2 CD, SMD and AD	1	**11** CD and SMD	7	**20** UCD and CD	3	**29** ID and CD	0
3 SD and UCD	5	**12** PSMD and CD	0	**21** OD and AD	0	**30** ID and UCD	2
4 AD	0	**13** SD and AD	7	**22** COMD and AD	1	**31** PSMD and SMD	0
5 SD	6	**14** UCD and OD	0	**23** COMD and CD	3	**32** CSD	8
6 OD and COMD	0	**15** SMD and COMD	1	**24** COMD and UCD	0	**33** SD and CD	9
7 OD and CD	1	**16** COMD and CD	1	**25** UCD and AD	4	**34** SD and COMD	0
8 AD and CD	4	**17** SMD and SD	1	**26** UCD	4	**35** CD, SMD and COMD	0
9 SMD and AD	0	**18** PSMD	0	**27** SMD and OD	0	**36** UCD, SD and CD	0
Research Question 4: For what software engineering activities are UML consistency rules used?							
Verification	220	Ref/ &Tra.	54	Understanding	21	Saf. & Sec.	7
Impact Analysis	57	Management	51	Formalization	14	-	-
Research Question 5: Which UML diagram elements are involved in UML consistency rules?							
class/es	161	action/s	18	invariant/s	9	capsule/es	4
state/s	86	Collaboration	17	interface/s	9	class name	3
operation/s	56	chart/s	17	Interaction	9	super class	2
association/s	45	machine/s	17	Sender	8	sub sequence	2
use case	41	pre-condition	16	Flow	8	sub classes	2
activity/ies	39	Strings	15	composite	8	scenario	2
message/s	38	Link	13	property	7	query	2
sequence/s	38	event/s	12	connector/s	7	post-state	2
object/s	36	guard/s	12	Call	7	package	2
instance/s	30	Visibility	11	node	6	swimlanes	1
name/s	26	Aggregation	11	generalization/s	6	stimulus	1
transition/s	26	post-condition/s	10	constraint/s	6	stimuli	1
type/s	23	method/s	10	Port	5	name pace	1
attribute/s	19	Receiver	10	Parameter	5	edge	1
Behavior	18	Multiplicity	9	Lifeline	4	actor	1

such as AD, SD etc.; 2) rules involving pairs of diagrams such as SD and UCD; 3) rules involving 3-tuples of diagrams such as CD, SMD and AD. However, not all the 14 diagrams appear in TABLE 2, since a diagram (or a pair or a 3-tuples of diagrams) is not shown in TABLE 2 if we were unable to find a rule involving that (those) diagram(s) in primary studies.

5.1 What are the existing UML consistency rules? (RQ1)

The principal observation we can make about RQ1 is that the researchers of UML consistency rules have typically defined a number of similar rules over and over again. Specifically, we collected a list of 621 rules from the 95 primary studies. After removing rules tha were inaccurate, redundant, and already presented in UML standards, we obtained a final set of 116 rules that are presented in TABLE 3 in Appendix A. In other words, only 18.68% (116 out of 621) of the rules initially collected are included in our final set of rules because of our include/exclusion criteria (section 3.2). The remaining rules were eliminated because they were inaccurate: 31.72%, 197 out of 621; resuling in 424 accurate rules. Specifically, 87 (14.00%) were not consistency rules (e.g., rules describing good modeling practices); 28 (4.50%) were explained in a too ambiguous language; 14 (2.25%) were obsolete because they referred to UML elements that are no longer used in the latest UML standard; and 68 (10.95%) were simply wrong (i.e., contradicting the UML specification). Other rules were mostly eliminated because of redundancies (57.08%, 242 out of 424). Finally, 66 out of 182 (36.26%) non-redundant rules were excluded because they are already part of the UML standard.

The rules with the largest number of references in primary studies are rule 112 (with 48 references), and rules 110, 48, and 115 (respectively with 33, 25, and 15 reference each). In a nutshell, rule 121 focus focus on the required visibility in order to exchange messages in a sequence diagram; rule 110 concerns the consistency of messages in a sequence diagram and the class diagram; rule 48 specifies the consistency of behavior specification in lifelines of a sequence diagram and a state machine of the corresponding classifiers; and rule 115 describe the consistency of class names in class diagrams and sequence diagrams. The complete list of references for the 116 rules presented in TABLE 3 in Appendix A, and elsewhere [24].

5.2 Which types of consistency problems are tackled in the existing rules? (RQ2)

The results obtained for RQ2 show (TABLE 2) that the vast majority of rules are Horizontal (83.62%, 97 out of 116 rules) and Syntactic (81.90%, 95 out of 1116 rules). Moreover, we found 21 (18.10%) Semantic rules. Researchers strikingly described many more syntactic than semantic consistency rules. We conjecture that the main reason for this is that syntactic rules are easier to specify than semantic rules and that the UML standard more formally specifies syntax than semantics, which hinders the specification of semantic rules. It may also be the case that semantic rules are specific to the way in which the UML notation is actually used, which is organization, project, or team specific, and are therefore seldom described in published manuscripts.

Researchers have also paid much less attention to: 1) Invocatoin rules (8 rules, 6.90%); 2) Vertical rules (7 rules, 6.03%); Evolution rules (3 rules, 2.59%); and Onservation rules (only one rule, 0.86%). We believe that the mappings between vertical levels, the evolution of a UML model, the invocation and observation consistency, are concepts much less easy to understand than the mere specification of a UML model or the horizontal consistency levels, and this explains why, even though we found papers discussing these consistency dimensions, these papers did not present many consistency rules.

5.3 What types of UML diagrams are involved in UML consistency rules? (RQ3)

In Fig. 1, we use a bubble plot to represent multiple dimensions of the results in one figure: the bubble plot essentially embodies a two x–y scatter plot with bubbles at the category intersections. This synthesis method is useful since it provides both a map and a rapid overview of a research field [25]. Combined with the bubble plot are some bubbles (left-hand side) which are pie charts to help us describe the many dimensions involved in this research.

Fig. 1 shows the mapping of the selection process onto the 36 combinations of UML diagrams presented in TABLE 2: the combinations are numbered vertically in Fig. 1. On the left-hand side of Fig. 1, each bubble pie chart describes the number of rules included in the final set (in red) and the

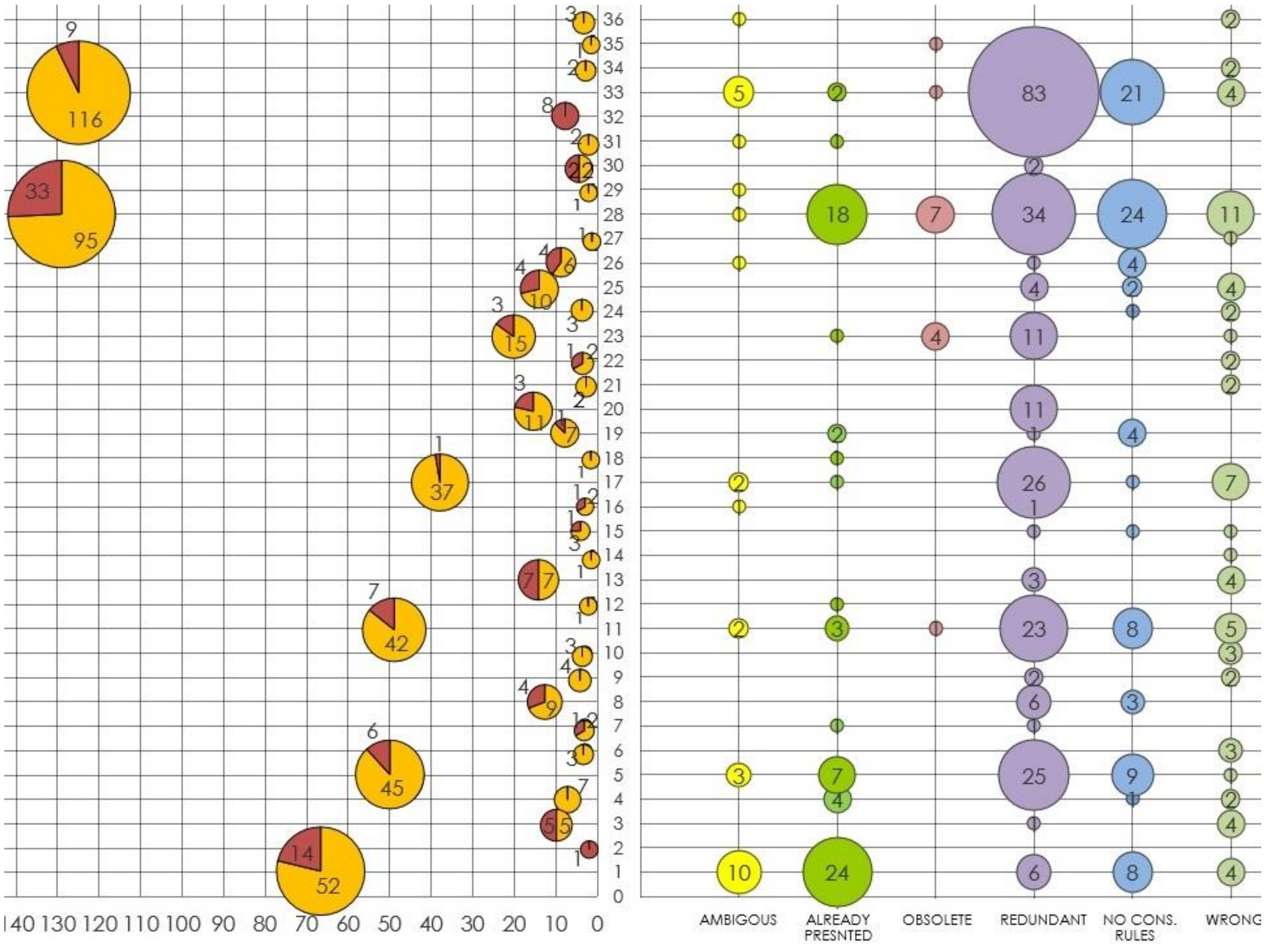

Fig. 1. Summary of UML Consistency rules selection

number of rules deleted (in yellow) for each of the 36 UML diagram combinations. For instance, for combination 1 (i.e., SMD, as per TABLE 2, i.e., State Machine Diagram as per section 3.3) the initial set of rules involving only the State Machine Diagram contained 66 rules (14+52), of which we removed 52 and kept only 14. On the right-hand side, we present our reasons for deleting rules and the magnitude of each reason for each combination of diagrams. For instance, of the 52 deleted rules involving only the State Machine Diagram, 10 were deleted because they were ambiguous (beyond repair), 24 were already in the standard, six were redundant (with rules we kept), eight were not consistency rules, and four were wrong.

The results show that the diagram combination that involves the largest number of rules in the initial set, specifically 128 rules (33+95), is combination 28 (y-axis), i.e., the Class Diagram: the majority of the rules involve only the Class Diagram. This is followed by combination 33, i.e., rules involving the Class Diagram and the Sequence Diagram, with 125 rules (9+116). These two sets of rules make up 40.74% (253 out of 621 rules) of the total number of initial rules, and 36.21% of the final set of rules (42 out of 116). It is also important to mention that there is a huge gap between these two most involved diagram combinations and the next most involved diagram combinations, which are the State Machine Diagram (66 rules, y-axis value 1), Sequence Diagram (51 rules, y-axis value 5), Class Diagram with State Machine Diagram (49 rules, y-axis value 11), and State Machine Diagram with Sequence Diagram (38 rules, y-axis value 17). The absence of rules for some diagrams or some diagram combinations can be explained by the fact that the semantics and syntax of some diagrams overlap (e.g., package, object and class diagrams). However, it is possible to argue that the fact that the semantics and syntax of those diagrams overlap, and that there is a need for these different diagrams, should justify the definition of rules to ensure that the diagrams are consistent, unless all the rules required are already in the UML standard (which is unlikely). Another possible reason for a lack of rules for some diagrams (or diagram combinations) is that users of the UML notation follow a specific UML (behavior) modeling practice that is assumed to be standard and does not therefore require the definition of consistency rules. Nevertheless, since the UML allows different behavior modeling practices, rules underpinning such practices should be specified so as to be clear for all stakeholders.

Another interesting finding is the high percentage (88.33%) of rules eliminated for the Sequence Diagram. If we consider all the UML diagram combinations in which the Sequence Diagram is involved (rows 3, 5, 13, 17, 33 and 34 in Fig. 1), it is actually possible see that, when starting with the total of 240 rules collected, only 28 (11.67%) made it to the final set of 116 rules. The main reason why rules were discarded is redundancy, i.e., rules presented several times by authors: for instance, 25 of the 45 discarded rules involving only the Sequence Diagram (row 5 in Fig. 1) were redundant. One possible explanation for this is that the sequence diagram is one of the most ill-specified diagrams in the UML specification and researchers are attempting to make sense of it by specifying rules which, when coalesced, happen to be redundant with one another (138 of the 212 eliminated rules, 65.09%, were redundant with other rules). We discussed the 36 different combinations of UML diagrams (detailed in TABLE 2) covered by the final set of 116 rules. Our results show that rules collected describe consistency in only 10 of the 14 UML diagrams.

Fig. 2 shows only seven diagrams because we grouped rules for the Sequence, Communication, Interaction Overview and Timing diagrams together under the Interaction Diagram (ID) category since these diagrams are different variants of the ID [15]. We did not find any rules involving the Package, Component, Profile, Timing, Interaction

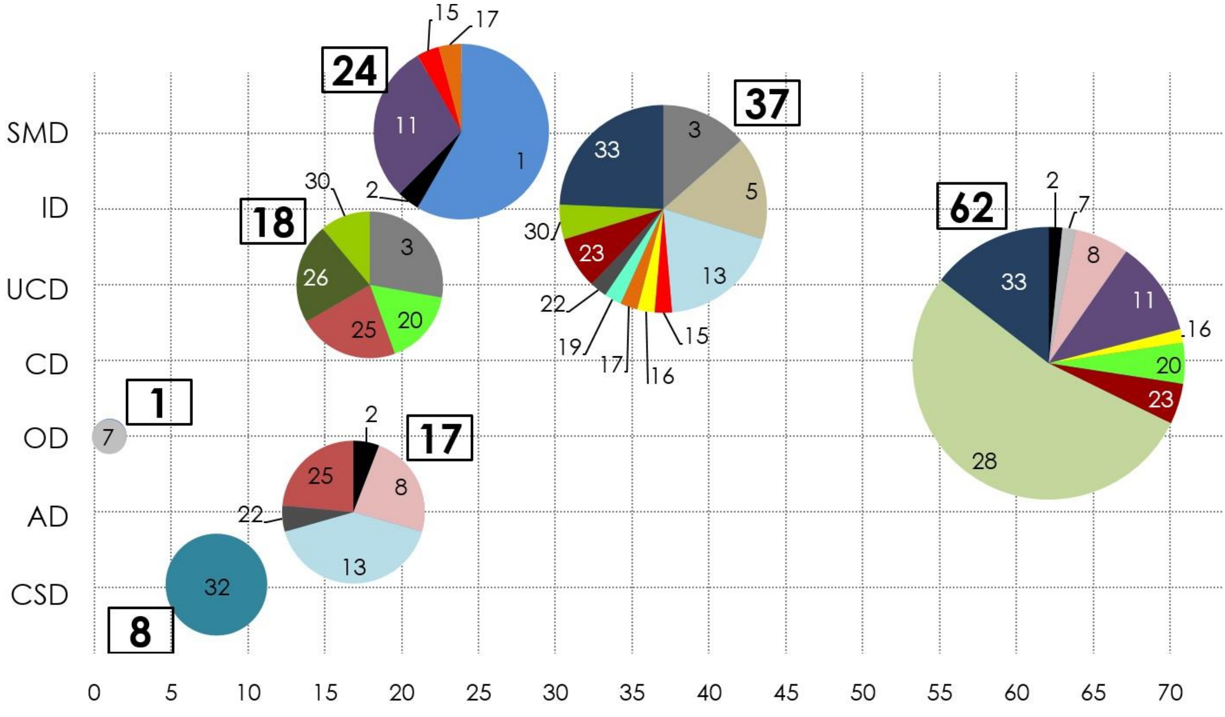

Fig. 2. UML Consistency rules grouped into UML diagrams.

Overview and Deployment diagrams. We conjecture that there are several reasons for the lack of rules involving these diagrams. First, as already mentioned, the fact that the Package Diagram and the Deployment Diagrams overlap with the Class Diagram might be a reason for the lack of rules involving these diagrams; However, as agrued previously, this similarity between the two diagrams could be the very reason needed to justify more rules involing these two diagrams or the Package Diagram with other diagrams. The lack of widespread tool support and underspecification/ambiguity in the specification for the Timing and Interaction Overview diagrams is a possible reason for the lack of rules involving these diagrams.

Fig. 2 uses the same color code and ID# numbers for diagram combinations as TABLE 2. The figure shows, for each of the seven diagrams involved in at least one rule (labels on the y-axis), the total number of rules found that involve that diagram (large font size, bold faced, square-bordered number) and the proportions of those rules that are shared with other diagrams as a pie chart. These data were taken directly from TABLE 2, although they are presented in a different form. For instance, the Activity Diagram (AD) was found to be involved in 17 rules. TABLE 2 indicates that one rule involves AD with CD and SMD (combination number 2), four rules involve AD and CD (combination 8), seven rules involve AD and SD (combination 13), one rule involves AD and COD (combination 22) and four rules involve AD and UCD (combination 25); These appear clock-wise from the top of the pie-chart for row AD in Fig. 2. Note that upon summing up the rule count of each pie chart in Fig. 2 we obtain 167 rather than 116 because when a rule involves more than one diagram it contributes to more than one pie chart.

Not surprisingly, the UML diagrams that are most involved in the final set of 116 rules are the Class Diagram (62 rules, 53.45%), the Interaction Diagram (37 rules, 31.90%), and the State Machine Diagram (24 rules, 20.69%). Research on UML consistency rules has paid much less attention to the Use Case Diagram (22 rules, 15.52%) and the Activity Diagram (17 rules, 14.66%).

The diagrams that are least covered are the Composite Structure Diagram and the Object Diagram, which only had 6.90% (eight rules) and 0.86% (one rule) of the total of 116 rules, respectively.

5.4 For what software engineering activities are UML consistency rules used? (RQ4)

In this section, we discuss the results of the different uses of rules. As we already explained in Section 5, we answered this RQ by considering the set of 424 rules rather than only the final set of 116 rules. We chose to do this because, since redundant rules may have been defined in the context of different Software Engineering activities, we wished to show what researchers have focused on.

Results (pie chart on the left-hand side of Fig. 3) show that 51.89% (220 out of 424) of the rules are proposed for model consistency verification (Verification in the figure), 13.44% (57 out of 424) for impact analysis (Impact Analysis in the figure), 12.74% (54 out of 424) for model refinement and transformation (ref. &tra.), 12.03% (51 out of 424) for consistency management (Management), 4.95% (21 out of 424) for model understanding (understanding), 3.30% (14 out of 424) for model formalization (formalization), and 1.65% (7 out of 424) for safety and security consistency (saf. & sec.). All the percentages shown in this section are rounded off in the pie chart (left-hand side) of the Fig. 3.

In the light of these figures, the reader may be surprised to discover that half of the consistency rules have been defined without any additional context of use such as specific model-driven activities. This may suggest that not many UML-based software engineering activities require UML diagrams to be consistent, which would be surprising, or at least that papers describing UML-based software engineering activities do not discuss (or need to rely on?) consistency rules. On the other hand, this may rather mean that typical UML-based software engineering activities require the same set of consistency rules, which can therefore be described independently of their context of use.

5.5 Combining RQ3) and RQ4)

Upon combining the data presented for RQ4) in the previous section (For what software engineering activities are UML consistency rules used?) with the data presented in the analysis of RQ3) in section 5.2 (What types of UML diagrams are involved in UML consistency rules?), we show in Fig. 3 (right-hand side) which UML diagram was covered by rules in which Software Engineering activity.

The bubble plot in Fig. 3 (right-hand side) shows that the Class Diagram (CD) is mostly involved in rules used for verification (116 rules) and impact analysis (53 rules)

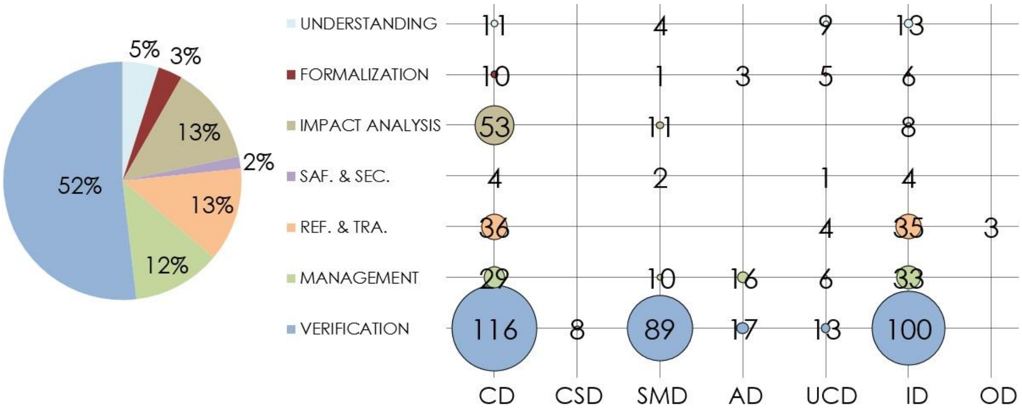

Fig. 3. Summary of rules between UML diagrams and uses in Software Engineering activities.

activities. Furthermore, the Interaction Diagram (ID) and State Machine Diagram (SMD) have 100 and 89 rules, respectively, which are used for verification. This is not entirely surprising since these three UML diagrams are likely the behavioral UML diagrams that are most frequently used in the activities cited above.

Fig. 3 (right-hand side) shows a total of 660 rules rather than 424 accurate rules because we grouped all the rules related to each of the seven UML diagram separately, resulting in some rules being repeated.

In Fig. 4 we present a bubble plot distribution that was obtained by combining the results of RQ3 and RQ4 with the years of the rules publications [24]. In Fig. 4, we coupled the years of rule's publication, on the right-hand with the UML diagram covered, and on the left-hand with the Software Engineering activity that used the rule.

The right-hand side of Fig. 4 shows the number of rules presented in the period between 2000 and 2012, and classified by the seven UML diagrams covered by the rules. As in the previous two sections, we grouped all the rules related to each of the seven UML diagrams, resulting in 660 rules instead of 424 rules, since some rules were repeated. For instance, the 17 rules presented for CD and SMD (combination 11 in TABLE 2) were included in both the CD count and in the SMD.

However, the left side of Fig. 4 shows only 424 rules in the period between 2000 and 2012 (without overlapping) classified in the eight Software Engineering activities presented in section 3.3. On the the left-hand side of Fig. 4, no rules overlap because each of the 95 primary study papers [8] was assigned only one of the eight Software Engineering activities presented in section 3.3. The rules presented in each paper were consequently classified according to only those seven activities.

The results show (left-hand side of Fig. 4) that a great number of rules were proposed in 4 of the 13 years considered in this study: 49.76% (211 out of 424 rules) of the rules were published between 2002 and 2005. In these 4 years, 152 out of 259 (58.69%) of the total number of rules related to the Class Diagram and 103 out of 199 (51.76%) of the total number of rules related to the Interaction Diagram were published (right-hand side of Fig. 4). Moreover, in these 4 years we can see that the three peaks of Software Engineering activities use (left-hand side of Fig. 4): 1) 57 rules presented in 2003 for impact analisys, 32 rules in 2005 for verification, and finally 25 rules in 2004 for refinement and transformation activities. We can conclude that these four years represent the most prolific period in literature in terms of publishing UML consistency rules to date.

Another important aspect that should be highlighted is

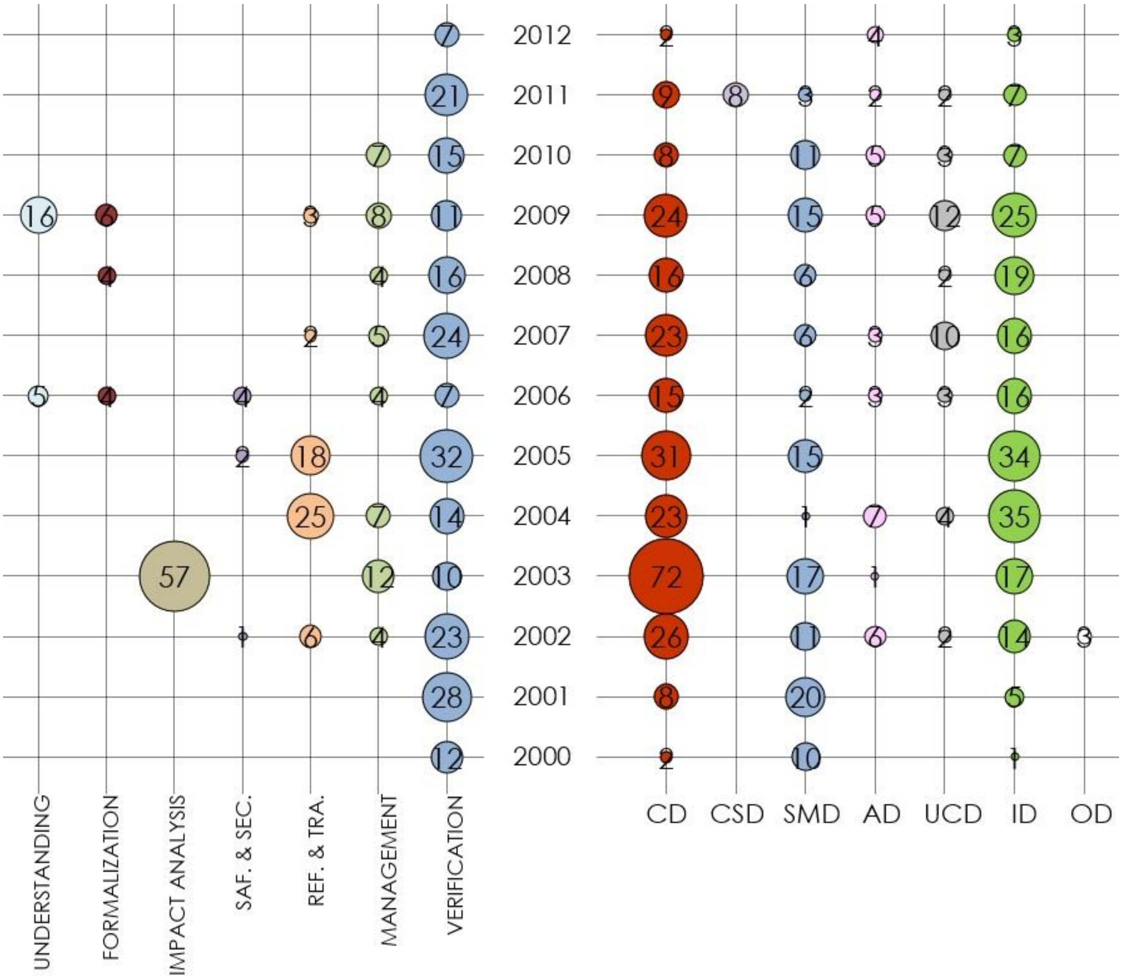

Fig. 4. Summary of UML Consistency rules between UML diagrams.

that the number of UML consistency rules covering the Verification activity (left side of Fig. 4) remained relatively stable in the entire period considered, with a peak of 32 new rules in 2005 and with only seven rules being presented in 2006 and 2012. Furthermore, the right-hand side of Fig. 4 shows that class and interaction diagrams remained stable in the period between 2002 and 2009 with a peak of 72 rules in 2003 and 35 rules in 2004, respectively. The fact that researchers consistently continued to focus on class and interaction diagrams during these years, shows the vital importance of these diagrams. The number of rules decreased in 2012, but this is likely because many papers published in that year were not yet available online when we performed searches [8].

5.6 Which UML diagram elements are involved in UML consistency rules? (RQ5)

Fig. 5 shows the 200 most frequently used words in the plain English specification of the 116 rules found (4761 words in total), in which we can recognize the 60 UML element names already presented in TABLE 2. This is a weighted word cloud of the 116 rules found in the 95 papers included as primary studies in our SMS [8].

Fig. 5 was created using Tagxedo [26] , which uses a streaming algorithm to filter the text input. The aim of Fig. 5 is to provide a first impression of the main UML diagram elements referenced in the final set of 116 UML consistency rules. Not surprisingly, the UML elements most involved in the rules are related to the class, sequence, state machine, use case and finally activity diagrams. These results concerning the model elements most frequently involved in UML consistency issues may be useful as regards helping focus the teaching of UML by concentrating on specific elements. Indeed, if an element is not rigorously specified, then there is a risk of many semantics issues appearing in the model.

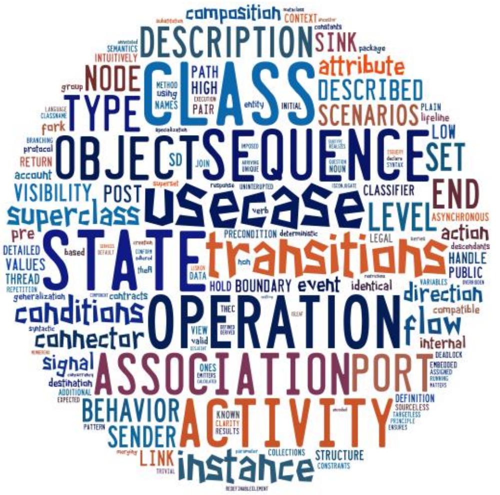

Fig. 5. Summary of the UML diagram elements involved in UML Con-

5.7 Additional results

In this section we shortly discuss the 66 rules that were excluded from the final set of rules because they had already been presented in previous UML standards. It is important to note that 66 of the 182 (36.26%) non-redundant and accurate UML consistency rules presented by different authors had already been proposed in previous UML standards. We compared the rules with the applicable UML specification version at the time of publication (in which the rule was found): we either found a reference to the applicable version in the publication or inferred it from the time of publication; finally, we also checked whether the rule was contained in the most recent version of the specification (UML 2.4.1 and 2.5). The 66 rules were excluded because they had already been presented in one of the previous UML standards (UML 1.3, 1.5, 2.0, 2.4.1, and 2.5).

6 Threats to Validity

The main issues that may constitute a threat to the validity of our research are related to publication bias, selection bias, inaccuracy in data extraction, and misclassification [27]. Selection bias refers to the distortion of a statistical analysis owing to the criteria used to select the rules. We attempted to mitigate this risk by carefully defining: 1) our inclusion and exclusion criteria in order to select primary studies [8], and 2) our exclusion criteria in order to select rules in this work, based on our predefined research questions. We used seven search engines to search journals, conference and workshop proceedings that are relevant to UML consistency rules [8]. We did not consider grey literature (e.g., PhD theses, books) because it might have affected the validity of our results. We then organized the selection of rules by following a rigorous multi-phase process, during which our filtering criteria were applied, to arrive at a consolidated set of UML consistency rules.

The limited number of rules in this research could be an inherent statistical threat to validity, despite the fact that we attempted to mitigate this by following rigorous guidelines that are well-known in the empirical software engineering community [12]. However, the classification scheme and process that we provide in this paper may be used as a starting point for future research.

The terminology used for UML consistency rules in this study has been identified from a number of other studies found in [8]. We observed that the terminology used by researchers is not always consensual and the authors in this field sometimes discuss UML consistency issues using different terms such as “Horizontal consistency” and" Intra-model consistency" to refer to the same concept. Thus, by following a systematic research method [12], we developed a set of terms with the intention of providing a useful and accessible set of definitions for the terms that are used within the UML consistency domain. Nevertheless, we consider that this set of terms and definitions is a threat to validity that needs to be pointed out.

Another threat to consider is that, since in this study we relied solely on rules that have been published in literature [8], the fact that we did not find rules involving some diagram(s) (described in section 5.3) does not mean that none have been created.There is a lot of tool support for UML, and since these tools do support some kinds of model-driven development activities, they must rely on, (and likely enforce) UML diagram consistency. However, the rules they rely on are not necessarily in the public domain or at least discussed in published literature. In addition, some rules might also be for specific domains (industry, organization, project, team specific, etc).

Finally, as it is impossible to thoroughly search for every rule published on UML consistency, we acknowledge that some rules might not have been included, such as rules concerning diagram (model) synthesis, rules in standard UML-driven software development processes discussed in textbooks, and rules for domain specific languages.

7 CONCLUSION

In recent years, a great number of UML consistency rules have been proposed by researchers in order to detect inconsistencies between UML diagrams. However, no previous study has, to the best of our knowledge, summarized and analyzed these UML consistency rules by following a systematic process. This work presents the results obtained after following a systematic protocol, whose aim was to identify, present and analyze a consolidated set of UML Consistency rules from literature. No such mapping study or review existed prior to our work.

We obtained the set of UML Consistency rules by following a systematic procedure inspired by well-known guidelines for performing systematic literature reviews in software engineering presented by Kitchenham and Charters [12]. A significant amount of space in this paper has been used to discuss the 116 rules and we have also discussed how the systematic process was defined, since we felt that it is of the upmost importance to allow readers to precisely understand the results and the conclusions we came to, and to allow other researchers to replicate the study or compare their results with ours.

From an initial set of 621 UML consistency rules, published in literature and extracted from seven scientific research databases, a total of 116 rules were eventually selected and analyzed in depth, by following a precise selection protocol driven by four research questions. We observe that (in no particular order of importance):

- The UML diagram that is most involved in the final set of the 116 rules is the Class Diagram (62 rules, 53.45%), followed by the Interaction Diagram (37 rules, 31.90%) and the State Machine Diagram (24 rules, 15.52%). This is not entirely surprising since these are likely the most frequently used UML diagrams [28]. However, recent research [29] suggests that the Activity Diagram is the second most frequently used UML diagram after the Class Diagram. Considering the very scarce number of rules we found for the Activity Diagram (17 of 116 rules, 14.66%), we believe that future research should focus more on this diagram. However, if we compare the last UML 2.5 spec. (released in 2015) [6]with the most frequently used UML 2.0 spec. (released in 2005) [30], we can see that the UML has put more effort into the definition of activity diagrams, which consequently leads to (explicitly or otherwise) consistency rules concerning that diagram. Another diagram to which research on UML consistency rules has paid much less attention is the Use Case Diagram (22 rules, 15.52%). We believe that one of the reasons for these results is that the semantics of the use case diagram is simple and the semantics generally presents the include, extend and generalization, even though the semantics for this diagram is not, however, very clear to every one: Cockburn argues [31] for instance that generalization between use cases is not clear and necessary, while other [32-34] argue that there are consistency rules between use case, and more so use case descriptions and other diagrams such as class and sequence diagrams. In addition, the mapping between semantics in the use case diagram and other diagrams is not clear either: for instance, if we have a generalization between use cases and use cases to map in two sequence diagrams, how does the generalization (in the use case) translate into relations between the two sequence diagrams.
- Besides the Class, Interaction, and State Machine diagrams, there is a need for much additional research on consistency rules involving other UML diagrams. For example, no rule was found for the Package, Component, Timing, Profile, Interaction Overview and Deployment Diagrams. One consideration that could partially explain the lack of rules for these diagrams is their overlap with the Class diagram syntax/semantics.
- Additionally, 38.97% (242 of 621) of the collected rules that have been proposed by researchers over the years describe redundant (similar or even identical) rules. This highlights the need for a central repository for such rules, such as this manuscript. Another adequate place in which to record those rules would be the UML specification itself.
- We found that 36.26% (66 of 182) of non-redundant and accurate UML consistency rules presented by different authors had already been presented in one of the previous UML standards.
- The main software development activity that justified the description of UML diagram consistency rules is UML consistency verification, 51.89%, followed by impact analysis, 13.44%, UML model refinement & transformation, 12.74%, and management of consistency, 12.03%;
- We found that 49.76% of the rules were published between the years 2002 and 2005, and this period represents the most prolific period in literature in terms of publishing UML consistency rules;
- The results show that the vast majority of the rules are horizontal and syntactic rules, and very few address vertical, semantic, invocation, observation and evolution consistency. One reason for this could be the fact that the UML syntax is more formally described (by the UML metamodel) in the specification than semantics, and creating syntactic rules may therefore prove more feasible. Another reason for this lack could be explained by the fact that users mostly employ a few types of behavior diagrams that they understand or find suitable. Despite the

fact that the topic of UML consistency is mature, it still needs to evolve to include more definitions of UML consistency rules in all dimensions.

By demonstrating that there are various areas concerning UML consistency rules that require improvements, our research calls for future work in this area. Firstly, we pose the following question: should some of the rules presented in this work, for instance, the most referenced rules numbered 112, 110, 48 and 115 (see Appendix A), be specified in the UML standard?

Finally, we believe that the final consolidated set of UML consistency rules resulting from this research could be used as a reference in the future by researchers in the field of UML model consistency.

Different avenues for future work can be considered.

According to [8], there are still very few evaluation works on consistency rules reported in literature. We therefore initialized the process of validating the rules collected by organizing the 1st International Workshop on UML Consistency Rules (WUCOR) [35], which will be followed by the implemention of an online survey that will be used to obtain the opinions of experts from academia and industry. In addition, the set of rules will be checked against a number of UML models from academia and industry.

As part of our future work, we also believe that additional consistency rules could be collected from other sources, such as: 1) textbooks on UML-based object-oriented software development [36] that implicitly or explicitly suggest consistency rules; 2) techniques in which UML diagrams are synthesized from other diagrams [37], in other words the process of enforcing some consistency rules between diagrams; 3) consistency rules presented in the context of Domain Specific Languages [38].

Acknowledgment

This work has been developed within the SEQUOIA Project, (TIN2015-63502-C3-1-R) (MINECO/FEDER) funded by Fondo Europeo de Desarrollo Regional and Ministerio de Economía y Competitividad, and a NSERC discovery grant.

References

[1] J. Mukerji and J. Miller, "Overview and guide to OMG's architecture," Object Management Group2003.

[2] D. Thomas, "MDA: Revenge of the modelers or UML utopia?," *IEEE Software,* vol. 21, pp. 15–17, 2004.

[3] M. Usman, A. Nadeem, K. Tai-hoon, and C. Eun-suk, "A Survey of Consistency Checking Techniques for UML Models,"in Proceedings of Advanced Software Engineering and Its Applications, Hainan Island, China, 2008.

[4] M. Genero, A. M. Fernández-Saez, H. J. Nelson, G. Poels, and M. Piattini, "A Systematic Literature Review on the Quality of UML Models," *Journal of Database Management,* vol. 22, pp. 46-70, July-September 2011 2011.

[5] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Information and Software Technology,* vol. 51, pp. 1631-1645, 2009.

[6] OMG, "OMG Unified Modeling LanguageTM - Superstructure Version 2.5," 2015.

[7] Z. Manna and R. J. Waldinger, "Toward automatic program synthesis. Commun.," *ACM* vol. 14, pp. 151-165, 1971.

[8] D. Torre, Y. Labiche, and M. Genero, "UML consistency rules: a systematic mapping study,"in Proceedings of 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014), London, UK, 2014.

[9] N. Ibrahim, R. Ibrahim, M. Z. Saringat, D. Mansor, and T. Herawan, "Consistency rules between UML use case and activity diagrams using logical approach," *International Journal of Software Engineering and its Applications,* vol. 5, pp. 119-134, 2011.

[10] J. Simmonds, R. V. Straeten, V. Jonkers, and T. Mens, "Maintaining Consistency between UML Models using Description LogicZ," *RSTI – L'Object LMO'04,* vol. 10, pp. 231-244, 2004.

[11] G. Spanoudakis and A. Zisman, "Inconsistency management in software engineering: Survey and open research issues," in *Handbook of Software Engineering and Knowledge Engineering*, S. K. Chang, Ed., ed Singapore: World Scientific Publishing Co., 2001, pp. 329-380.

[12] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University2007.

[13] D. Kalibatiene, O. Vasilecas, and R. Dubauskaite, "Rule Based Approach for Ensuring Consistency in Different UML Models,"in Proceedings of 6th SIGSAND/PLAIS EuroSymposium 2013, Gdańsk, Poland, 2013.

[14] M. A. Ahmad and A. Nadeem, "Consistency checking of UML models using Description Logics: A critical review,"in Proceedings of 6th International Conference on Emerging Technologies Islamabad, Pakistan, 2010.

[15] OMG, "OMG Unified Modeling LanguageTM - Superstructure Version 2.4.1," 2011.

[16] IEEE, "IEEE Standard for Software VeriÞcation and Validation Plans (IEEE Std 1012-1986)," 1992.

[17] R. F. Paige, D. S. Kolovos, and F. A. C. Polack, "Refinement via Consistency Checking in MDA," *Electronic Notes in Theoretical Computer Science,* vol. 137, pp. 151-161, 2005.

[18] Z. Pap, I. Majzik, A. Pataricza, and A. Szegi, "Methods of checking general safety criteria in UML statechart specifications," *Reliability Engineering & System Safety,* vol. 87, pp. 89-107, 2005.

[19] O. Pilskalns, D. Williams, D. Aracic, and A. Andrews, "Security Consistency in UML Designs,"in Proceedings of 30th Annual International Computer Software and Applications Conference, Chicago, USA, 2006.

[20] L. C. Briand, Y. Labiche, and L. O'Sullivan, "Impact analysis and change management of UML models,"in Proceedings of International Conference on Software Maintenance, Amsterdam, The Netherlands, 2003.

[21] I.-Y. Song, R. Khare, Y. An, and M. Hilsbos, "A Multi-level Methodology for Developing UML Sequence Diagrams,"in Proceedings of 27th International Conference on Conceptual Modeling, Barcelona, Spain, 2008.

[22] J. Yang, "A framework for formalizing UML models with formal language rCOS,"in Proceedings of 4th International Conference on Frontier of Computer Science and Technology, Shanghai, China, 2009.

[23] Y. Labiche, "The UML is more than boxes and lines,"in Proceedings of Workshops and Symposia at MODELS 2008, Toulouse, France, 2008.

[24] D. Torre, Y. Labiche, M. Genero, and M. Elaasar, "A systematic

identification of consistency rules for UML diagrams," Carleton University, Ottawa2015.

[25] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering,"in Proceedings of 12th International Conference on Evaluation and Assessment in Software Engineering (EASE 2008), Bari, Italy, 2008.

[26] H. Leung. (2014). *Tagxedo*. Available: http://www.tagxedo.com/

[27] D. I. K. Sjoberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg*, et al.*, "A Survey of Controlled Experiments in Software Engineering," *IEEE Trans. Softw. Eng.,* vol. 31, pp. 733-753, September 2005 2005.

[28] B. Dobing and J. Parsons, "How UML is used," *ACM,* vol. 49, pp. 109-113, May 2006 2006.

[29] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, "What are the used UML diagrams? A Preliminary Survey,"in Proceedings of In Proceedings of 3rd International Workshop on Experiences and Empirical Studies in Software Modeling - CEUR Workshop Proceedings (EESSMod 2013), Miami, Florida - USA, 2013.

[30] OMG, "OMG Unified Modeling LanguageTM - Superstructure Version 2.0," 2005.

[31] A. Cockburn, *Writing Effective Use Cases*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[32] N. Ibrahim, R. Ibrahim, and M. Z. Saringat, "Definition of Consistency Rules between UML Use Case and Activity Diagram,"in Proceedings of 2nd International Conference, Daejeon, South Korea, 2011.

[33] J. Chanda, A. Kanjilal, S. Sengupta, and S. Bhattacharya, "Traceability of Requirements and Consistency Verification of UML Use case, Activity and Class Diagram: A Formal Approach,"in Proceedings of IEEE ICM2CS New Delhi, 2009.

[34] L. Fryz and L. Kotulski, "Assurance of System Consistency During Independent Creation of UML Diagrams,"in Proceedings of 2nd International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX '07), 2007.

[35] D. Torre, Y. Labiche, M. Genero, M. Elaasar, T. K. Das, B. Hoisl*, et al.*, "1st International Workshop on UML Consistency Rules (WUCOR 2015): Post workshop report," *SIGSOFT Softw. Eng. Notes,* vol. 41, pp. 34-37, 2016.

[36] D. Torre, Y. Labiche, M. Genero, and M. Elaasar, "UML consistency rules in technical books,"in Proceedings of IEEE International Symposium on Software Reliability Engineering, Fast Abstract (ISSRE 2015).

[37] D. Torre, Y. Labiche, M. Genero, and M. Elaasar, "UML diagram synthesis techniques: a systematic mapping study," Carleton University, Ottawa2015.

[38] B. Hoisl and S. Sobernig, "Consistency Rules for UML-based Domain-specific Language Models: A Literature Review," in *1st International Workshop on UML Consistency Rules (WUCOR 2015) co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015)*, Ottawa, Canada, 2015.

[39] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software,* vol. 80, pp. 571–583, 2007.

[40] T. Mens, R. Van der Straeten, and J. Simmonds, "A framework for managing consistency of evolving UML models,"in Proceedings of Software Evolution with UML and XML, Hershey 2005.

[41] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille, "Consistency problems in UML-based software development,"in Proceedings of International Conference on UML Modeling Languages and Applications, Lisbon, Portugal, 2005.

[42] G. Engels, J. H. Hausmann, and R. Heckel, "Testing the consistency of dynamic UML diagrams,"in Proceedings of Integrated Design and Process Technology, Pasadena, California, 2002.

[43] G. Engels, R. Heckel, and J. M. Küster, "Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model,"in Proceedings of 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, Toronto, Ontario, Canada, 2001.

[44] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen, "A methodology for specifying and analyzing consistency of object-oriented behavioral models," *Sigsoft Software Engineering Notes,* vol. 26, pp. 186-195, September 2001 2001.

[45] R. Wieringa, N. A. M. Maiden, N. R. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Eng.,* vol. 11, pp. 102–107, 2006.

[46] ProQuest, "RefWorks - A web-based bibliography and database manager " http://www.refworks.com/2014.

[47] D. Torre, Y. Labiche, and M. Genero, "UML consistency rules: a systematic mapping study," Carleton University, Ottawa2014.

APPENDIX A

This appendix contains TABLE 3 in which the final set of 116 UML consistency rules is presented. The top row in grey presents the reference number (see TABLE 2) and the name of the UML diagram/s involved. The first column is the univocal number of each UML consistency rule. The second column represents the description of the rules. The third column shows H, V, I, O, or E depending on whether the rule covers the Horizontal Consistency, Vertical Consistency, Evolution Consistency, Invocation Consistency, or Observation Consistency dimension. The fouth column shows Sy or Se based on whether Syntactic or Semantic rules are described. The following tables contain the following data:

- type(s) of UML diagram(s) involved in the UML consistency rules (grey row) with the appropriate ID# presented in TABLE 2);
- the univocal number of each UML consistency rule;
- the description of the UML consistency rules;
- UML consistency dimensions (for definitions, see Section 4 of Appendix B):
    - H: which corresponds to Horizontal Consistency;
    - V: which corresponds to Vertical Consistency;
    - I: which corresponds to Invocation Consistency;
    - O: which corresponds to Observation Consistency;
    - E: which corresponds to Evolution Consistency;
    - Sy: Syntactic Consistency;
    - Se: Semantic Consistency.

TABLE 3
SET OF UML CONSISTENCY RULES

1 - State Machine Diagram				
1	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. The initial states of the state machine diagrams U' and U must be identical.	**I**	**Se**	[1]
2	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. Every transition of U' which is already in U has at least the same source states and sink states as it has in U.	**I**	**Se**	[1]
3	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. For each transition t in U' that is already present in U, the guard condition g'(t) in U' must be at least as strong as the guard condition g(t) for t in U: g'(t) → g(t).	**I**	**Se**	[1]
4	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. A transition of U in U' cannot therefore receive an additional source state or sink state that is already present in U.	**I**	**Se**	[1]
5	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. A transition added to U' does not receive a source state or a sink state that was already present in U.	**I**	**Se**	[1]
6	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. The set of transitions of U' is a superset of the set of transitions of U.	**I**	**Se**	[1]
7	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. A transition in U' which is already present in U has in U' at most the source states that the transition has in U.	**I**	**Se**	[1]
8	Consider two State Machines U' and U of a class O' and its superclass O, where U' extends the state machine of U by adding states and transitions. For each transition t in U' that is already present in U, U: g(t) → g'(t).	**I**	**Se**	[1]
9	Consider two State Machine U’’ and U of a class O’’ and its superclass O, where U’’ refines the state diagram of U by means of a refinement function h that maps transitions onto transitions and states of U’’ and that maps simple states of U’’ onto simple states of U. Intuitively, the concept of the refinement function means that if a simple state s of U has been refined to a composite state in U’’, then h maps the substates of s in U’’ and transitions between these states to s, and h maps the transitions in U’’ that are incident to the substates of s into transitions of U that are incident to s. s is a state in U, so s Є U (from the rule: "a simple state s of U","s of h(t')") s' is a refined state in U', so s' Є U' (from the rule: "a state s' in U' ","sink state s' of t' in U' ") Source+(t): t → set of states. Source+(t) = { source of t}[2] U sub-states*{source of t} s' Є Source+(t) For every transition t' in S': for every source state s of h(t'), there exists a state s' in S' such that h(s') =	**E**	**Se**	[1]

	s, and for every sink state s of h(t') there exists a sink state s' of t' in S' such that h(s') = s. Informal rule definition: A transition t'' (of U'') that is refined from t (of U) must have s'' (state sender and receiver of U'') that is refined from s (of U) Є Source+(t).			
10	Consider two State Machine U'' and U of a class O'' and its superclass O, where U'' refines the state diagram of U by means of refinement function h that maps transitions onto transitions and states of U'' and that maps simple states of U'' onto simple states of U. Intuitively, the concept of the refinement function means that if a simple state s of U has been refined to a composite state in U'', then h maps the substates of s in U'' and transitions between these states to s, and h maps the transitions in U'' that are incident to the substates of s into transitions of U that are incident to s. s is a state in U, so s Є U (from the rule: "a simple state s of U","s of h(t')") s' is a refined state in U', so s' Є U' (from the rule: "a state s' in U' ","sink state s' of t' in U' ") Source+(t): t → set of states. Source+(t) = { source of t} U sub-states*{source of t} s' Є Source+(t) For every source state s' of a transition t' in S'', where s'' and t'' do not belong to the same refined state (i.e., h(s'') /= h(t'')), h(s'') is therefore a source state of h(t''), and for every sink state s'' of a transition t'' in S'', where s' and t' do not belong to the same refined state, h(s'') is therefore a sink state of h(t''). Informal rule definition: This rule is about transitions that go between states that are not substates of a complex state that is a refinement of a simple state. The rule says that such transitions have their sources and sinks in U' as refinements of those ones in U.	**E**	**Se**	[1]
11	Using a signal/message on a transition in a state diagram that no object sends, creates structural and syntactic inconsistencies. The information about the sending object can be found in another state machine, in another partition of the same state machine, in a sequence diagram, or any other diagram where one can specify an object can send a signal/message.	**H**	**Sy**	[3]
12	A state machine should be deadlock-free.	**H**	**Se**	[4-6]
13	A state machine must be deterministic, that is, in every state, only one transition (accounting for the different levels of nested states) should fire on a reception of an event.	**H**	**Sy**	[5, 7, 8]
14	An abstract operation cannot be invoked in a state machine.	**H**	**Sy**	[9]
2 - Class Diagram, State Machine Diagram and Activity Diagram				
15	There is an inconsistency if a precondition on an operation is in contradiction with a state machine or an activity diagram including a call of that operation.	**H**	**Sy**	[10]
3 - Sequence Diagram and Use Case Diagram				
16	If a use case is further specified by one or more sequence diagram, then every scenario described in one of those sequence diagrams should match a sequence of steps in that use case's use case description.	**H**	**Se**	[11, 12]
17	If a sequence diagram depicts all the behavior required for successful completion of a use case, it follows that each post-condition specified in the use case description must be achieved by a message (or a set of messages) in the sequence diagram (including referred diagrams) for that use case.	**H**	**Sy**	[13]
18	If a sequence diagram depicts all the behavior required for successful completion of a use case, it follows that results achieved by alternative message flows in the sequence diagram (including referred diagrams) must correspond to post-conditions specified in the use case description.	**H**	**Sy**	[13]
19	Each action specified or implied in a use case description should be detailed in a corresponding message or set of messages in the sequence diagram corresponding to that use case. Depending on the clarity and completeness of the use case description text, the author of the sequence diagram may need to infer some of the operations.	**H**	**Se**	[13]
20	A use case is complemented by a set of sequence diagrams, each sequence diagram representing an alternative scenario of the use case.	**H**	**Se**	[13]
4 - Activity Diagram (*<u>no rule</u>*)				
5 - Sequence Diagram				
21	Arguments to messages must represent information that is known to the sender. This includes attribute values of the sender, navigation expressions starting with the sender, constants; This should account for inheritance.	**H**	**Sy**	[13]
22	Variables used in the guard of a message must represent information that is known to the sender. This includes attribute values of the sender, navigation expressions starting with the sender, constrants; This should account for inheritance.	**H**	**Sy**	[13-16]
23	If SD2 is a sequence diagram referred to by an Interaction Use (a nested sequence diagram) embedded in sequence diagram SD1, then for every matched pair of message to and from SD2 in SD1, there is a corresponding matched pair of sourceless message and targetless message in SD2.	**H**	**Sy**	[14, 16]
24	In a sequence diagram, if an attribute is assigned the return value of a message, then the types have to be compatible	**H**	**Sy**	[9]
25	Return messages from ExecutionSpecification instances should always be shown.	**H**	**Sy**	[13]

26	Arguments of messages should always be shown.	H	Sy	[13]
6 – Object Diagram and Collaboration Diagram (*no rule*)				
7 - Object Diagram and Class Diagram				
27	The number of occurrences of a link in an object diagram, an instance of an association in a class diagram, must satisfy the multiplicity constraints specified for the association.	H	Sy	[17]
8 - Activity Diagram and Class Diagram				
28	A class name that appears in an activity diagram also appears in the class diagram.		Sy	[11, 18-20]
29	An action that appears in an activity diagram must also appear in the class diagram as an operation of a class.	H	Sy	[12, 18, 20]
30	When an activity diagram specifies an exachange of information, through control or data flow, between two different instances, the class diagram should specify a mechanism (e.g., direct association) so that those instances can indeed communicate.	H	Sy	[18]
31	Swimlanes (Activity pattern in UML 2.0) in an Activity diagram (represented as className in activity state) must be present as a unique class in a class diagram.	H	Sy	[19]
9 - State Machine Diagram and Activity Diagram (*no rule*)				
10 – Use Case Diagram and State Machine Diagram (*no rule*)				
11 - Class Diagram and State Machine Diagram				
32	When a state machine specifies the behavior of a class, the actions and activities in the state machine should be operations of the class (in the class diagram) which behavior the state machine specifies. An action or activity can be part of a navigation expression, in which case the navigation expression must legal according to the class diagram and the context (class) of the specified behavior in the state machine.	H	Sy	[3, 7, 9, 18, 21-23]
33	When a state machine specifies the behavior of a class, any property (i.e., attribute, navigation) in the state machine should be one of the class (in the class diagram) which behavior the state machine specifies. A property can be part of a navigation expression, in which case the navigation expression must legal according to the class diagram and the context (class) of the specified behavior in the state machine.	H	Sy	[9, 24]
34	When the state machine diagram specifies the behaviour of a class in the class diagram, the class is an active class with a classifierBehavior, then the triggers of the transitions in the state machine diagram are operations of the active class.	H	Sy	[3, 10, 25, 26]
35	No operation can be called on a state machine or from a state machine if this breaks the visibility rules of the class diagram (public, protected, private).	H	Sy	[9]
36	When behaviour is triggered from a state machine diagram (e.g., calling an operation, sending a signal) that describes the behaviour of a class, and the triggered behaviour belongs to another class, then the former must have a handle to the latter as specified in the class diagram. Another way of saying this is that the former must have visibility to the latter. A specific case of this situation is when the former class has an association (possibly inherited) to the latter class.	H	Sy	[9, 23]
37	For a send action, there should be a reception within the classifier of the receiver instance that corresponds to the signal of the send action describing the expected behavior response to the signal.	H	Sy	[23]
38	For all call or send events specified in the class diagram for a classifier (context) which behavior is specified with a state machine, there should be transitions in that state machine describing the detailed behavior of the events.	H	Sy	[23]
12 – Protocol State Machine Diagram and Class Diagram (*no rule*)				
13 - Sequence Diagram and Activity Diagram				
39	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, the different diagrams should specify the same scenarios: e.g., same sequence of messages/operations/actions, same branching or repetition conditions.	H	Sy	[12, 27]
40	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a flow of interaction between objects in an activity diagram should be a flow of interactions between the same objects in a sequence diagram.	H	Sy	[18]
41	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a sequence of messages in the sequence diagram, uninterupted by control flow structures, must correspond to one activity node in the activity diagram which name is the series of message names of the sequence diagram.	H	Sy	[27]
42	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a synchronous message between objects running in different threads of control in the sequence diagram must match a join node for the receiving side (thread) in the corresponding activity diagram, and a fork node for the asynchronous reply.	H	Sy	[27]
43	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, an asynchronous creation of an active object in the sequence diagram must match a fork node in the corresponding activity diagram: the input action node is matching the calling thread message; two outgoing edges should flow out of the fork node, one for the node matching continuation of execution in the calling thread and one for the node matching the newly created active object.	H	Sy	[27]
44	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, an asynchronous message sent between two active objects in the sequence diagram should match, in the corresponding activity diagram, a join node on the receiver side and a fork node on the sender side.	H	Sy	[27]

45	If an activity diagram shows scenarios of an operation and that operation appears in a sequence diagram, a InteractionUse in the sequence diagram must match an activity node in the activity diagram that refers to the activity diagram matching the sequence diagram referred to in the InteractionUse.	**H**	**Sy**	[27]
14 – Use Case Diagram and Object Diagram (*no rule*)				
15 - State Machine Diagram and Communication Diagram				
46	When one specifies an active class, i.e., one that has a state-based behaviour described in a state machine diagram, and an instance of this active class is used in a communication diagram, the messages sent to this objects and emitted by this object as specified in the communication diagram must comply to the protocol specified in the state machine diagram.	**H**	**Sy**	[18, 28]
16 - Communication Diagram and Class Diagram				
47	In order for objects to exchange messages in a communication diagram, the sending object must have a handle to the receiving object as specified in the class diagram. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	**H**	**Sy**	[29]
17 - State Machine Diagram and Sequence Diagram				
48	When one specifies an active class, i.e., one that has a state-based behaviour described in a state machine diagram, and an instance of this active class is used in a sequence diagram, the messages sent to this objects and emitted by this object as specified in the sequence diagram must comply (e.g., sequence and types of signals, receivers and emitters of signals) to the protocol specified in the state machine diagram.	**H**	**Sy**	[1, 18, 21, 24, 30-44]
18 – Protocol State Machine Diagram (*no rule*)				
19 - Communication Diagram				
49	If communication diagram A is a specialization of communication diagram B, all the messages present in B have to be included in A.	**E**	**Sy**	[45]
20 - Use Case Diagram and Class Diagram				
50	The noun of the use case's name should equal the name of one class in the class diagram.	**H**	**Sy**	[11, 12, 18, 20, 46]
51	The verb of the use case's name should equal the name of an operation of a class in the class diagram.	**H**	**Sy**	[11, 18, 20, 46]
52	Entity classes correspond to data manipulated by the system as described in a use case description.	**H**	**Sy**	[21]
21 – Object Diagram and Activity Diagram (*no rule*)				
22 - Communication Diagram and Activity Diagram				
53	If an activity diagram specifies a flow of interaction between objects, and those objects (or a subset of those objects) appears in a communication diagram, then the communication diagram should specify the same flow of interaction.	**H**	**Sy**	[18]
23 - Communication Diagram and Class Diagram				
54	Objects involved in a communication diagram should be instances of classes of the class diagram.	**H**	**Sy**	[18, 47-49]
55	In order for objects to exchange messages in a communication diagram, the sending object must have a handle to the receiving object as specified in the class diagram. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	**H**	**Sy**	[18, 49, 50]
56	Each class in the class diagram appears with at least one instance in at least one communication diagram.	**H**	**Sy**	[15, 18, 47-49]
24 – Communication Diagram and Use Case Diagram (*no rule*)				
25 - Use Case Diagram and Activity Diagram				
57	If a use case U (the including use case) includes use case V (the included use case) in the use case diagram, and both use case U's flows and use case V's flows are specified as activity diagrams, then the activity diagram specifying use case U should contain an action node (likely a CallBehaviorAction node) that refers to the activity diagram specifying use case V.	**H**	**Sy**	[11, 51, 52]
58	Each use case is described by at least one activity diagram.	**H**	**Sy**	[51, 52]
59	Each event (flow of steps) specified or implied in the use case description should be detailed in a corresponding event of the activity diagram. This rule is valid only if the activity diagram further specifies the use case.	**H**	**Sy**	[20]
60	An actor that is associated with a use case will be an activity partition in the activity diagram describing that use case.	**H**	**Sy**	67]
26 - Use Case Diagram				
61	The use case description of a use case in the use case diagram should contain at least one event (flow of steps). Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**	[53]
62	An event (flow of steps) in a use case description of a use case of the use case diagrm has to be of either a basic or an alternate type. Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**	[53]
63	Each use case in the use case diagram must have a use case description specifying event flows. Even if a use case description may be shown/represented in a separate view respect the use case diagram, we consider that a use case comes with a description.	**H**	**Sy**	[21]

64	The name of a use case must include a verb and a noun (for instance “Validate User”).	H	Sy	[46]
	28 - Class Diagram			
65	A class invariant must be satisfied by any non-trivial instantiation of the class diagram (i.e., an instantiation that is not reduced to having no instance of any class).	H	Sy	[10]
66	The type of a relationship between two classes at a (high) level of abstraction (e.g., plain association, aggregation, composition, generalization) must be the same as the type of a refinement of that relationship at a more concrete (low) level of abstraction. For instance, a plain association at a low level of abstraction being abstracted as an aggregation (a high level of abstraction) denotes an inconsistency.	V	Se	[54]
67	A class diagram is consistent if it can be instantiated without violating any of the constraints in the diagram (e.g., association end multiplicities).	H	Sy	[55]
68	Two classes are equivalent if they denote the same set of instances whenever the constraints imposed by the class diagram are satisfied. Determining equivalence of two classes allows their merging, thus reducing the complexity of the diagram.	H	Sy	[55]
69	A class relationship at a (low) level of abstraction must have an abstraction at a higher level of abstraction, either a class or a relation.	V	Se	[54, 56]
70	If a class relationship A refines relation B, A must have the same destinations classes as the destination classes of the abstraction of B.	V	Se	[54]
71	If a class relationship A refines relation B, A must have the same type as B.	V	Se	[54]
72	The group of relationships between any two high level classes must be identical with the group of relationships between their corresponding low-level classes: ensures the same interactions among low level classes and high-level classes.	V	Se	[54]
73	If a navigation expression is used in an operation contract, then the expression must be a legal one (according to the syntax of the language and the class diagram).	H	Sy	[9]
74	This Liskov’s substitution principle holds.	O	Se	[1, 57-59]
75	A class that realizes an interface must declare all the operations in the interface with the same signatures (including parameter direction, default values, concurrency, polymorphic property, query characteristic).	H	Sy	[9, 60]
76	An abstract operation can only belong to an abstract class.	H	Sy	[25]
77	If an operation appears in a pre or post condition then it must have the property isQuery equal to true.	H	Sy	[9]
78	No (public) method of a class violates, as indicated by its pre and post-conditions, the class invariant of that class.	H	Sy	[61]
79	In a class, the names of the association ends (on the opposite side of associations from this class) and the names of the attributes (of the class) are different.	H	Sy	[62]
80	No precondition should violate the class invariant.	H	Sy	[9]
81	No post-condition should violate the class invariant.	H	Sy	[9]
82	A class that contains an abstract operation must be abstract.	H	Sy	[9]
83	There must be no cycle in the directed path of aggregation associations: A class cannot be a part of an aggregation in which it is the whole; A class cannot be a part of an aggregation in which its superclass (or ancestor) is the whole.	H	Sy	[9]
84	A class cannot be a part of more than one composition - no composite part may be shared by two composite classes.	H	Sy	[9]
85	Each concrete class, i.e., it is not abstract, must implement all the abstract operations of its superclass(es).	H	Sy	[9]
86	If an attribute’s type is a class, then that class has to be visible to the class containing the attribute. Example: same package or there exists a path in the class diagram that allows the class containing the attribute to have a hold on that type.	H	Sy	[9]
87	If the return type of an operation is a class, then that class has to be visible to the class containing the operation. Example: same package or there exists a path in the class diagram that allows the class containing the operation to have a hold on that type.	H	Sy	[9]
88	An operation may not be overridden by a descendant class only if its isLeaf attribute (from metaclass RedefinableElement) is defined accordingly.	H	Sy	[9]
89	A static operation cannot access an instance attribute (as indicated by its pre and post conditions, for instance).	H	Sy	[9]
90	A static operation cannot invoke an instance operation (as indicated by its pre and post conditions, for instance).	H	Sy	[9]
91	If an association end has a private visibility, then the class at this end can only be accessed via the association by the class at the other association end (e.g., as indicated by pre and post conditions of operations of the class at this other end of the association).	H	Sy	[9]
92	If an association end has a protected visibility, then the class at this end can only be accessed via the association, by the class at the other association end and its descendants (e.g., as indicated by pre and post conditions of operations of the class at this other end of the association).	H	Sy	[9]
93	The multiplicity range for an attribute must be adhered to by all elements (operation contracts, guard conditions) that access it.	H	Se	[9]
94	For class A's operations to use another class B, as indicated by contracts in A, there must be a means (e.g., in the form of a path involving associations, generalization and/or dependencies) in the class diagram for A to get a hold on B.	H	Sy	[9]

95	A class at a low-level of abstraction refines at most one class from a higher-level of abstraction.	V	Sy	76]
96	A class at a high-level of abstraction is refined by at least one class from a lower-level of abstraction.	V	Sy	[56]
97	There should not be semantically redundant paths between any two classes in the class diagram graph, unless precisely specified by a constraint (e.g., specified in OCL). For instance, from class A, it may be possible to navigate as self.theA.theB, along with self.theC.theB. The question is then whether the two collections self.theA.theB and self.theC.theB are identical.	H	Sy	[63]
29 – Interaction Diagram and Class Diagram (*no rule*)				
30 - Use Case Diagram and Interaction Diagram				
98	Each interaction diagram corresponds to a use case in the use case diagram, and each use case in the use case diagram is specified by an interaction diagram.	H	Sy	[21]
99	If a use case is further specified by one or more interaction diagrams, then every scenario described in one of those interaction diagrams should match a sequence of steps in that use case's use case description.	H	Sy	[21]
31 – Protocol State Machine Diagram and State Machine Diagram (*no rule*)				
32 - Composite Structure Diagram				
100	A delegation connector connects a port on the boundary of a classifier to a port on the internal structure (or parts) of the classifier. It basically delegates signals or operation calls arriving on the boundary port to the internal port, or those coming from the internal port to the boundary port. Therefore, these ports at the end of the delegation connector must have the same (or compatible) interfaces. If both ports require the interface, then the direction of delegation is from the boundary port to the internal port. If both ports provide the interface, then the direction of delegation is from the internal port to the boundary port.	H	Sy	[64]
101	If an assembly connector exists between two ports, one of the ports (the source) must be a required port and the other (the destination) must be a provided port. This rule describes the opposite case of delegation, where both ports at the end of an assembly connector have conjugate interfaces (one port requires an interface; the other provides the same interface). What matters is that the two ports must either have the same interface but one of them is marked as isConjugate, while the other is not, or they should have conjugate interfaces (i.e., ones that have the same operations or signal receptions that are annotated as provided on one interface and required on the other).	H	Sy	[64]
102	If a connector is typed with an association, the direction of the association must conform to the direction of the connector as derived from the direction of the ports at its ends (association navigable from class A to class B if the connector between A and B indicates that A requires services that B provides). Given that the direction of associations and connectors (however, it is calculated) could be encoded using the order of their 'memberEnd' and 'end' collection, respectively, the rule is basically saying that such direction should be the same in both cases, if a connector is typed by an association.	H	Sy	[64]
103	If a link outgoing from a port is statically typed with an association, then the association must be navigable to an interface which is part of the set of interfaces and the type indicated by the association must belong to the set of transported interfaces for that link.	H	Sy	[64]
104	If the link originates from a component, then the link must be statically typed with an association, and the type of the entity at the other end of the link must be compatible with (i.e. be equal or a subtype of) the type at the corresponding end of the association.	H	Sy	[64]
105	The set of transported interfaces by a link should not be empty.	H	Sy	[64]
106	If several non-typed connectors start from one port, then the sets of interfaces transported by each of these connectors have to be pair wise disjoint.	H	Sy	[64]
107	The union of the sets of interfaces transported by each of the connectors originating from a port P must be equal to the set of interfaces provided/required by P.	H	Sy	[64]
33 - Sequence Diagram and Class Diagram				
108	The type of a lifeline (type of the connectable element of the lifeline) in a sequence diagram must not be an interface nor an abstract class.	H	Sy	[9, 15]
109	In case a message in a sequence diagram is referring to an operation, that operation must not be abstract.	H	Sy	[9, 15]
110	If a message in a sequence diagram refers to an operation, through the signature of the message, then that operation must belong, as per the class diagram, to the class that types the target lifeline of the message.	H	Sy	[9, 11, 13-16, 18, 21, 22, 25, 33, 39, 49, 53, 61, 65-78]
111	Interactions between objects in a sequence diagram, specifically the numbers of types of interacting objects, must comply with the multiplicity restrictions specified by the class diagram (e.g., association end multiplicities).	H	Sy	[14, 16, 57, 71, 79]
112	In order for objects to exchange messages in a sequence diagram, the sending object must have a handle to the receiving object as specified in the class diagram. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.	H	Sy	[9, 13-16, 18, 22, 23, 39, 43, 49,

				57, 61, 65-67, 69, 71, 73-77, 79, 80]
113	The behavioral semantics of a composition or aggregation association in the class diagram must be inferred in sequence diagrams. For instance, in a whole-part (composition) relation, the part should not outlive the whole.	**H**	**Sy**	[13, 79]
114	Each public method in a class diagram triggers a message in at least one sequence diagram.	**H**	**Sy**	[15, 24]
115	Each class in the class diagram must be instantiated in a sequence diagram.	**H**	**Sy**	[13, 14, 18, 22, 48, 49, 70, 80]
116	No operation can be used in a message of a sequence diagram if this breaks the visibility rules of the class diagram (public, protected, private).	**H**	**Sy**	[9]
34 - Sequence Diagram and Communication Diagram (*no rule*)				
35 - Class Diagram, State Machine Diagram and Communication Diagram (*no rule*)				
36 - Class Diagram, Sequence Diagram and Use Case Diagram (*no rule*)				

REFERENCES OF CONSISTENCY RULES

[1] M. Stumptner and M. Schrefl, "Behavior consistent inheritance in UML,"in Proceedings of 19th international conference on Conceptual modeling, Salt Lake City, Utah, USA, 2000.

[2] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Eng.,* vol. 11, pp. 102-107, 2005.

[3] L. A. Campbell, B. H. C. Cheng, W. E. McUmber, and R. E. K. Stirewalt, "Automatically Detecting and Visualising Errors in UML Diagrams," *Requirements Engineering,* vol. 7, pp. 264-287, 2002.

[4] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen, "A methodology for specifying and analyzing consistency of object-oriented behavioral models," *Sigsoft Software Engineering Notes,* vol. 26, pp. 186-195, September 2001 2001.

[5] C. Schwarzl and B. Peischl, "Static- and dynamic consistency analysis of UML state chart models,"in Proceedings of 13th international conference on Model driven engineering languages and systems: Part I, Oslo, Norway, 2010.

[6] H. Qiu, "Consistency Checking of UML-LIGHT with CSP," WS 2004/2005: Seminar : Modellbasierte Softwareentwichklung Department of Computer Science, University of Paderborn, Paderborn, Germany 2004.

[7] G. Costagliola, V. Deufemia, F. Ferrucci, and C. Gravino, "Exploiting Visual Languages Generation and UML Meta Modeling to Construct Meta-CASE Workbenches," *Electronic Notes in Theoretical Computer Science,* vol. 72, pp. 25-35, 2003.

[8] Z. Pap, I. Majzik, A. Pataricza, and A. Szegi, "Methods of checking general safety criteria in UML statechart specifications," *Reliability Engineering & System Safety,* vol. 87, pp. 89-107, 2005.

[9] L. C. Briand, Y. Labiche, and L. O'Sullivan, "Impact analysis and change management of UML models,"in Proceedings of International Conference on Software Maintenance, Amsterdam, The Netherlands, 2003.

[10] E. Astesiano and G. Reggio, "An attempt at analysing the consistency problems in the UML from a classical algebraic viewpoint,"in Proceedings of 16th International Workshop, Frauenchiemsee, Germany, 2003.

[11] P. G. Sapna and H. Mohanty, "Ensuring consistency in relational repository of UML models,"in Proceedings of 10th International Conference on Information Technology, Orissa, India, 2007.

[12] C. F. Borba and A. E. A. Da SIlva, "Knowledge-based system for the maintenance registration and consistency among UML diagrams,"in Proceedings of 20th Brazilian Conference on Advances in Artificial Intelligence, São Bernardo do Campo, Brazil, 2010.

[13] M. Hilsbos and I.-Y. Song, "Use of Tabular Analysis Method to Construct UML Sequence Diagrams,"in Proceedings of 23th International Conference on Conceptual Modeling, Shanghai, China, 2004.

[14] J. Yang, Q. Long, Z. Liu, and X. Li, "A predicative semantic model for integrating UML models,"in Proceedings of 1st international Conference on Theoretical Aspects of Computing, Guiyang, China, 2004.

[15] R. F. Paige, D. S. Kolovos, and F. A. C. Polack, "Refinement via Consistency Checking in MDA," *Electronic Notes in Theoretical Computer Science,* vol. 137, pp. 151-161, 2005.

[16] L. Quan, L. Zhiming, L. Xiaoshan, and J. He, "Consistent code generation from UML models,"in Proceedings of Australian Conference on Software Engineering, Brisbane, Australia, 2005.

[17] J. H. Hausmann, R. Heckel, and S. Sauer, "Extended model relations with graphical consistency conditions,"in Proceedings of UML 2002 Workshop on Consistency Problems in UML-based Software Development, Blekinge Institute of Technology, 2002.

[18] A. Ohnishi, "A supporting system for verification among models of the UML," *Systems and Computers in Japan,* vol. 33, pp. 1-3, April 2002 2002.

[19] J. Chanda, A. Kanjilal, S. Sengupta, and S. Bhattacharya, "Traceability of requirements and consistency verification of UML use case, activity and Class diagram: A Formal approach,"in Proceedings of International Conference on Methods and Models in Computer Science, New Delhi, India, 2009.

[20] A. Kanjilal, S. Sengupta, and S. Bhattacharya, "Analysis of object-oriented design: A metrics based approach,"in Proceedings of IEEE Region 10

Annual International Conference, Singapore; Singapore, 2009.

[21] Y. Labiche, "The UML is more than boxes and lines,"in Proceedings of Workshops and Symposia at MODELS 2008, Toulouse, France, 2008.

[22] H. Il-Kyu and K. Byung-Wook, "Meta-validation of UML structural diagrams and behavioral diagrams with consistency rules,"in Proceedings of IEEE Pacific Rim Conference on Communications, Computers and signal Processing, Victoria, B.C., Canada, 2003.

[23] S. K. Kim and D. Carrington, "A formal object-oriented approach to defining consistency constraints for UML models,"in Proceedings of Australian Conference on Software Engineering, Melbourne, Australia, 2004.

[24] J. Kienzle, W. A. Abed, and J. Klein, "Aspect-oriented multi-view modeling,"in Proceedings of 8th ACM international conference on Aspect-oriented software development, Charlottesville, Virginia, USA, 2009.

[25] O. Vasilecas, R. Dubauskaitė, and R. Rupnik, "Consistency checking of UML business model," *Technological and Economic Development of Economy,* vol. 17, pp. 133-150, 2011.

[26] J. Yang, "A framework for formalizing UML models with formal language rCOS,"in Proceedings of 4th International Conference on Frontier of Computer Science and Technology, Shanghai, China, 2009.

[27] N. Pattanasri, V. Wuwongse, and K. Akama, "XET as a Rule Language for Consistency Maintenance in UML,"in Proceedings of 3rd International Workshop, Hiroshima, Japan, 2004.

[28] A. Zisman and A. Kozlenkov, "Knowledge Base Approach to Consistency Management of UML Specifications,"in Proceedings of 16th IEEE international conference on Automated software engineering, Coronado Island, San Diego, CA, USA, 2001.

[29] F. J. L. Martínez and A. T. Álvarez, "A precise approach for the analysis of the UML models consistency,"in Proceedings of 24th international conference on Perspectives in Conceptual Modeling, Klagenfurt, Austria, 2005.

[30] B. Bjørn, "Consistency checking UML interactions and state machines," Master Thesis, Department of Informatics, University of Oslo, Oslo, Norway, 2008.

[31] D. Du, J. Liu, H. Cao, and M. Zhang, "BAS: A Case Study for Modeling and Verification in Trustable Model Driven Development," *Electronic Notes in Theoretical Computer Science,* vol. 243, pp. 69-87, July 2009 2009.

[32] D. Ruta and V. Olegas, "The approach of ensuring consistency of UML model based on rules,"in Proceedings of 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies, Sofia, Bulgaria, 2010.

[33] A. Egyed, "Fixing Inconsistencies in UML Design Models,"in Proceedings of 29th international conference on Software Engineering Minneapolis, MN, USA, 2007.

[34] H. Wang, T. Feng, J. Zhang, and K. Zhang, "Consistency check between behaviour models,"in Proceedings of International Symposium on Communications and Information Technology, Beijing, China, 2005.

[35] P. Pelliccione, P. Inverardi, and H. Muccini, "CHARMY: A Framework for Designing and Verifying Architectural Specifications," *IEEE Transactions on Software Engineering,* vol. 35, pp. 325-346, 2009.

[36] J. Kuster and J. Stroop, "Consistent design of embedded real-time systems with UML-RT,"in Proceedings of 4th International Symposium on Object-Oriented Real-Time Distributed Computing, Magdeburg, Germany, 2001.

[37] W. Shengjun, J. Longfei, and J. Chengzhi, "Ontology Definition Metamodel based Consistency Checking of UML Models,"in Proceedings of 10th International Conference on Computer Supported Cooperative Work in Design, Nanjing, China, 2006.

[38] K. N. Chang, "Model checking consistency between sequence and state diagrams,"in Proceedings of International Conference on Software Engineering Research and Practice, Las Vegas, NV, USA, 2008.

[39] A. Egyed, E. Letier, and A. Finkelstein, "Generating and evaluating choices for fixing inconsistencies in UML design models,"in Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Italy, 2008.

[40] A. Nimiya, T. Yokogawa, H. Miyazaki, S. Amasaki, Y. Sato, and M. Hayase, "Model checking consistency of UML diagrams using alloy," *World Academy of Science, Engineering and Technology,* vol. 71, pp. 547-550, 2010.

[41] X. Zhao, Q. Long, and Z. Qiu, "Model checking dynamic UML consistency,"in Proceedings of 8th International Conference on Formal Engineering Methods, Macao, China, 2006.

[42] Y. Hammal, "A modular state exploration and compatibility checking of UML dynamic diagrams,"in Proceedings of 6th IEEE/ACS International Conference on Computer Systems and Applications, Doha, Qatar, 2008.

[43] K. Lano, "Formal specification using interaction diagrams,"in Proceedings of 5th IEEE International Conference on Software Engineering and Formal Methods, London; United Kingdom, 2007.

[44] M. N. Alanazi and D. A. Gustafson, "Super state analysis for UML state diagrams,"in Proceedings of 2009 WRI World Congress on Computer Science and Information Engineering Los Angeles, California, USA, 2009.

[45] H. Nakanishi, T. Miura, and I. Shioya, "Formalizing UML collaborations by using description logics,"in Proceedings of 2nd IEEE International Conference on Computational Cybernetics, Vienna, Austria, 2004.

[46] C. M. Zapata, G. González, and A. Gelbukh, "A rule-based system for assessing consistency between UML models,"in Proceedings of 6th Mexican International Conference on Advances in Artificial Intelligence, Aguascalientes, Mexico, 2007.

[47] R. g. Laleau and F. Polack, "Using formal metamodels to check consistency of functional views in information systems specification," *Information and Software Technology,* vol. 50, pp. 797-814, 2008.

[48] R. F. Paige, P. J. Brooke, and J. S. Ostroff, "Metamodel-based model conformance and multiview consistency checking," *ACM Transactions Software Engineering Methodology,* vol. 16, p. 11, July 2007 2007.

[49] H. Il-kyu and B. Kang, "Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram,"in Proceedings of

9th International Conference on Intelligent Data Engineering and Automated Learning, Daejeon, South Korea, 2008.

[50] H. Hamed and A. Salem, "UML-L: a UML based design description language,"in Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon, 2001.

[51] T. Mens, R. Van Der Straeten, and M. D'Hondt, "Detecting and resolving model inconsistencies using transformation dependency analysis,"in Proceedings of 9th International conference on Model Driven Engineering Languages and Systems, Genova, Italy, 2006.

[52] N. Ibrahim, R. Ibrahim, M. Z. Saringat, D. Mansor, and T. Herawan, "Consistency rules between UML use case and activity diagrams using logical approach," *International Journal of Software Engineering and its Applications,* vol. 5, pp. 119-134, 2011.

[53] S. Sengupta and S. Bhattacharya, "Formalization of UML diagrams and their consistency verification: A Z notation based approach,"in Proceedings of 1st India software engineering conference, Hyderabad, India, 2008.

[54] A. Egyed, "Consistent adaptation and evolution of class diagrams during refinement,"in Proceedings of 7th International Conference (FASE 2004), Held as Part of the Joint European Conferences on Theory and Practice of Software, Barcelona, Spain, 2004.

[55] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini, "A Formal Framework for Reasoning on UML Class Diagrams,"in Proceedings of 13th International Symposium on Foundations of Intelligent Systems, Lyon, France, 2002.

[56] W. Shen, K. Wang, and A. Egyed, "An efficient and scalable approach to correct class model refinement," *IEEE Transactions on Software Engineering,* vol. 35, pp. 515-533, 2009.

[57] R. Van Der Straeten, "Inconsistency detection between UML models using RACER and nRQL,"in Proceedings of the KI-2004 Workshop on Applications of Description Logics, Ulm, Germany, 2004.

[58] G. Engels, R. Heckel, and J. M. Küster, "Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model,"in Proceedings of 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, Toronto, Ontario, Canada, 2001.

[59] R. Van Der Straeten, V. Jonckers, and T. Mens, "A formal approach to model refactoring and model refinement," *Software & Systems Modeling,* vol. 6, pp. 139-162, 2007.

[60] R. Wagner, H. Giese, and U. Nickel, "A plug-in for flexible and incremental consistency management,"in Proceedings of International Conference on the Unified Modeling Language 2003 (Workshop 7: Consistency Problems in UML-based Software Development), San Francisco, USA, 2003.

[61] L. Jing, L. Zhiming, H. Jifeng, and L. Xiaoshan, "Linking UML models of design and requirement,"in Proceedings of Australian Conference on Software Engineering, Melbourne, Australia, 2004.

[62] K. C. Lavanya, K. V. Bala, H. Mohanty, and R. K. Shyamasundar, "How Good is a UML Diagram? A Tool to Check It,"in Proceedings of IEEE Region 10, Melbourne, Australia, 2005.

[63] M. Snoeck, C. Michiels, and G. Dedene, "Consistency by construction: the case of MERODE,"in Proceedings of Conceptual Modeling for Novel Application Domains, Workshops ECOMO, IWCMQ, AOIS, and XSDM, Chicago, IL, USA, 2003.

[64] I. Ober and I. Dragomir, "Unambiguous UML composite structures: the OMEGA2 experience,"in Proceedings of 37th international conference on Current trends in theory and practice of computer science, Nový Smokovec, Slovakia, 2011.

[65] G. Spanoudakis and H. Kim, "Diagnosis of the significance of inconsistencies in object-oriented designs: a framework and its experimental evaluation," *Journal of Systems and Software,* vol. 64, pp. 3-22, October 2002 2002.

[66] A. Egyed, "UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models "in Proceedings of 29th international conference on Software Engineering, Minneapolis, MN, USA, 2007.

[67] A. Egyed, "Instant consistency checking for the UML,"in Proceedings of 28th international conference on Software engineering, Shanghai, China, 2006.

[68] H. Ledang, "B-based Consistency Checking of UML Diagrams,"in Proceedings of 2nd National Symposium on Research, Development and Application of Information and Communication Technology, Hanoi, Vietnam, 2004.

[69] R. Van Der Straeten, "ALLOY for Inconsistency Resolution in MDE,"in Proceedings of BENEVOL 2009 The 8 th BElgian-NEtherlands software eVOLution seminar, Université catholique de Louvain - Belgium, 2009.

[70] J. Chanda, T. Janowski, H. Mohanty, A. Kanjilal, and S. Sengupta, "UML-Compiler: A Framework for Syntactic and Semantic Verification of UML Diagrams,"in Proceedings of 6th international conference on Distributed Computing and Internet Technology, Bhubaneswar, India, 2010.

[71] Z. Khai, A. Nadeem, and G.-s. Lee, "A Prolog Based Approach to Consistency Checking of UML Class and Sequence Diagrams,"in Proceedings of Communication and Networking - International Conference (FGCN 2011), Held as Part of the Future Generation Information Technology Conference (FGIT 2011), in Conjunction with GDC 2011, Jeju Island, South Korea, 2011.

[72] I.-Y. Song, R. Khare, Y. An, and M. Hilsbos, "A Multi-level Methodology for Developing UML Sequence Diagrams,"in Proceedings of 27th International Conference on Conceptual Modeling, Barcelona, Spain, 2008.

[73] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, "Using Description Logic to Maintain Consistency between UML Models,"in Proceedings of 6th International Conference on Unified Modeling Language, Modeling Languages and Applications, San Francisco, CA, USA, 2003.

[74] J. Muskens, R. J. Bril, and M. R. V. Chaudron, "Generalizing Consistency Checking between Software Views,"in Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture, Pittsburgh, Pennsylvania, USA, 2005.

[75] L. Xiaoshan, L. Zhiming, and J. He, "A formal semantics of UML sequence diagram,"in Proceedings of Australian Conference on Software Engineering, Melbourne, Australia, 2004.

[76] G. Spanoudakis, K. Kasis, and F. Dragazi, "Evidential diagnosis of inconsistencies in object-oriented designs," *International Journal of Software Engineering and Knowledge Engineering,* vol. 14, pp. 141-178, 2004.

[77] A. Reder and A. Egyed, "Computing repair trees for resolving inconsistencies in design models,"in Proceedings of 27th IEEE/ACM International

Conference on Automated Software Engineering, Essen, Germany, 2012.

[78] O. Vasilecas and R. Dubauskaite, "Ensuring consistency of information systems rules models,"in Proceedings of International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, Ruse, Bulgaria, 2009.

[79] O. Pilskalns, D. Williams, D. Aracic, and A. Andrews, "Security Consistency in UML Designs,"in Proceedings of 30th Annual International Computer Software and Applications Conference, Chicago, USA, 2006.

[80] C. F. J. Lange and M. R. V. Chaudron, "Effects of defects in UML models: an experimental investigation,"in Proceedings of 28th international conference on Software engineering, Shanghai, China, 2006.

APPENDIX B

In this appendix we present the main components of the protocol required to carry out an SMS **[12]** presented in [8]. In this research, the SMS [8] was used as starting point to build the consolidated set of 116 UML costistency rules. At the end of this appendix we also present the execution of the SMS.

1. Research Questions

The underlying motivation for the research questions was to determine the current state of the art as regards UML consistency rules, and this guided the design of the review process. In order to identify the current state of the art of UML consistency rules, we considered seven research questions (RQs):

TABLE 4
RESEARCH QUESTIONS

Research questions	**Main motivation**
RQ1: What are the UML versions used by researchers in the approaches found?	To discover what UML versions are used in the approaches that handle UML consistency.
RQ2: Which types of UML diagrams have been tackled in each approach found?	To discover the UML diagrams that research has focused upon, to reveal the UML diagrams that are considered more important than others, in addition to identifying opportunities for further research.
RQ3: What are the UML consistency rules to be checked?	To find the UML consistency rules to be checked and to assess the state of the field.
RQ4: Which types of consistency problems have been tackled in the rules found?	To find the types of consistency problems tackled in the rules. The data found are categorized into three consistency dimensions split into three sub-dimensions: 1) horizontal, vertical and evolution consistency; 2) syntactic and semantic consistency; 3) observation and invocation consistency.
RQ5: Which research type facets are used in research on UML model consistency?	To determine whether the field is generally more applied or more basic research, along with identifying opportunities for future research. The papers found were categorized into six types: evaluation research, validation research proposal of solution, philosophical papers, opinion papers and personal experience papers.
RQ6: Is the approach presented automatic, manual or semi-automatic?	To discover how the approaches used to check UML consistency are implemented, in other words whether their check system is presented in an automatic, manual or semi-automatic manner.
RQ7: How are the UML consistency rules specified? How are the UML consistency rules checked?	To discover how the consistency rules used to check the consistency of the UML diagrams are specified (e.g., Plain English, OCL, Promela) and to discover with which tools those consistency rules are checked (e.g., SPIN, OCL-Checker)

2. Search strategy

Conducting a search for primary studies requires the identification of search strings (SS), and the specification of the parts of primary studies (papers) in which the search strings are sought (the search fields). We identified our search strings by following the procedure of Brereton et al [39]:

1. Define the major terms;
2. Identify alternative spellings, synonyms or related terms for major terms;
3. Check the keywords in any relevant papers that are already available;
4. Use the Boolean OR to incorporate alternative spellings, synonyms or related terms;
5. Use the Boolean AND to link the major terms.

The major search terms were "UML" and "Consistency" and the alternative spellings, synonyms or terms related to the major terms are presented in the following table.

TABLE 5
SEARCH STRINGS

Major Terms	**Alternative terms**
UML	(uml OR unified modeling language OR unified modelling language)
Consistency	(consistency OR inconsistency)

In the selection of the SS, we considered various alternatives. For example the SS used in the SLR on consistency management [5] was discarded owing to the fact that it might not strictly focus on UML consistency rules: we are much more interested in collecting rules than in identifying consistency management issues and solutions. (This is the main reason why we obtained a different set of primary studies.) Other SSs were experimented with, but it is not possible to discuss all the alternative search strings below owing to space limitations. In the set of alternative SSs, we selected the following, as it allowed us to retrieve the largest number of useful papers, i.e., the largest number of papers focusing on UML consistency:

((uml OR unified modeling language OR unified modelling language) AND (consistency OR inconsistency))

We did not establish any restrictions on publication years up to December 12, 2012. The SMS process started in September 2012 and was completely finished on October 2013. We used the before mentioned SS with the following seven search engines: IEEE Digital Library, Science Direct, ACM Digital Library, Scopus, Springer Link, Google Scholar, and WILEY. The searches were limited to the following search fields: title, keywords and abstract.

3. Selection procedure and inclusion and exclusion criteria

In this section we discuss the inclusion and exclusion criteria used. We then discuss the process followed to include a primary study in this SMS. The inclusion criteria were:

- Electronic Papers (EPs) focusing on UML diagram consistency which contained at least one UML consistency rule;
- EPs written in English language;
- EPs published in peer-reviewed journals, international conferences and workshops;
- EPs published up to December 12, 2012.
- EPs which proposed UML consistency rules with a restriction (or extension) of the UML models that do not strictly follow the OMG standard [15].

The exclusion criteria were:

- EPs not focusing on UML diagram consistency;
- EPs which did not present a full-text paper (title, abstract, complete body of the article and references) but were reduced to an abstract, for instance;
- EPs focusing on UML diagram consistency which did not contain at least one UML consistency rule;
- Duplicated EPs (e.g., returned by different search engines);
- EPs which discussed consistency rules between UML diagrams and other non-UML sources of data, such as requirements or source code.

4. Data extraction strategy

We extracted the data from the primary studies according to a number of criteria, which were directly derived from the research questions detailed in TABLE 4. Using each criterion to extract the data required, we read the full text of each of the 95 primary studies. Once recorded, we collected the data on an Excel spreadsheet employed as our data form. The following information from each primary study was extracted and collected on the Excel data form:

- Search engines: where the paper was found (see section 2 of the Appendix B);
- Inclusion and Exclusion Criteria;
- Data related to Research Questions:
  - What UML version was used;
  - What are the UML consistency rules discussed (see Appendix B);
  - What diagrams are involved in consistency rules (see section 3.3 of this paper);
  - UML consistency dimensions: UML diagram consistency is discussed in the literature according to several dimensions [40]:
    - Horizontal, Vertical and Evolution Consistency: Horizontal consistency, also called intra-model consistency, refers to consistency between different diagrams at the same level of abstraction (e.g., class and sequence diagrams during analysis) in a given version of a model [41]. Vertical Inconsistency, also called inter-model consistency, refers to consistency between diagrams at different levels of abstraction (e.g., analysis vs. design) in a given version of a model [42]. Evolution consistency refers to consistency between diagrams of different versions of a model in the process of evolution [41].
    - Syntactic versus Semantic consistency: Syntactic consistency ensures that a specification conforms to the abstract syntax specified by the meta-model, and requires that the overall model has to be well formed [42]. Semantic consistency requires diagrams to be semantically compatible [42]. This does not mean that semantic consistency is necessarily restricted to behavioral diagrams. For instance, operation contracts (e.g., pre and post conditions) provided in the class diagram specify behavior. Semantic consistency applies at one level of abstraction (with horizontal consistency), at different levels of abstraction (vertical consistency), and during model evolution (evolution consistency) [14].
    - Observation versus Invocation consistency: Observation consistency requires an instance of a subclass to behave like an instance of its superclass, when viewed according to the superclass description [43] . In terms of UML state machine diagrams (corresponding to protocol state machines) this can be rephrased as "after hiding all new events, each sequence of the subclass

state machine diagram should be contained in the set of sequences of the superclass state machine diagram." Invocation consistency requires an instance of a subclass of a parent class to be used wherever an instance of the parent is required [44]. In terms of UML state machine diagrams (corresponding to protocol state machines), each sequence of transitions of the superclass state machine diagram should be contained in the set of sequences of transitions of the state machine diagram for the subclass.

    - Tool support (Automatic, Semi-Automatic, Manual);
- Automatic means that a tool automatically checks the UML consistency rules with no human intervention;
    - Semi-automatic means that the checking of the UML consistency rules was partially automated (for instance when the checking of a UML model needs a user's supportfor the process to be completed);
    - Manual means that the UML consistency rules were not supported by any implemented and automatic tool.
- What mechanisms were used to specify the rules: e.g., plain language, Promela, etc.;
- How are the UML consistency rules checked: e.g., SPIN, OCL-Checker, etc.;
- Research type facet followed in the paper, for which we used the following classification [45]:
    - Evaluation research (ER): this is a paper that investigates techniques that are implemented in practice and an evaluation of the technique is conducted. This means that the paper shows how the technique is implemented in practice (solution implementation) and what the consequences of the implementation are in terms of benefits and drawbacks (implementation evaluation).
    - Proposal of solution (PS): this is a paper that proposes a solution to a problem and argues for its relevance, without a full-blown validation.
    - Validation Research (VR): this is a paper that investigates the properties of a solution that has not yet been implemented in practice.
    - Philosophical papers (PP): this is a paper that sketches a new way of looking at things, a new conceptual framework, etc.
    - Opinion paper (OP): this is a paper that contains the author's opinion about what is wrong or good about something, how something should be done, etc.
    - Personal experience paper (PEP): this is a paper that places more emphasis on what and not on why.

5. Execution

The planning for this SMS with the seven search engines begun in September 2012 and was completed on December 12, 2012. In this section we present the execution of the SS in the seven search engines and the selection of primary studies according to the inclusion/exclusion criteria previously described. In order to document the review process with sufficient details [12], we describe the multi-phase process of the four sub-phases we followed:

- First sub-phase (SP1): the search string was used to search the seven search engines, as mentioned earlier.
- Second sub-phase (SP2): we deleted duplicates automatically, by using the RefWorks tool [46]; we also removed duplicates manually.
- Third sub-phase (SP3): we obtained an initial set of studies by reading the title, abstract and keywords of all the papers obtained after SP2 while enforcing the inclusion and exclusion criteria. When reading just the title, abstract and keywords of a paper was not sufficient to decide whether to include or exclude it, we checked the full-text.
- Fourth sub-phase (SP4): all the papers identified in SP3 were read in their entirety and the exclusion criteria were applied again. This resulted in the final set of primary studies.

TABLE 6 breaks down the number of papers we have found by sub-phases. Row SP1 in TABLE 6 shows the first results which were obtained by running the SS in the seven search engines selected. The next two rows show the results obtained after applying SP2 and SP3 of the study selection process. We eventually collected 95 primary studies for further analysis. The complete list of references can be found elsewhere [47].

TABLE 6
SUMMARY OF PRIMARY STUDIES SELECTION

Sub phase	IEEE	Scopus	Springer Link	Google Scholar	WILEY	ACM	Science Direct	Total
SP1: Raw results	363	601	163	341	9	87	39	1603
SP2: No duplicates	279	325	159	247	9	80	36	1135
SP3: First selection	62	64	62	28	4	33	14	267
SP4: Primary studies	16	21	21	12	1	16	8	**95**