Systematic Review of Software Behavioral Model Consistency Checking

FAIZ UL MURAM, HUY TRAN, AND UWE ZDUN, University of Vienna, Faculty of Computer Science, Software Architecture Research Group, Vienna, Austria

In software development, models are often used to represent multiple views of the same system. Such models need to be properly related to each other in order to provide a consistent description of the developed system. Models may contain contradictory system specifications, for instance, when they evolve independently. Therefore, it is very crucial to ensure that models conform to each other. In this context, we focus on consistency checking of behavior models. Several techniques and approaches have been proposed in the existing literature to support behavioral model consistency checking. This paper presents a Systematic Literature Review (SLR) that was carried out to obtain an overview of the various consistency concepts, problems, and solutions proposed regarding behavior models. In our study, the identification and selection of the primary studies was based on a well-planned search strategy. The search process identified a total of 1770 studies, out of which 96 have been thoroughly analyzed according to our predefined SLR protocol. The SLR aims to highlight the state-of-the-art of software behavior model consistency checking and identify potential gaps for future research. Based on research topics in selected studies, we have identified seven main categories: targeted software models, types of consistency checking, consistency checking techniques, inconsistency handling, type of study and evaluation, automation support, and practical impact. The findings of the systematic review also reveal suggestions for future research, such as improving the quality of study design and conducting evaluations, and application of research outcomes in industrial settings. For this purpose, appropriate strategy for inconsistency handling, better tool support for consistency checking and/or development tool integration should be considered in future studies.

CCS Concepts: • **General and reference** →**Surveys and overviews;** *Design; Verification; Validation;* • **Software and its engineering** →**Consistency; Formal methods;**

Additional Key Words and Phrases: Software Behavioral Model, Consistency Checking, Consistency Types, Systematic Literature Review

ACM Reference format:
Faiz UL Muram, Huy Tran, and Uwe Zdun. 2017. Systematic Review of Software Behavioral Model Consistency Checking. 1, 1, Article 1 (April 2017), 44 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

The research is supported by the Wiener Wissenschafts-, Forschungs- und Technologiefonds (WWTF), Grant No. ICT12-001.
Author's addresses: F. UL Muram (faiz.ulmuram@univie.ac.at), H. Tran (huy.tran@univie.ac.at) and U. Zdun (uwe.zdun@univie.ac.at), University of Vienna, Faculty of Computer Science, Software Architecture Research Group, Vienna, Austria.

1 INTRODUCTION

The development of a software system often goes through a number of different stages and iteration cycles, and each of them can introduce new elements or more detailed specifications of the system [36]. In addition, software systems are constantly evolving. In many areas of software engineering, behavioral models are used to represent the behavioral aspects of a software system. Examples of behavior models are UML activity models, state machines, and sequence models [35], Simulink® Stateflow® [30], the Business Process Model and Notation (BPMN) [34], the Business Process Execution Language (BPEL) [33], and Event-driven Process Chains (EPC) [37], to name but a few. In the past decades, a substantial number of software engineering research works have been devoted to consistency checking between different models or multiple views of the models that are used in software development (for a comprehensive list of existing works, please refer to Usman et al. [41] and Lucas et al. [28]).

An example is ensuring consistency over multiple abstraction levels: Many models are created as "high-level" models [21, 40, 43]. That is, they are mainly used to convey the core concepts or principles of the reality they represent in an abstract and/or concise way (e.g., requirements models or design models). In addition, technical or "low-level" models are often created as refinements of the high-level models with purposes such as providing a precise specification of the source code, executing the model (e.g., in a process engine, interpreter, or virtual machine), or generating executable code directly from the model, e.g., in model-driven software development (MDSD) [14, 40]. It is crucial that the overlapping parts of the high-level and low-level models are in sync with each other [21, 39, 43]. Unfortunately, multiple models of the same system (or reality) are often drifting apart over time, and inconsistencies arise among them, when they are created by different stakeholders and evolved independently [21, 39, 43].

For instance, high-level models might be changed according to new requirements, and low-level models are changed as the implementation is modified. If not each change is systematically propagated to all other models of the same system (or reality), the evolved models may include inconsistencies. Inconsistencies can also occur due to the multi-view nature of many models [13, 21]. A system can be defined by multiple views that specify different aspects of the system. Inconsistencies can occur due to the overlaps between the views [21]. Consistency is a general goal to be obtained while building the models. In particular, during system (and model) evolution it is crucial to maintain the consistency between different behavior models. As Spanoudakis and Zisman [39] also emphasized, there are severe negative effects of model inconsistencies that may delay and, therefore, increase the cost of the system development process, jeopardize properties related to the quality of the system, and make it more difficult to maintain the system [39]. These negative effects can be multiplied manifoldly, especially in the context of developing modern large scale software systems that are far more complex and might consist of numerous models and interconnected subsystems (e.g., cloud-based systems, Internet of Things systems, networks of sensors based systems, or cyberphysical systems) [2, 18, 38].

There are only a few secondary studies in collecting and analyzing evidence regarding the research of model consistency checking and management [21, 28, 39, 41]. The survey of Spanoudakis and Zisman [39] presents a broad view and discusses open research issues. Among of those secondary studies, only the SLR conducted by Lucas et al. [28] has systematically investigated and provided insights regarding consistency concepts, proposals, problems, supported models and the maintainability of consistency management approaches, but only for UML models in the timespan from 2001 to 2007.

We note that the aforementioned studies, except the survey of Spanoudakis and Zisman [39], mainly focus on UML based modeling and development. They aim to cover broadly both structural and behavioral consistency checking of software artifacts. Although UML is widely used in academia and industry, there are still a considerable number of non-UML modeling and development methods, for instance, in the domains of workflow and business process management, embedded and real-time systems, and service-oriented systems, to name but a few. Considering only UML-based methods and techniques can likely lead to bias against non-UML approaches. Thus, we decided to conduct a systematic literature review of consistency checking with a broader scope and, therefore, research objectives. Our study strives for investigating consistency checking beyond the domain of UML-based software development. As checking of structural aspects has been extensively reviewed [21, 28, 39, 41], we opted to pay special attention to the behavioral aspects of software system modeling. We planned to carry out the SLR at a finer granularity, for instance, we studied in-depth the degrees of automation support, inconsistency handling, tool support, evaluation, and evidence of application in industry settings (these aspects are not yet considered or only in limited form in the previous studies). Chen and Ali Babar [7] emphasized that these aspects are important as they indicate the practical impacts of the research outcomes to both science and industrial practices. Nevertheless, we adopted and extended some fundamental concepts and categories of consistency checking and management that have been proposed in these studies.

The paper is organized as follows: Section 2 describes background and context of the consistency checking problem along with fundamental concepts. Section 3 explains the systematic literature review process used in this review. Section 4 discusses the results of SLR in relation to the addressed research questions. Section 5 presents the discussion on the results of our SLR. Finally, Section 6 concludes the paper with main findings and potential future research directions.

2 BACKGROUND ON CONSISTENCY CHECKING OF SOFTWARE MODELS

According to [39], an inconsistency is described as "*a state in which two or more overlapping elements of different software models make assertions about aspects of the system they describe which are not jointly satisfiable*". As [39] stated, the problem of inconsistencies in software models have been a big concern of the software engineering community for a long time. As a system is often modeled from different viewpoints by different stakeholders, in different levels of abstraction and granularity [4, 12], there may exist contradicting information [10, 39]. There are several definitions of consistency and its classification emerging in the literature [10, 21, 28, 41, 43].

A typical consistency problem happens among different types of representations (e.g., models) of the same aspect of a software system. For instance, for describing the interactions among various objects, a UML sequence diagram and/or a communication diagram can be used with respect to a temporal or structural viewpoint, respectively [10]. Engels et al. [10] consider this type of consistency "*horizontal consistency*" whilst Huzar et al. [21] and Usman et al. [41] call it "*intra-model consistency*".

Looking into another dimension, a software model can be refined or transformed into a richer form with more details. This scenario happens quite often in model-driven software development in which model transformations are extensively used to map a high-level, abstract model down to a lower level of abstraction [14, 40]. This type of consistency is named "*vertical consistency*" [10], "*inter-model consistency*" [21], or "*refinement*" [28]. We note that the term "model" is interpreted rather differently by Huzar et al. [21] (as a set of diagrams) and Engels et al. [10] (as an umbrella term that embraces the meaning of a diagram) and, therefore, their definition of consistency types are not totally overlapping. [43] used the term "*evolution consistency*" to indicate the consistency between different versions of the same model, which fits to the category "vertical consistency" in Engels et al. [10].

In the context of this study, we adopt the umbrella term "*model*" as proposed by [10, 39] and use the terms "model" and "diagram" interchangeably with the same meaning unless stated otherwise in case it is necessary to distinguish between these terms. As a result, the interpretation of *horizontal consistency* and *vertical consistency* based on this point of view is used in our SLR.

In addition, our initial study revealed that several approaches investigate the consistency of a single model itself for correctness, determinism, and so forth. These consistencies do not fit nicely into the two categories mentioned above. Inspired by the philosophical stance presented by Mens and Van Gorp [31] in categorizing model transformations, we consider the following definitions of consistency types for software models in this SLR.

- *Endogenous consistency* indicates consistencies within the same model regarding the properties of itself.
- *Exogenous consistency* denotes consistencies between different models. It can be refined into two sub-categories: horizontal and vertical consistencies.
    - *Horizontal consistency* denotes consistencies among different kinds of models or a model against rules and constraints that are manually specified or derived from other artifacts.
    - *Vertical consistency* shows the consistency among models of the same type at different levels of abstraction. This also includes the consistency between a design model and a corresponding implementation.

Existing studies distinguish between syntactic and semantic consistencies [10]. *Syntactical consistency* aims to ensure that a model conforms to a predefined syntax specified by a certain meta-model or grammar [10]. This ensures the well-formedness of the model [10]. Syntactic (or structural) consistency checking has been extensively discussed in [21, 28, 41].

As the main focus of our study are behavioral models, we will investigate further into semantic consistency. *Semantic consistency* addresses the semantic compatibility of models, for instance, the same identifier that occurs in different models should refer to the same entity (i.e., have same meaning). Semantic consistency is stricter and often requires syntactical consistency beforehand. In the context of horizontal consistency, semantic requires models of different viewpoints to be semantically compatible with regards to the aspects of the system which are specified in both sub-models. For vertical consistency problems, semantic consistency requires that a refined model is semantically consistent with the model it refines. Semantic consistency depends very much on the underlying semantics of the models being used and of the development process. Eshuis and Grefen [11] also observed that several model structures are described by different syntaxes but represent the same behavior. Therefore, the identification of a suitable *semantic domain* is important for consistency checking of software models in general and behavior models in particular [10].

3 SYSTEMATIC LITERATURE REVIEW PROCESS

An SLR provides a key instrument for identifying, evaluating, and interpreting all existing research related to a particular research question, phenomenon of interest or topic area [25, 26]. SLRs are a form of secondary study, while individual studies that contribute to the SLR are considered primary studies. The process of conducting an SLR must be explicitly defined and well executed. To facilitate the planning and execution of this SLR, we leverage the guidelines performing SLRs in software engineering recommended in [26] with adjustments recommended in [24, 25]. A clear description of the phases for conducting the review process will be presented in this section.

(1) *Planning the review*: The goal of this phase is to define the research questions and methods for developing a review protocol (see Section 3.1),
(2) *Conducting the review*: In this phase, the review protocol defined in the previous phase will be enacted (see Section 3.2),
(3) *Reporting the review*: In this phase, the results of the review are documented, validated, and reported (see Sections 4 and 5).

The aforementioned phases are not done sequentially at once but rather in an iterative manner with feedback loops. In particular, many activities are created during the protocol development phase and should be refined when execution of the review takes place. For example, search terms, inclusion criteria, and exclusion criteria can be refined during the course of the review. The protocol provides details of the plan for the review, such as specifying the search process to be followed and the conditions to apply when selecting relevant primary studies.

3.1 Planning the Review

The first phase for undertaking an SLR is related to specifying pre-review activities for conducting the SLR. The planning phase includes identification of the reasons for carrying out the systematic review (see Section 3.1.1), specifying the research questions (see Section 3.1.2), defining the search strategy (see Section 3.1.3), and establishing the inclusion and exclusion criteria (see Section 3.1.4).

3.1.1 Need for the Literature Review. There are several reasons for undertaking this systematic review. The main objective of this SLR is to systematically select and review the published literature with regard to behavioral model consistency and summarize all existing practices and information in a well-defined and unbiased manner including problems, limitation, future trends, and possible opportunities within the context of software behavior model consistency.

This SLR aims at identifying the current state-of-the-art of consistency checking of software behavior models not only in the field of UML based modeling but also in various other domains. As mentioned above, the existing reviews merely covered UML models. Hence, no comprehensive systematic review in the area of software behavior models consistency checking has been previously published. Our study also aims at a finer level of granularity with regard to automation support, inconsistency handling, tool support, evaluation, and evidence of application in industry settings (which have not yet been well considered in the previous studies) as these aspects indicate the practical impacts of the research outcomes to both science and industrial practices [7]. The perspective taken in this study is both of practitioners and researchers working on behavioral model consistency checking to provide them up-to-date state-of-the-art research and therefore guide them to identify relevant studies that suit their own needs. Furthermore, we aim to appraise evidence of research on consistency checking of software behavior models as well as to identify challenges and open problems that may provide insights for further investigations.

3.1.2 Research Questions. Defining the research questions is one of the most important activities of any systematic review because they guide the selection of primary studies and data extraction. For this purpose, we leverage the Goal-Question-Metric (GQM) approach [3], which is a systematic method for organizing measurement programs. The GQM model starts with specifying the certain goal (i.e., purpose, object, issue, and viewpoint). Then, the goal is refined into several questions, each question is then refined into metrics [3]. By using GQM, a goal for conducting the SLR is defined. The goal is refined into several research questions, and, subsequently, these questions are refined into metrics that provide means to answer these questions. By providing the answers of the questions, the data can be analyzed to identify whether the goals are achieved or not. The goal for our SLR is:

Purpose Understand and characterize ...
Issue ... the consistency checking ...
Object ... of behavioral models used in software development ...
Viewpoint ... from a researcher's and engineer's viewpoint.

Based on the aforementioned goal, we derive the following research questions.

- **RQ1**: How did research in consistency checking of software behavioral models develop over time?
- **RQ2**: What are the methods, languages, or techniques used in each of the primary studies? This research question is refined into following research questions:
  - **RQ2.1**: What types of models have been studied?
  - **RQ2.2**: What kinds of consistency problems have been addressed?
  - **RQ2.3**: What consistency checking techniques have been used?
  - **RQ2.4**: What inconsistency handling techniques have been proposed?
  - **RQ2.5**: What levels of automation have been supported?
  - **RQ2.6**: What types of study and evaluation have been conducted?
- **RQ3**: What is the potential practical impact of the primary studies?
- **RQ4**: What are the limitations of the existing methods?

RQ1 and RQ2 aim at describing the state-of-the-art of research on consistency checking of software behavioral models. The expected outcome will be a comprehensive view of behavioral model consistency checking in various dimensions. This allows us to categorize the state-of-the-art in consistency checking with respect to the behavioral models. RQ3 is proposed for learning about the practical impact of the existing methods in academia and industry. It aims to provide both researchers and practitioners with evidence about what methods could be used in practice and to which degree. A lack of evidence or poor evidence could highlight the need for more rigorous studies and applications in real or quasi-real settings. RQ4 is formulated to identify gaps in current research that could yield insights regarding the issues or open problems in software behavioral models consistency research and provide directions for further studies.

3.1.3 Search Strategy. Our search strategy aims to find a comprehensive and unbiased collection of primary studies from the literature related to the research questions. Therefore, we devised a search strategy that maximizes the possibility to discover every relevant publications in a search result. For the electronic search, we leveraged the major databases that are widely used in computer science (CS) and software engineering (SE) research as reported in [45], which are the ACM Digital Library[1], the IEEE Explore Digital Library[2],

[1]See http://dl.acm.org
[2]See http://ieeexplore.ieee.org

SpringerLink[3], and ScienceDirect[4]. These are rich and comprehensive databases containing bibliographic information of a plethora of publications from all major publishers of the computing literature. We note that ISI Web of Science (WoS)[5] is also a large database of scientific studies. However, in the field of software engineering, WoS mainly records publications published in premium journals and only a few conferences, whereas conferences (and sometimes workshops and symposiums) are major outlets for publishing in CS and SE. Thus, we mainly use WoS along with Google Scholar[6] for cross-checking with the search results from the chosen sources and for performing some meta-analyses. Our search terms stem from the research questions and can be categorized into two major dimensions, as shown in Table 1. On the one hand, we aim at exploring the different behavior models or diagrams considered in the primary studies. On the other hand, we aim to discover relevant types of studies that have been performed for checking different types of inconsistencies. In order to ensure a sufficient scope of searching, we also consider different alternative words (e.g., behavior/behaviour/behavioural, model/diagram), synonyms (e.g., model, diagram, workflow, process), and abbreviations for the search terms. Then we used the boolean operator "OR" to join the alternate words and synonyms and the boolean operator "AND" to form a sufficient search string.

Table 1. Literature Search Dimensions and Keywords

Dimension	**Search Keywords**
Type of models	behaviour diagram, behavior diagram, behavioural diagram, behavioral diagram, behaviour model, behavior model, behavioural model, behavioral model, activity diagram, sequence diagram, state diagram, state machine, statechart, collaboration diagram, communication diagram, interaction diagram, timing diagram, workflow, business process, process model, BPMN, WSBPEL, EPC, finite state machine, FSM, state-transition, Stateflow
Type of studies	containment/contain/containing, refine/refining/refinement, inconsistencies/inconsistent/inconsistency, consistent/consistency/consistencies

3.1.4 Inclusion and Exclusion Criteria. Inclusion and exclusion criteria are used to identify the suitability of primary studies and making decisions for inclusion or exclusion of an article in the SLR based on the addressed research questions. Our inclusion and exclusion criteria are shown in Table 2.

3.2 Conducting the Review

The second phase of the SLR is to perform the search strategy defined in the planning stage. The steps involved in conducting the review are primary studies selection presented in Section 3.2.1, quality assessment presented in Section 3.2.2, and data extraction and synthesis described in Section 3.2.3.

[3]See http://link.springer.com
[4]See http://www.sciencedirect.com
[5]See http://webofknowledge.com
[6]See http://scholar.google.com

Table 2. Inclusion and Exclusion Criteria

Type	Description
Inclusion	**I1** Study is internal to software domain. We are only interested in consistency checking for software systems.
	I2 Study is about consistency checking related to software behavioral models/diagrams.
	I3 Study comes from an acceptable source such as a peer-reviewed scientific journal, conference, symposium, or workshop.
	I4 Study reports issues, problems, or any type of experience concerning software behavioral model consistency.
	I5 Study describes solid evidence on software behavioral model consistency checking, for instance, by using rigorous analysis, experiments, case studies, experience reports, field studies, and simulation.
Exclusion	**E1** Study is about hardware or other fields not directly related to software.
	E2 Study is not clearly related to at least one aspect of the specified research questions.
	E3 Study reports only syntactic or structural consistency checking of models/diagrams.
	E4 Secondary literature reviews.
	E5 Study does not present sufficient technical details of consistency checking related to software behavioral models (e.g., they have a different focus (i.e., version control) and have insufficient detail)
	E6 Study did not undergo a peer-review process, such as non-reviewed journal, magazine, or conference papers, master theses, books and doctoral dissertations (in order to ensure a minimum level of quality).
	E7 Study is not in English.
	E8 Study is a shorter version of another study which appeared in a different source (the longer version will be included).

3.2.1 Primary Studies Selection. We illustrate the search process and the number of primary studies identified at each stage in Figure 1. We strive for a comprehensive list of studies reported without any additional constraints. That is, our search strategy dated back to late eighties since that time period is often considered to foster consistency checking of software artifacts [39]. We started with the search in June 2015 and ended the search process at the end of July 2015 using the search string described in Section 3.1.3. The initial search, however, is performed in 2013. After consolidating the results, overall 1770 studies have been identified. In the initial search process, we identified 1698 studies; whereas the snowballing process added 72 more studies.

Two researchers working independently have identified the relevant studies by quickly scanning parts of each publication such as the title, abstract and keywords (and sometimes the conclusion in case it was difficult to extract information from the abstract). The second selection stage is based on the aforementioned inclusion and exclusion criteria. We note that there were a number of duplications due to different reasons. For instance, some authors published their journal articles which were extended versions of previously published

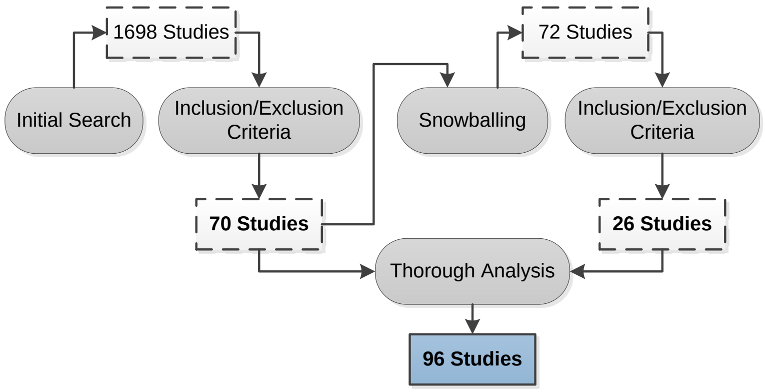

Fig. 1. Overview of the stages and results of our search process

workshop and/or conference papers. Therefore, we worked carefully to eliminate all potential duplications and retained only the most complete (or recent) versions of the duplicates. After the inclusion and exclusion stage (including removing duplications), there are 70 primary studies remained out of 1698 identified studies in initial search.

We are aware that it is impossible to achieve a total set of publications using the aforementioned automated searches. Therefore, we performed an additional *snowballing* process [6] (i.e., manually scanning and analyzing the references and citations of these primary studies) to ensure that our study also covered relevant follow-up works that might exist but have not been included in the search. In particular, we collected references and related works from each of 70 primary studies selected in the automated search phase. The snowballing search continued until no more relevant studies were found. As a result, there are total 72 studies collected from the snowballing process.

Some of these studies did not appear in the results of the aforementioned automated searches. Some others existed but were not found in the first search because either they have too short abstracts or their titles or abstracts do not explicitly include the proposed search terms (but their contents and contribution do). Twenty-six out of the 72 additional studies satisfied the inclusion criteria and exclusion criteria, and therefore, are selected. Finally, we analyzed the remaining primary studies thoroughly to ensure that we had obtained the most relevant studies. This in-depth analysis results in total 96 studies within the time period of 1999–2015 that are included for further consideration in our study.

3.2.2 Quality Assessment. The quality assessment criteria are used to determine the rigorousness and credibility of the used research methods and the relevance of the studies. This assessment is important to limit bias in conducting this SLR, to obtain insight into potential differences, and to support the interpretation of the results. Three main quality assessment criteria have been applied that are based on the assessment criteria introduced in [26, 27]. We used a checklist based scoring procedure to evaluate the quality of each selected study and to provide a quantitative comparison between them. The scoring procedure has only three optional answers: “Yes = 1”, “Partly = 0.5”, or “No = 0”. Therefore, for a given study, its quality score is computed by summing up the scores of the answers to the quality assessment questions.

– *Relevance.* Are all the collected studies relevant to the objective of this literature review? In this SLR, the quality assessment is specifically focused on accumulating only those studies that report adequate information to answer the targeted research questions. The quality assessment has been performed according to our inclusion and exclusion criteria by the first two researchers who have independently reviewed each study. In case of contradicting opinions, the third researcher reviewed and resolved the issues together with the two researchers.
– *Coverage.* Is the literature research cover all relevant studies? In this SLR, the two researchers have independently reviewed each study in a number of iterations to ensure that none of the relevant studies are missed. To achieve this, the search is performed on the entire list of relevant studies following with the screening of the titles, abstracts, keywords, and conclusion. Moreover, a snowballing process is conducted to broaden the scope of selected studies. Finally, in-depth analyses have been performed after accessing the full text.
– *Validation.* Do the collected studies contain adequate data and information? In this SLR, it is analyzed whether the primary studies contain the necessary information to answer the targeted research questions. In particular, we devised a number of questions to assess the validation of the relevant studies, such as: Is the technique/tool clearly defined? How rigorously is the technique evaluated? Does the study add value to academia or industry?

Table 3. Data Collection

Extracted data	**Relevant RQ**
Author(s)	Study overview
Title	Study overview
Year	Study overview, RQ1
Studied domains	Study overview
Publication types and venues (c.f. Section 4.1)	Study overview, RQ1
Active research groups (c.f. Section 4.1)	Study overview, RQ1
Types of studied behavioral models/diagrams (c.f. Section 4.2)	RQ2.1
Consistency checking types (c.f. Section 4.3)	RQ2.2, RQ4
Consistency checking techniques (c.f. Section 4.4)	RQ2.3, RQ4
Formalization methods (c.f. Section 4.4)	RQ2.3, RQ4
Degree of inconsistency handling (c.f. Section 4.5)	RQ2.4, RQ4
Degree of automation support (c.f. Section 4.6)	RQ2.5, RQ4
Tool support for consistency checking (c.f. Section 4.6)	RQ2.5, RQ3, RQ4
Types of study and evaluation (c.f. Section 4.7)	RQ2.6, RQ4
Citations (c.f. Section 4.8)	RQ3
Development tool support and integration (c.f. Section 4.8)	RQ3, RQ4
Levels of application evidence (c.f. Section 4.8)	RQ3, RQ4

3.2.3 Data Extraction and Data Synthesis. All remaining 96 primary studies were analyzed in-depth and relevant data were extracted from these studies. In the data extraction step, we used spreadsheets to record and correlate the extracted information. In this SLR, we concentrate on extracting the following data items from each study. Table 3 shows the corresponding data extracted from the set of selected studies. After the data extraction stage was completed, we synthesized the resulting information such that they are suitable and sufficient for answering the review questions. We will explain the rationale of extracting data and analyze the extracted data in Section 4. For further details on the classification and encoding of every aspect related to extract data, please refer to Appendix C.

4 RESULTS

In this section, we summarize the main results obtained in the systematic review together with an analysis of the collected data in order to determine the current research trends and identify existing gaps and open problems of the current methods. Table 16 in Appendix A shows the artifacts studied by each approach, type of semantic domain and consistency checking technique supported by existing literature, and the studied domains. For a list of these selected studies with full details, please refer to Table 17 in Appendix B.

4.1 Historical Development (RQ1)

This section presents the result about the research, publication trend (i.e., time and venue) and active research groups in the context of behavioral models consistency checking. Figure 2 shows an overview of the distribution of selected studies per year grouped by publication types (e.g., journal, conference, workshop, symposium, etc.). We can see that consistency checking of software behavior models has drawn considerable attention and became active at the beginning of the 2000s. The number of studies reaches a peak around 2006-2008. Apart from that, the trend of research in behavioral models consistency checking seems stable over time. We did not set a lower boundary for the year of publication in our search process, yet the timeframe of identified studies reflects also the timeframe of activeness and maturation of behavioral models consistency checking field.

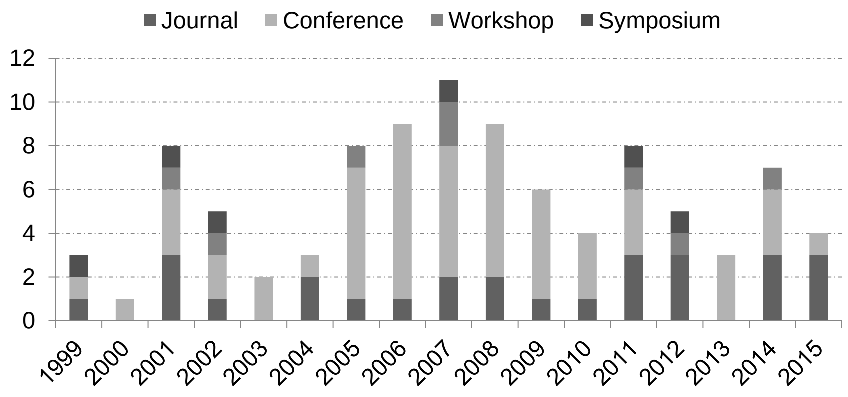

Fig. 2. **Primary Studies per Year Grouped by Publishing Venues**

With regard to the active research groups within the area of behavioral models consistency checking, we looked at the affiliation details of the selected primary studies. The assignment of contributed studies of each active research group is based on the affiliations that is given in these studies. Table 4 presents the active research groups (with at least three publications within behavioral models consistency checking) along with the corresponding number of contributed studies. The results depict that the University of Paderborn, Germany and Nanjing University, China are the leading ones in terms of the number of publications.

Table 4. Active Research Groups and Numbers of Studies

Affiliations	**Studies**	**Total**
University of Paderborn, Germany	S16, S17, S26, S29, S30, S39, S55	7
Nanjing University, China	S20, S31, S32, S47, S73, S86	6
University of Potsdam, Germany	S72, S81, S82, S83, S84	5
Universität München, Germany	S5, S37, S64, S92	4
Eindhoven University of Technology, The Netherlands	S19, S77, S78, S82	4
University of Toronto, Canada	S56, S62, S63	3
Lingnan University, Hong Kong	S88, S89, S90	3

4.2 Targeted Software Models (RQ2.1)

This subsection discusses different types of software models tackled in the current literature regarding behavior model consistency. Table 16 in the Appendix A shows the results of our SLR regarding targeted software models (i.e., the behavioral models being investigated for consistency checking). We depict in Figure 3 the distribution of different behavioral models considered in the selected studies. The UML specification 1.x uses the term "*statecharts*" whilst the UML specification 2.x switches to "*state machines*" [35, Sec. 15] instead. Please note that, UML statecharts (or UML state machines) are based on or derived from Harel's statecharts, which extends the classic notion of finite state machines with additional support for hierarchy, concurrency and communication [20]. For this reason, we use the term "statecharts" in this study to denote the aforementioned variants of Harel's statecharts and "state machine" to explicitly denote the classic definition of finite state machines. Some other types of behavioral models, such as live sequence charts (LSCs) and Simulink Stateflow are also various extensions of Harel's statecharts but target different modeling purposes and application domains. Thus, we opted to separate these types of models as shown in Figure 3. We note that "*process model*" is an umbrella term for BPMN, BPEL, and workflow models that are used for describing the behavior of process-centric information systems (PAIS). The category "Other" indicates a mix of various types of other behavioral models that do not belong to any of the aforementioned categories.

We can see that research on behavioral models consistency checking focuses on four major types of behavior models, which are statechart (36.59%), sequence diagrams (21.14%), process model (14.63%), and activity diagrams (9.96%). Less attention (i.e., 1.63%) has been paid on collaboration diagrams, LSCs, stateflow and labeled transition systems.

4.3 Types of Consistency Checking (RQ2.2)

Another aspect to be considered in our study is the types of consistency checking tackled by the selected publications. Static consistency checking techniques examine and analyze the targeted behavioral models and/or their abstracted versions without running systems. Examples are model checking techniques that systematically and exhaustively explore the states of software systems. The advantage of static checking is that it can be performed in early phases of software modeling and development where no executable products are produced yet. However, static checking can be computationally expensive due to the cost of exhaustive analysis of large and complex models. For instance, model checking often suffers from the problem of state explosion [8].

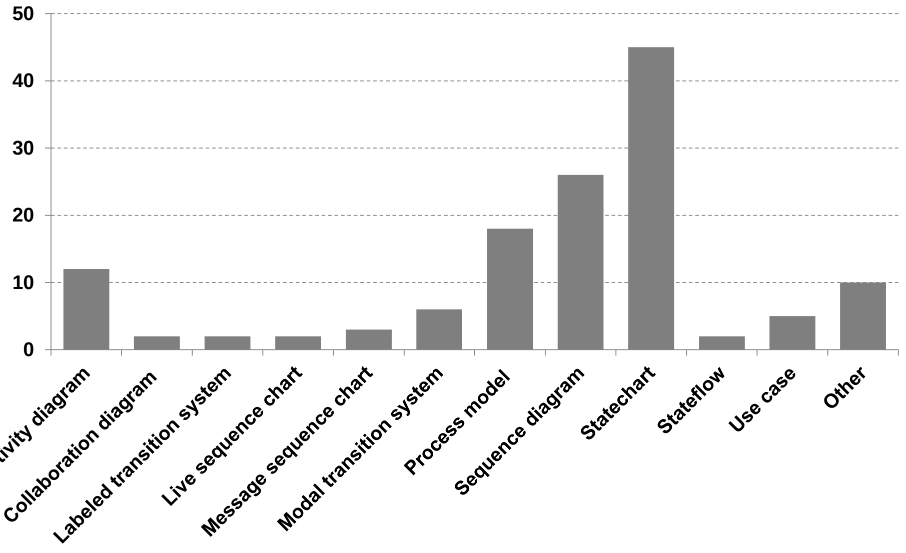

Fig. 3. Types of Studied Software Behavioral Models

In contrast, dynamic consistency checking aims to reveal inconsistencies while the system is running, either by code instrumentation or monitoring. Dynamic checking can achieve better computational performance as it only examines a small subset of system spaces (i.e., the actual execution traces). However, dynamic checking techniques, on the one hand, require a running system. On the other hand, in contrast to static checking, these techniques can not reveal all potential errors.

Apart from static and dynamic consistency checking, there are studies using symbolic execution or simulation approaches to validate the consistency of models. These approaches often consider a subset of input parameters and verify the response and consistency of the systems. As such, these studies do not exactly belong to either of the aforementioned categories. Therefore, we placed them in an additional category, namely, "*symbolic execution/simulation*" (SYM/SIM) to refer to these kinds of studies.

Table 5 depicts the distribution of consistency checking types in terms of static, dynamic, and symbolic execution/simulation checking. "Static consistency checking" are addressed prominently in 94 studies (93.07%) whilst "symbolic execution/simulation" is used in six studies (5.94%). Only one study (0.99%) covers dynamic consistency checking. Note that the sum of the numbers of studies on consistency checking types exceeds the total number of studies within a specific category, because the same study could address more than one type of consistency. Four studies (4.17%) out of a total of 96 use a combination of static+simulation consistency checking, whereas only one study (1.04%) uses a combination of static+dynamic consistency checking.

Figure 4 presents a bubble-plot distributed over two dimensions regarding: year of publication and consistency checking types. The results show that the majority of research attention has been paid to the static consistency checking throughout the years. Only one study has been found on dynamic consistency checking in the year 2009; however, the

Table 5. Types of Consistency Checking

Types	Studies	Total
Static	S1–51, S53–S69, S71–S96	94
Dynamic	S65	1
SYM/SIM	S48, S50, S52, S70, S17, S96	6
Endogenous	S15, S25, S37, S42, S44, S47, S55, S56, S61, S67–S69, S71, S74, S76, S79	16
Horizontal (Exogenous)	S1–S3, S6–S10, S13–S20, S22–S28, S30, S32–S36, S38, S41, S42, S44–S48, S50, S54, S56–S60, S62, S63, S65–S68, S70, S73, S77, S78, S84–S88, S90–S96	66
Vertical (Exogenous)	S4, S5, S11, S12, S15, S21, S29, S31, S39, S40, S43, S49, S51–S53, S56, S58, S64, S71, S72, S74, S75, S80–S83, S89	27

SYM/SIM consistency checking has received more attention in the timeframe 2010-2013 but is still at a rather low level.

Based on the aforementioned statistics, we can see that the research on dynamic and simulation consistency checking is much less than the research on static consistency checking. This is perhaps in accordance with Giannakopoulou and Havelund's observation that dynamic checking often requires special treatments ranging from extended semantics of temporal logics to monitoring and analysis algorithms [16].

We also examined the existing methods from the perspective of distinguishing endogenous and exogenous consistency checking. As mentioned in Section 2, endogenous consistency concentrates on a single behavioral model whereas exogenous consistency targets various types of models. In addition, exogenous consistency is further classified into vertical and horizontal consistency. Table 5 depicts the support of endogenous and exogenous (i.e., vertical and horizontal) consistency types of the selected primary studies. We can see that exogenous consistency (85.32%) is addressed more prominently than endogenous consistency (14.68%). Regarding exogenous consistency, we found that 66 studies (60.55%) focus on horizontal consistencies whilst 27 studies (24.77%) investigate vertical consistencies.

We found that seven of the studies (7.29%) out of 96 tackle the combination endogenous+horizontal consistency and two of the studies (2.08%) endogenous+vertical. There are two studies (2.08%) using the combination horizontal+vertical and one study (1.04%) uses endogenous+horizontal+vertical.

The year-wise distribution of endogenous and exogenous consistency checking types is shown in Figure 4. We can see that much of research attention has been paid to the endogenous consistency checking from 2007 to 2008. Apart from that, horizontal consistency has received more attention from 2005 to 2009, while comparatively less attention has been given to the vertical consistency checking.

As the complexity of the systems increases, the timing requirements become more and more stringent, especially if the system's reliability is a key concern. It is therefore necessary to investigate the studies that support for time-related consistency checking. Considering the existing methods in terms of support for a notion of time during consistency checking, we propose a three-level category of time support in consistency checking, which includes "1=Not considered", "2=Implicit using of underlying timing model or rules", and "3=Explicit

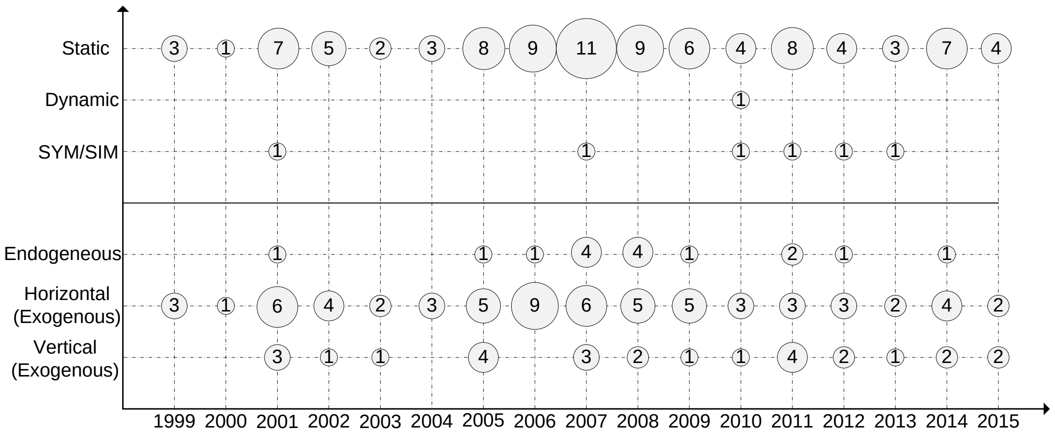

Fig. 4. Primary Studies per Year Grouped by Types of Consistency Checking

timing model and analysis"[7]. The first level refers to studies that do not consider time during consistency checking. The second and third level refer to studies that consider implicit usage of an underlying time model and an explicit timing model and analysis (i.e., using explicit timed models or real-time constraints), respectively.

Table 6. Support for the Notion of Time in Consistency Checking

Level	Studies	Total
1	S1–S5, S7, S10–S17, S19, S23–S29, S33, S37, S40, S41, S43, S45, S46, S48, S49, S51–S56, S58, S60–S68, S70–S72, S74–S94, S96	72
2	S6, S18, S22, S30, S32, S34, S39, S47, S50, S57, S73	11
3	S8, S9, S20, S21, S31, S35, S36, S38, S42, S44, S59, S69, S95	13

Table 6 presents the result of our SLR regarding the notion of time in consistency checking. We found that a majority of the existing studies do not consider support for time, i.e., 72 studies (75.00%) out of a total of 96. Only 11 studies (11.46%) support the second level "2=Implicit using of underlying timing model or rules", whereas, 13 studies (13.54%) explicitly address timed models and real-time constraints. Naturally, these 13 studies stem from the domain of embedded and real-time systems where time critical conditions are often the highest priority.

4.4 Consistency Checking Techniques (RQ2.3)

This section discusses the techniques used for checking the consistencies of software behavioral models. In particular, we have explored two major aspects of consistency checking techniques including the semantic domains (and correspondingly formal paradigms) employed by existing methods for formalizing the input models and constraints as well as the techniques (e.g., model checking, logical inference, theorem proving, etc.) that are used for performing consistency checking.

[7]Please refer to Table 18 in the Appendix C for further details of timing support scales

The descriptive results of investigating the semantic domain used by each approach are shown in Table 16 of Appendix A. A semantic domain, when appropriately employed, can help reduce the ambiguity in modeling behavior of software systems with precise mathematical terms [10]. In addition, grounding on a solid semantic domain enables the use of several model checkers and theorem provers for consistency checking. We adopt the classification of semantic domains into the following formal paradigms as proposed by [19].

- **State transition**: The description defines the transition relation on a set of states.
- **Algebra**: The description specifies the set of operations and their relations. An event is denoted by a function (also known as an operation). The behavior of functions is specified by a set of equations (axioms) that describes how these functions are related. A special form of algebra, namely, *process algebra*, in which operations are applied to elementary processes and events, describes how events may occur.
- **Logic**: The behavior of functions is specified by a set of equations (axioms) that describes how these functions are related [28].
- **Other**: The formal semantic domains that do not exactly belong to either of the aforementioned definitions are included in this particular category.

Table 7. Semantic Domains

Semantic Domain	**Studies**	**Total**
State Transitions	S2–S6, S8–S14, S16, S18, S20–S25, S27, S28, S30–S34, S36–S39, S43–S48, S50, S51, S53–S57, S60, S63–S66, S68, S69, S71–S74, S76–S79, S82–S84, S86, S87, S90, S91, S93, S94, S95, S97	69
Logic	S4, S11, S13, S18, S19, S21–S23, S26, S32, S34, S39, S42, S44, S45, S47, S50, S54, S57, S61, S67, S79, S84, S92–S94	26
Process Algebra	S1, S8, S15, S29, S35, S40, S41, S52, S58, S59, S62, S80, S85, S87–S89, S96	17
Others	S7, S17, S26, S49, S52, S70, S72, S75, S81	9

Table 7 presents the studies that are classified according to the aforementioned semantic domains. 57.02% of the studies (i.e., 69 out of 121 studies) are based on state transitions. 17 studies (14.05%) use process algebra as the semantic domain. 26 studies (21.49%) leverage different kinds of logics as their semantic domain. The category "Other" shown in Table 7 embraces all studies (9, i.e., 7.44%) that use a semantic domain different from the three domains mentioned above. Please note that a study may employ more than one semantic domain, and therefore, may be based on multiple domains. 20 studies (20.83%) out of a total of 96 use the combination of state transitions+logics as their semantic domains, while two studies (2.08%) use state transitions+process algebra. There are three studies (i.e., 1.04%) using the combinations state transitions+other, logics+other and process algebra+other.

Considering the consistency checking techniques of the existing studies, we adopted and extended the four main categories proposed by Spanoudakis and Zisman [39], which are: model checking (i.e., using or combining with existing model checkers), specialized algorithm (i.e., algorithms designed for analyzing models to detect inconsistencies), logical inference (i.e., using formal inference techniques to derive inconsistencies), and theorem proving (i.e., reasoning using theorem provers).

Analyzing the selected primary studies we have found that a wide range of studies used specialized algorithm to identify and detect the inconsistencies of software models, i.e., 51 studies (51.00%) out of 100. The main reason is that these approaches investigate different sets of inconsistency problems on particular domains and, therefore, often propose specific algorithms or heuristics for the particular contexts being studied. About 41 studies (41.00%) use model checking techniques given the existence of several model checkers such as NuSMV[8], SPIN[9], FDR[10], UPPAAL[11], GROOVE[12], and LSTA[13], to name but a few.

Table 8. Consistency Checking Techniques

Technique(s)	**Studies**	**Total**
Model Checking	S1, S4, S8, S9, S11, S13–S16, S18, S22, S23, S29, S32, S33, S34, S36, S39, S44–S47, S50, S53, S54, S57–S59, S63, S70, S78, S84, S85, S87, S88, S90–S94, S96	41
Specialized Algorithm	S2, S4–S7, S11, S12, S17, S20, S21, S24–S28, S30, S31, S35, S37, S38, S40, S41, S43, S44, S48, S49, S51, S52, S55, S60, S62, S64, S65, S66, S68, S69, S71–S77, S79, S80–S83, S86, S89, S95	51
Logic Inference	S10, S19, S42, S61, S67	5
Theorem Proving	S3, S56, S67	3

We note that, even though a reasonable number of studies (21.49%) are based on the logic semantic domain as mentioned above, logic inference techniques are used only in five studies for identifying inconsistencies or checking consistency of software models (5.00%). The use of theorem proving techniques is even less frequent, they are used only in three studies (3.00%). These results partially explains the emergence of model checking techniques backed by powerful model checkers [9].

Nonetheless, model checking techniques cannot thoroughly cover every aspect of consistency problems and may suffer from the problem of state explosion [8]. This could explain why a majority of the existing methods need to develop specialized algorithms for particular purposes. The summary of consistency checking techniques is shown in Table 8. Please note that, the sum of the numbers of studies on a technique category exceeds the total number of studies, because some studies used more than one technique. Two studies (2.08%) out of a total of 96 use the combination model checking+specialized algorithm to identify the inconsistencies of software models, while one study (1.04%) uses the combination specialized algorithm+theorem proving, and one study (1.04%) uses the combination logic inference+theorem proving techniques.

4.5 Inconsistency Handling (RQ2.4)

Detecting behavioral inconsistencies of software systems is important but still far from complete. Handling inconsistencies has been considered a central task in consistency management [39, 44]. Handling of inconsistencies addresses how to deal with any inconsistencies

[8]See http://nusmv.fbk.eu
[9]See http://spinroot.com
[10]See https://www.cs.ox.ac.uk/projects/fdr
[11]See http://www.uppaal.org
[12]See http://groove.cs.utwente.nl
[13]See http://www.doc.ic.ac.uk/ltsa

and analyzing the impacts and consequences of particular methods of dealing with inconsistencies [39]. We derive a scale from 1–5 for different degrees of handling with inconsistencies based on the set of activities proposed by Spanoudakis and Zisman [39]:

– 1. Not mentioned / not considered
– 2. Systematic inconsistency diagnosis
– 3. Identifying handling actions
– 4. Evaluating costs and risks
– 5. Automated action selection and execution

Table 9. Degree of Inconsistency Handling

Level	Studies	Total
1	S2–S10, S13, S14, S16, S17, S20–S27, S29–S34, S36–S38, S40–S44, S46–S59, S63–S70, S72–S78, S80–S82, S85–S96	78
2	S1, S11, S12, S15, S18, S19, S22, S28, S35, S39, S45, S60–S62, S71, S79, S83, S84	18
3–5	–	0

In summary, inconsistencies can be handled by different activities including systematically diagnosing inconsistencies and their causes, identifying necessary actions for resolving inconsistencies, evaluating costs and risks, and automatically selecting and execution the corresponding actions.

Table 9 shows our analysis results for inconsistency handling. We have found that most of the existing approaches provide little or even no support for this important aspect. In particular, a large number of studies (78, i.e., 81.25%) do not consider any sort of inconsistency handling. Only 18 studies (18.75%) support certain forms of inconsistency diagnosis, for instance, analyzing the counterexamples or error traces back to the origins of the problems. Unfortunately, none of the studies supports any higher levels of inconsistency handling (i.e, from 3–5).

4.6 Automation Support (RQ2.5)

Some prior secondary studies such as [28] examined the support for automation in model consistency checking but they propose only two levels "yes" (automated) and "no" (manual). In fact, there exist some techniques where human intervention is partially necessary for specifying models or consistency rules, and the rest can be automatically performed. Moreover, there are different phases in consistency checking including specification, checking, and possibly handling inconsistencies (as mentioned above). Therefore, first of all, we decided to refine the automation support into a three-level scale in order to cover the aforementioned cases: which are "1=Manual", "2=Semi-automated" and "3=Fully automated"[14]. Moreover, we investigate the automation support in three main phases of consistency checking (i.e., specification, checking and handling inconsistencies) as analyzed above and assign the corresponding level for each primary study.

We noticed that the phase consistency checking, due to sound formalization and well-defined checking algorithms and/or techniques, can be performed automatically (i.e., reaching the level 3 "Fully-automated"). Unfortunately, this is not the case for inconsistency handling.

[14]Please refer to Table 20 in the Appendix C for further details of automation support levels

As only a few of the existing approaches consider a limited form of inconsistency handling, which is the diagnosis of the checking results, the level of automation supported by these approaches is 2 (i.e., "Semi-automated") because human intervention is necessary for investigating the yielded errors. For these reasons, we do not present details for the two phases consistency checking and inconsistency handing, but rather focus only on the specification phase.

Table 10. Automation Support for Specification Phase

Automation Level	**Studies**	**Total**
1	S3, S5, S10, S11, S12, S18, S19, S21, S22, S23, S26, S29, S35, S43, S44, S46, S47, S48, S50, S54, S57, S59, S65–S67, S70, S77, S79, S93, S94	30
2	S1, S2, S4, S6, S7, S9, S13–S17, S24, S25, S27, S28, S30–S34, S36, S38, S39–S42, S45, S49, S51–S53, S55, S56, S58, S60–S64, S68, S69, S71–S76, S78, S80–S92, S95, S96	62
3	S8, S20, S37, S74	4

We present the distribution of the existing approaches in the literature with respect to the aforementioned scale of automation support in the specification phase in Table 10. A considerable number of studies (31.25%) falls into Level 1 (i.e., "Manual") because they either assume the existence of formal logic constraints or require manual efforts in specifying the input consistency constraints (e.g., rules specified in LTL/CTL etc.) or, sometimes, the input models. We note that these manual tasks ask for considerable knowledge of the underlying formalisms and formal techniques, and therefore, are often not easy to use for many software engineers. A larger number of studies (64.58%) assume the existence of the input models but propose automated transformations of these inputs into formal representations. For this reason, we consider them "semi-automated" because strictly speaking the input models assumed by these approaches are often manually created. Nonetheless, these are design models and created during the modeling and development of software systems, anyway, and there is automated translation support. Only four studies (4.17%) can be considered "fully-automated" as they encode consistency rules in the corresponding algorithms or libraries and, therefore, do not require any manual effort in the specification phase. However, the downside could be that the hard-coded consistency rules could lessen the extensibility and generality, and therefore, the scope of application of these approaches.

4.6.1 Tool Support for Consistency Checking (RQ2.5, RQ3). Another important dimension is to consider tool support provided by the existing approaches for consistency checking. We believe that more or better tool support would showcase a proof of feasibility and attract the attention of practitioners to the corresponding techniques. The lack of any sort of tool support (e.g., research prototypes, demos, etc.) will hinder practitioners to use or at least explore the corresponding proposed methods and techniques, and therefore, hinder the transfer of research results into industrial practice.

We propose three levels of evaluation of tool support as shown in Table 22: "1=Not mentioned/Not considered", "2=Only using existing tools/libraries", and "3=Prototypes (includes using existing tools/libraries)"[15].

[15]Please refer to Table 22 in the Appendix C for a full description of tool support levels

Table 11. Evidence of Tool Support for Consistency Checking

Level	Studies	Total
1	S2, S6, S11, S17, S24, S27, S38, S42, S64, S66, S68, S71, S73, S75, S81, S83	16
2	S3, S4, S9, S13, S15, S22, S23, S29, S31, S33, S40, S41, S43, S47, S50, S55–S59, S69, S70, S74, S77, S78, S80, S85, S87–S91, S93, S95	34
3	S1, S5, S7, S8, S10, S12, S14, S16, S18–S21, S25, S26, S28, S30, S32, S34–S37, S39, S44–S46, S48, S49, S51–S54, S60–S63, S65, S67, S72, S76, S79, S82, S84, S86, S92, S94, S96	46

The distribution of levels of tool support found in the literature on software model consistency checking is shown in Table 11. We did not find evidence of tool support for detecting and handling inconsistencies in 16 studies (16.67%). 34 studies (35.42%) used existing tools and libraries to carry out consistency checking, for instance, model checkers such as SPIN, (Nu)SMV, UPPAAL, FDR, GROOVE, LSTA, and Maude or theorem provers such as SPASS and Z/Eves. Out of 96 there are 46 studies (47.92%) that provide specific tool support for checking consistency between software models including the development of prototypes or tool-chains that combine the implementation of various aspects such as model transformations, consistency checking algorithms, and/or existing tools and libraries. Figure 5 shows the distribution of primary studies per year grouped by levels of tool support. We can see that the trend of tool support for checking consistency by using existing tools and libraries (i.e., level 2) has received more attention from 2005 to 2008, while comparatively the developing prototypes or tool-chains (i.e., level 3) reached a peak in 2014.

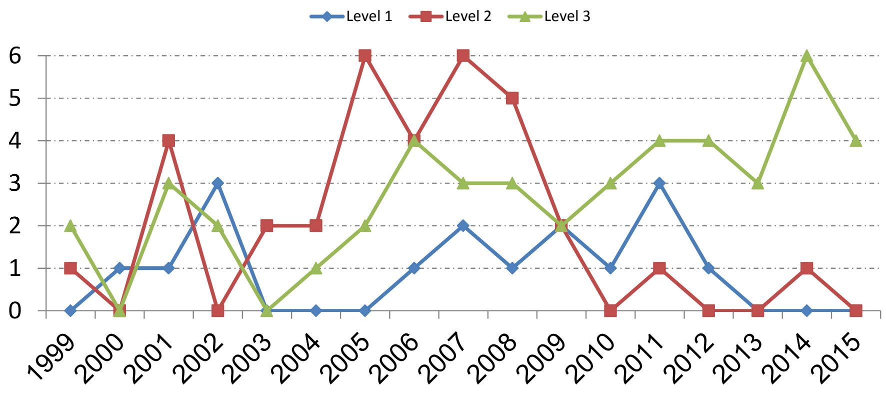

Fig. 5. Levels of Tool Support per Year Distribution

4.7 Types of Study and Evaluation (RQ2.6)

As a part of this SLR we investigated the nature of evidence presented in the selected primary studies and analyzed how an evaluation is performed and reported. The main motivation is that different types of studies provide different strengths of evidence and evaluation. Practitioners could take the strength of evidence and evaluation into consideration before adopting a specific methodology and tool.

We adopted the classification of types of study and evaluation in six categories as proposed by Chen and Ali Babar [7]. The six types of study and evaluation are Rigorous Analysis (RA), Case Study (CS), Discussion (DC), Example (EX), Experience Report (ER), Field Study (FS), Laboratory Experiment with Human Subjects (LH), Laboratory Experiment with Software Subjects (LS), and Simulation (SI)[16].

Table 12. Types of Study and Evaluation

Types	**Studies**	**Total**
DC+EX	S7, S16, S47, S50, S57	5
RA+DC	S2, S13, S30, S31, S52	5
RA+DC+EX	S8, S9, S15, S18–S20, S24, S25, S28, S32, S36–39, S41, S45, S46, S48, S49, S53, S54, S58, S59, S61–S66, S74, S76, S79, S82, S87, S90	35
RA+DC+EX+LS	S12, S22, S26, S35, S55, S72, S81, S83	8
RA+EX	S1, S3, S4, S6, S10, S11, S14, S17, S21, S23, S27, S29, S33, S34, S40, S42–S44, S51, S56, S60, S67, S68, S71, S73, S75, S77, S78, S80, S84–S86, S88, S89, S91, S93–S96	39
RA+EX+ER	S5, S69, S70, S92	4

Analyzing the data extracted from the selected studies, we see that a study may be a combination of different types. For instance, S8 presents and discusses formal foundations, consistency problems and gives concrete examples in BPEL. Thus, we clustered the data and divided them into appropriate groups based on the combination employed by each study. Table 12 shows the results of data analysis and clustering.

We observed that a great amount of the studies (36.46%) use the combinations RA+DC+EX or RA+EX (40.63%). There are eight studies (8.33%) using the combination RA+DC+EX+LS that, in addition to what is done in RA+DC+EX, also perform some sorts of evaluation with software models, for instance, rigorously estimating the scalability, correctness, algorithm precision or comparing with other approaches. Only a few studies (4.17%) report RA+EX+ER, experiences in some academic settings or application scenarios derived from industry practice. Unfortunately, none of the existing studies presents clear evidence using any sort of empirical study or validation.

Along with the aforementioned investigation on types of study and evaluation, we also assessed the level of rigor of the reported evaluations. Rigor is concerned with assessing how an evaluation is performed and reported [15]. In particular, we examined how well the selected studies report their evaluations with respect to three dimensions: the context of the evaluations (C), the design of the studies conducted in the evaluation (S), and the validity discussion (V). We leverage the score for each rigor dimension as proposed in [22] that comprises three levels "1=Weak", "2=Medium", and "3=Strong"[17]. Figure 6 depicts an overview of the results regarding the aforementioned dimensions of the evaluation's rigor.

We note that the rigor measure proposed above could be more appropriate for empirical studies. Applying the rigor measure uncovered that many studies mentioned dimensions related to rigor but do not describe these fully. Nonetheless, we consider this measure in

[16]Please refer to Table 24 in the Appendix C for a full description of these categories
[17]Please refer to Table 25 in the Appendix C for a full description of the rigor levels

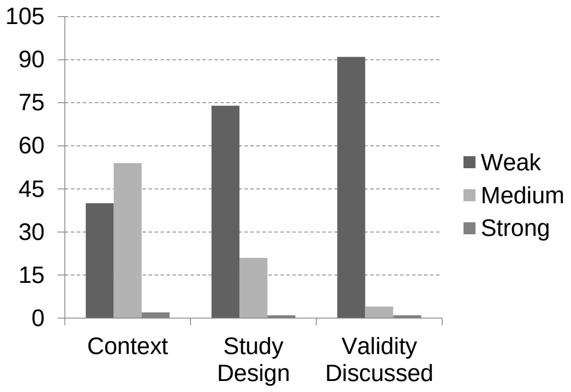

Fig. 6. Sum of Rigor Scores per Dimension

our study in order to show the lack of rigorous evaluations and provide some insights which researchers can consider when designing and conducting their studies to achieve a reasonable degree of quality.

The results shown in Figure 6 indicate that very few studies have been scored the strongest level in each dimension. The evidence regarding describing the study design and the validity discussions is very poor. It also reflects our analysis on the type of study and evaluation where only a small number of studies perform certain rigorous experiments with software models.

To illustrate the distribution of rigor measures, we considered a collective rigor metrics for each study, namely, R, which is calculated as $R = C + S + V$, following the suggestions of Ivarsson and Gorschek [22] and [15]. Thus, the value of R for each study is an integer in the range of 3–9. We show the distribution of the collective rigor metrics R in Figure 7.

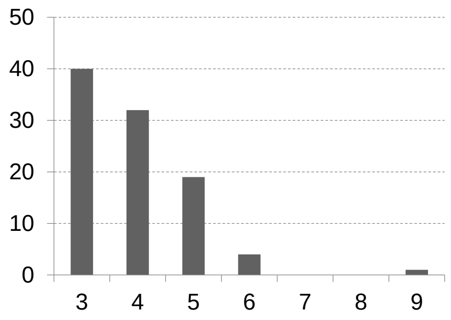

Fig. 7. Distribution of the Collective Rigor Metrics R

In accordance with our prior observation, the collective scores of the rigor metrics of the selected studies are distributed significantly towards the lower side of 3 (40 studies), 4 (32 studies), and 5 (19 studies), respectively. None of these studies could achieve the collective score of 7 or 8. The only study that reaches the best collective score is S26 in which the authors present rigorously and thoroughly all three dimensions of evaluation rigor. Figure 8 presents the temporal distribution of the rigor metrics for each study. We can see that the trend of rigor evaluations of studies increased in the last ten years, but overall the rigor is still rather low.

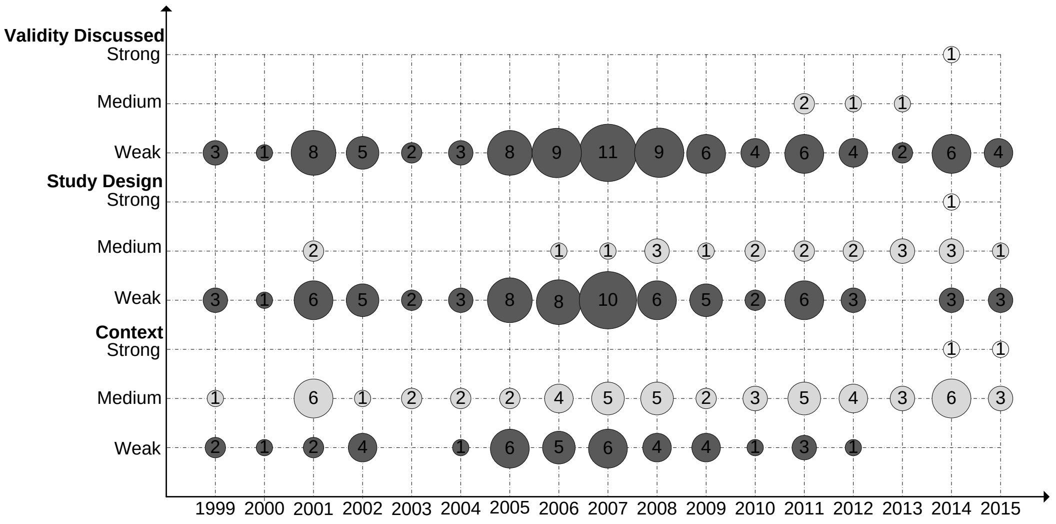

Fig. 8. Trend of Rigor Dimension per Year

4.8 Practical Impact (RQ3)

Analyzing the practical impact, or in other words, the effectiveness, of the existing methods for consistency checking of software behavioral models is a very challenging task because there is no consensus on an ultimate measurement or metrics for this aspect, to the best of our knowledge. After studying other relevant surveys in the field of computer science and software engineering, such as [1, 15], to name but a few, we propose to examine the practical impact of the existing methods from different dimensions, including citations (how a study influences the others), CASE/IDE tool integration (support in daily working environment of practitioners), tool support (feasibility study: mentioned in Section 4.6.1), degree of evidence, and the applicability of the proposed techniques in industrial settings.

4.8.1 Citations. Table 13 presents an overview of the top 10 of most highly cited studies along with the year of publishing, the total number of citations, and the average number of citations per year. The numbers of citations are obtained from Google Scholar. We note that these numbers are roughly estimated because Google Scholar may mistakenly count the citations of different authors who have similar or the same names. Also please note that these numbers can change over time. We sampled two times on Aug 9, 2015 and Aug 19, 2015 and noticed slight differences. The presented numbers are obtained as of date Aug 19, 2015. Nonetheless, the average of citations per year is quite stable as the change margins are small.

Because each of the studies in the top 10 is highly and frequently cited, we can draw the conclusion that at least the studies in the top 10 have substantial influence on other researchers. Unfortunately, we could not find sufficient evidence to conclude on the correlation between the citations and other aspects such as the used semantic domains, methods, techniques, or application domains.

Table 13. Top 10 Most Cited Studies

SID	Author(s)	Year	Total Citations	Avg. Citations per Year
S6	Bernardi et al.	2002	322	25
S63	Schäfer et al.	2001	247	18
S55	Nejati et al.	2007	243	30
S27	Harel et al.	2002	196	15
S36	Knapp et al.	2002	188	14
S15	Engels et al.	2001	169	12
S76	Van der Aalst et al.	2008	152	22
S18	Eshuis & Wieringa	2004	137	12
S28	Hausmann et al.	2002	127	10
S83	Weidlich et al.	2011	124	31

4.8.2 Development Tool Support and Integration. The second dimension of the practical impact that should be considered is the integration of the proposed techniques into the working environments of practitioners (such as software analysts, software architects, software developers), i.e., development tool support. Here, specifically, CASE[18] tools and IDEs[19]) are relevant. CASE/IDE integration could foster the practical use of the consistency checking techniques. It could help users of the development tools to obtain feedbacks and support them in handling inconsistencies detected in the models. Researchers could benefit by receiving improvement suggestions from the practitioners who are using the corresponding CASE/IDE tools in industrial settings.

We propose a three-level scale to assess the evidence of IDEs/CASE integration that comprises “1=Not mentioned/Not considered”, “2=Proposed/planned integration”, and “3=Fully implemented integration”[20]. Table 14 shows the analysis result of CASE/IDE integration found in the selected studies. The weakest scale is applied for studies that do not consider or mention at all about the integration with any CASE/IDE. There are 80 studies (83.33%) that fall into this level. On the other end of the spectrum, there are only five studies (5.21%) that develop fully working integration with existing CASE/IDE tools, i.e., reaching the strongest level. The rest of the studies (11.46%) (i.e., 11 out of total 96 studies) mainly propose plans for CASE/IDE integration or partially implement the integration.

Table 14. CASE/IDE Support and/or Integration

Level	Studies	Total
1	S2–S4, S6, S7, S9–S15, S17–S23, S26, S27, S29–S33, S35–S43, S45–S47, S49–S52, S55–S59, S61–S68, S71–S78, S80–S96	80
2	S5, S24, S28, S34, S48, S53, S54, S60, S69, S70, S79	11
3	S1, S8, S16, S25, S44	5

[18]CASE: Computer-Aided/Assisted Software Engineering
[19]IDE: Integrated Development Environment
[20]Please refer to Table 21 in the Appendix C for a full description of CASE/IDE support/integration levels

4.8.3 Application in Industrial Settings. For many research works in the field of software engineering, it is significant to apply and evaluate newly proposed or improved techniques in a real industrial settings. As Ivarsson and Gorschek [22] emphasize, research methods and techniques need to provide tangible evidence of the advantages of using them in order to impact industry. Ivarsson and Gorschek [22] also suggest a step-wise validation to enable researchers to test and evaluate in real settings with real users and applications (i.e., *empirical evidence*).

Hence, in our study we examined the evidence of any industrial settings presented in the chosen primary studies. The types of evidence could be critical for researchers to identify new topics for empirical studies, and for practitioners to assess the maturity of a particular method or tool. Some of the approaches use (small-scale) industrial scenarios to illustrate the applicability of their techniques whilst others evaluate their approaches by using small examples or cases (i.e., providing the readers with a rough idea of how to apply and use the proposed approaches). We adopted the 6-level scale of evidence as proposed by Alves et al. [1][21]:

- 1. No evidence provided.
- 2. Evidence obtained from demonstration or working out toy examples.
- 3. Evidence obtained from expert opinions or observations.
- 4. Evidence obtained from academic studies.
- 5. Evidence obtained from industrial studies.
- 6. Evidence obtained from industrial practice.

Table 15. Levels of Empirical Evidence

Level	Studies	Total
1	–	0
2	S2–S11, S13–S19, S21, S23–S31, S34, S36, S38, S40–S47, S49, S51, S53, S54, S56–S58, S60, S62–S64, S66–S68, S71, S73–S80, S84–S96	73
3	S1, S12, S22, S32, S33, S39, S44, S48, S50, S59, S61, S72	12
4	–	0
5	S20, S35, S37, S52, S55, S65, S69, S70, S81, S82, S83	11
6	–	0

The data of our investigation and analysis shows that 73 out of the total 96 studies (76.04%) are tested and evaluated using toy examples. 12 studies (12.50%) use some scenarios obtained from expert opinion or observation. Only 11 studies (11.46%) show a higher level of evidence based on industrial studies. None of the studies support the Levels 4 and 6. Table 15 shows the distribution of empirical evidence found in the literature on software model consistency checking. We note that there are no studies of Level 1 either. After closer analysis, we found that this is mainly due to the fact that studies falling into this level have been excluded because they are not presented at an acceptable quality level (c.f. Section 3.2.2).

We map roughly the six corresponding levels of evidence mentioned above into two levels that indicate whether the proposed methods/techniques are tried out in any industrial settings. Levels 4–6 of evidence can be considered being applied or tested in industrial settings whilst Levels 1-3 are not. We summarized the data according to this two-level scale,

[21] Please refer to Table 23 in the Appendix C for a full description of evidence levels

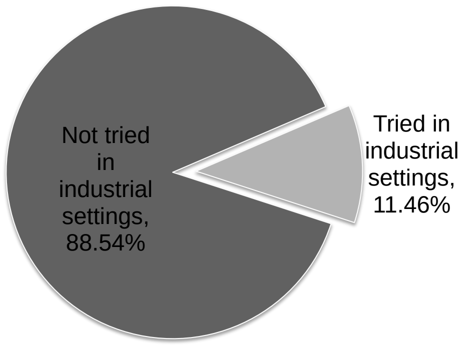

Fig. 9. Studies Tried in Industrial Settings

shown in Figure 9. Only 11 studies have validated or evaluated their proposed techniques and tools in any sort of industrial settings. The majority of studies (85, i.e., 88.54%) did not (but rather using other means such as rigorous analysis, discussion, and toy examples). We also observed that only one study (S70) out of 11 studies (11.46%) was evaluated in industrial settings in the year 2001 while all the others were evaluated between 2007 and 2015. That is, while the level industrial evidence seems rather weak in the total set of studies, it has clearly increased in recent years.

5 DISCUSSIONS

In this section, we will discuss the limitations of existing methods to answer the remaining Research Question RQ4 (c.f. Section 3.1.2) and discuss the validity of our study.

5.1 Limitations of the Existing Methods

This SLR is an attempt to give a comprehensive view of the state-of-the-art of research in software behavioral model consistency checking. Based on our analysis and interpretation presented in this article, we observe a number of limitations and open problems in the existing methods that could potentially point into interesting further research directions.

5.1.1 Unbalanced Focus of Consistency Checking Problems. There is a predominance of primary studies concentrating on static consistency checking and little attention has been paid on dynamic checking or consistency checking by means of symbolic execution or simulation. The huge advantage of static checking is to identify many inconsistencies without the need of and/or connection to any running systems. However, static checking also has considerable limitations like suffering from the state explosion problem [8] or being computationally expensive. A large number of static checking based approaches use model checkers, which only support finite and discrete data types (e.g., boolean, integer, vector) and provide limited or even no support for continuous data types (e.g., floating-point numbers) or strings.

Many distributed software systems are built upon highly flexibly and dynamic architectures like event-driven architectures [32]. Besides, there is also an emerging trend in software development on supporting higher degrees of flexibility [17, 42] or on-the-fly adaptations [29] at runtime. Static consistency checking of these types of flexible systems might not be solely enough as several unanticipated changes can happen during system execution. Symbolic execution or simulation can help to partially examine these dynamic changes, but ultimately, dynamic checking, through either system instrumentation or monitoring, should be employed for catching any potential inconsistencies. However, runtime verification is much more challenging [16] than static checking, and therefore, needs to draw more attention from both researchers and practitioners.

Similar observations can be made (in a less severe manner) for the case of support for time-related consistency checking. A rather small portion of the existing methods investigates time-related inconsistencies, especially in the context of real-time constraints. With respect to the niche fragment of the domain of embedded and real-time systems versus the remaining application domains, we do not see this as critical nuisance.

In summary, while the consistency topic overall is mature, much more work needs to be done in areas, such as dynamic checking, simulation and timing consistency checking of software behavioral models.

5.1.2 Focus on Specific Model Types. Besides consistency checking problems, we identified, research on behavioral models consistency checking focuses on four major types of behavior models, which are statecharts, sequence diagrams, process models, and activity diagrams. Considerably less attention has been paid on collaboration diagrams, LSCs, stateflow, and labeled transition systems. The semantics of stateflow and labeled transition systems are quite complex, which makes these models a weak option for modeling the system behavior. Hence, more work is required on all these models.

5.1.3 Lack of Consideration for Inconsistency Handling. Spanoudakis and Zisman [39] emphasized the vital role of inconsistency handling and suggested many aspects to consider for adequately handling consistency issues. Improper or inadequate inconsistency handling could lead to severe negative effects [39] and could be one of the major obstacles to making significant practical impact or to transferring the proposed methods and techniques to industry practice. Unfortunately, we found limited evidence of inconsistency handling on diagnosing inconsistencies. Most of the studies produce consistency checking outcomes in terms of formal representations such as counterexamples [9] or erroneous traces of states. However, this assumption should be reconsidered because most of the practitioners (e.g., software engineers) do not have sufficient knowledge of the underlying formal methods to understand these outcomes. As a result, a considerable amount of time and effort is consumed by the practitioners in order to handle the detected inconsistencies. It would be more pragmatic to present the consistency checking outcomes in appropriate forms of representation (which can be textual or graphical notations or visualization) with suitable abstractions that practitioners can better comprehend and use to resolve the problems detected in the models. Further important activities such as identifying corresponding handling actions, estimating the costs and risks of handling inconsistencies, and eventually choosing and executing handling actions need to be taken into account.

5.1.4 Lack of Rigorous Studies, Evaluation, and Practical Impacts. Lack of rigorous studies, evaluation, and practical impacts reflect the quality of studies being conducted in the literature of software behavioral model consistency checking. As we observed in Section 4.7, several primary studies tend to focus on introducing new ideas and solutions to consistency checking problems but fail to evaluate their contributions properly and show the validity of their approaches in larger contexts. This issue is mainly due to less consideration of rigorous study design and validity discussion. Moreover, our findings in Section 4.7 also show that many of the research results have not been tried out in any sort of industrial settings. As a consequence, these results remain pure academic contributions and exercises. This could be explained as academic results tend to be context-specific, and thus, difficult to generalize to many industrial situations Alves et al. [1]. While the average rigor of evaluations has increased in the last ten years, it is still at a very low level. The strongest case found in this SLR is a study which has been thoroughly and rigorously designed with test cases based on industry practice. We expect that the categories of types of study and evaluation presented in Table 24 and discussed in Section 4.7 could be considered for design and evaluation of future studies to achieve more scientific and repeatable results that have greater practical impact.

Another aspect that is crucial for gaining more practical impact is to provide adequate tool support for consistency checking and/or integration with existing development tools (such as CASE/IDE tools). It is noted that automation is usually achieved by integrating the techniques with existing software development tools. The benefit of tool support for consistency checking and development tool integration, which has been underlined above, is twofold. It provides intuitive means for fostering the application of the implemented techniques in industrial settings, and it allows for obtaining valuable feedbacks from practitioners who are able to use the tools for their daily tasks. Unfortunately, only a small number of research results from the existing studies have been incorporated in commercially used development tools. The rest is either focused on developing research prototypes or using existing tools and libraries (e.g., model checkers, theorem provers).

5.2 Study Validity

The main threats to validity in this systematic review are potential bias in our search strategy, the selection of the studies to be included, and data extraction. We aimed for an exhaustive list of high quality primary studies. Therefore, we followed strictly the guidelines recommended in [26] and took into account lessons learned in [5]. We have developed a clear protocol for searching and choosing primary studies (c.f. Section 3) including defining research questions, inclusion and exclusion criteria, and search strategy. The first author prepared the protocol, and the second and third authors reviewed and assessed whether the review plan was appropriate for addressing the research questions. We also obtained critical comments from the journal reviewers on earlier versions of the paper that lead to revisions of our research questions and replications of our study.

Despite a well-defined systematic procedure, we acknowledge that there may be missing primary studies, for instance, due to the coverage and quality of the search engines and search portals that we used in our search. It is crucial with respect to the extent where relevant keywords are not standardized or clearly defined [5]. That is, various studies may use the same terms with different semantics. To deal with this kind of threat, we have performed a number of activities. On the one hand, we considered our attempt reported in the earlier version of the paper as a pilot study. Then we synthesized relevant studies in the literature and, with help from internal and external reviewers, came up with a set of clearly defined concepts and categories as shown in Section 2. We used this set as a basis for guiding the replication of our study and for analyzing the selected primary studies. Secondly, we decided to carry out a "*snowballing*" process [6] in which two researchers manually scanned and analyzed the references and citations of the primary studies retrieved from the automated search with the search engines/portals. The main goal is to make sure that our study also covers follow-up works that might exist but have not been included in the search. As we presented in Section 3.2.1 the snowballing process has caught 72 missing studies out of which 26 studies have been selected.

Bias in study selection may be due to different interpretation and understanding of the researchers involved in the study and potential misalignment between our term and category definitions and other definitions. To alleviate these problems, two researchers performed an in-depth analysis and selection of primary studies with respect to the predefined inclusion and exclusion criteria and clearly recorded the reasons for including or excluding. The third researcher reviewed independently and provided corresponding judgments, especially for the cases of discrepancy or "in doubt". Then all researchers discussed and came to the final conclusion for each study.

During the data extraction phase, we confronted some difficulties, and thus, potential bias, in extracting objective information from the selected primary studies. First of all, this issue could be due to the subjective interpretation and/or poorly described report of relevant terms, methods, techniques, evaluations, and so on. Another difficulty is that different studies could choose different research and evaluation methods as mentioned in Section 4.7, and therefore, can compromise the accuracy of data extraction. We tried to minimize this bias by, first of all, defining the extraction strategy and date format with clear encoding descriptions in order to ensure consistent extraction of relevant data. Furthermore, the extraction has been performed independently by two researchers. Any sort of discrepancy from all involved researchers has been adequately recorded. The third researcher then performed the final reviewing and cross-checking. After that, we conducted discussion meetings to resolve any remaining divergence and disagreements. Nonetheless, we acknowledge that there is still a certain possibility of misunderstanding regarding the way we extracted data from the primary studies.

Considering the reliability, which concerns the ability of repeating a study with the same results, we have compared the results obtained in the earlier version and in the replication of this SLR. Apart from additional studies due to newly added keywords and the extended timeframe (shifted from 2013 to July 2015), we observed no significant differences between the two results. Given the well-defined protocol, data encoding, extraction, and analysis, we could expect that replications of our study should offer similar results. Precisely speaking, there might be slight variations of the search strategy and/or the set of selected primary studies but the underlying trends of our findings should remain unchanged.

Regarding external validity concerned with generalizing our study's findings, we aimed for a representative set of primary studies with clear research scope and objectives. We set no constraints on the time period such that our SLR can cover from the beginning of research on consistency checking in the eighties to the year 2015. Nonetheless, our findings might not be the same when being generalized to broader scopes or time periods. This could be due to the fact that our current findings are mainly based on qualitative analysis. We could consider rigorous quantitative analysis and inferences to enable the possibility of analytical and statistical generalizations but this is rather beyond the scope of this SLR.

6 CONCLUSIONS

This paper presents a systematic review of the literature on research in software behavioral consistency checking. We aimed to identify the current trend in this field and investigate various dimensions ranging from used methods, languages, techniques to the practical impacts of the identify studies. To achieve this, we have identified a total of 1770 studies by combining automated searches and manual snowballing, out of which 96 have been studied in-depth according to our predefined SLR protocol. Through in-depth analysis and interpretation of the collected data, we obtain many interesting findings along with a number of gaps and open problems that could provide insights for further investigation. In summary, there are promising accomplishments thanks to sound formal foundations that have been employed in most of the existing studies for consistency checking of software behavioral models. Nevertheless, future studies should pay sufficient consideration for improving the quality of study design and conducting evaluations to achieve solid and repeatable scientific results that have greater impact in both academia and industry. Moreover, the application of research outcomes in industrial settings should also be considered and planned adequately in earlier stages. For this purpose, appropriate strategy for inconsistency handling, better tool support for consistency checking and/or development tool integration should be considered in future studies. Our proposed categories and encoding in this SLR, which is adapted from other relevant studies, can also be considered as a guideline or recommendation for practitioners in evaluating relevant methods and techniques as well as for researchers in designing rigorous studies and evaluations that aim for higher practical impact.

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers and the editor for the thorough feedback on earlier versions of this paper. Their comments and suggestions were of great help.

REFERENCES

[1] Vander Alves, Nan Niu, Carina Alves, and George Valença. 2010. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology* 52, 8 (Aug. 2010), 806–820. DOI:http://dx.doi.org/10.1016/j.infsof.2010.03.014

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (oct 2010), 2787–2805. DOI:http://dx.doi.org/10.1016/j.comnet.2010.05.010

[3] Victor R. Basili and David M. Weiss. 1984. A Methodology for Collecting Valid Software Engineering Data. *IEEE Trans. Softw. Eng.* 10, 6 (Nov. 1984), 728–738. DOI:http://dx.doi.org/10.1109/TSE.1984.5010301

[4] Eerke Boiten, Howard Bowman, John Derrick, Peter Linington, and Maarten Steen. 2000. Viewpoint consistency in {ODP}. *Computer Networks* 34, 3 (2000), 503 – 537. DOI:http://dx.doi.org/10.1016/S1389-1286(00)00114-6

[5] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software* 80, 4 (2007), 571–583. DOI:http://dx.doi.org/10.1016/j.jss.2006.07.009

[6] David Budgen, Andy J Burn, O Pearl Brereton, Barbara A Kitchenham, and Rialette Pretorius. 2011. Empirical evidence about the UML: a systematic literature review. *Softw. Pract. Exper.* 41, 4 (April 2011), 363–392. DOI:http://dx.doi.org/10.1002/spe.1009

[7] Lianping Chen and Muhammad Ali Babar. 2011. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology* 53, 4 (2011), 344–362. DOI:http://dx.doi.org/10.1016/j.infsof.2010.12.006

[8] EM Clarke, William Klieber, M Nováček, and P Zuliani. 2011. Model Checking and the State Explosion Problem. In *International Summer School on Tools for Practical Software Verification (LASER) - Revised Tutorial Lectures*. Springer, Elba Island, Italy, 1–30. DOI:http://dx.doi.org/10.1007/978-3-642-35746-6_1

[9] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. 1999. *Model Checking*. MIT Press, Cambridge, MA, USA.

[10] Gregor Engels, Jochem M. Küster, Reiko Heckel, and Luuk Groenewegen. 2001. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *8th European Software Engineering Conference, Held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, NY, USA, 186–195. http://dx.doi.org/10.1145/503209.503235

[11] Rik Eshuis and Paul Grefen. 2007. Structural Matching of BPEL Processes. In *Fifth European Conference on Web Services (ECOWS'07)*. IEEE, Halle, Germany, 171–180. DOI:http://dx.doi.org/10.1109/ECOWS.2007.22

[12] Anthony Finkelsteiin, Dov M. Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. 1993. Inconsistency Handling in Multi-Perspective Specifications. In *Proceedings of the 4th European Software Engineering Conference on Software Engineering (ESEC '93)*. Springer-Verlag, Berlin, Heidelberg, 84–99. http://dl.acm.org/citation.cfm?id=645384.651465

[13] Pascal Fradet, Daniel Le Métayer, and Michaël Périn. 1999. Consistency checking for multiple view software architectures. *ACM SIGSOFT Software Engineering Notes* 24, 6 (Nov. 1999), 410–428.

[14] David Frankel. 2002. *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, Inc., New York, NY, USA.

[15] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. 2014. Variability in Software Systems–A Systematic Literature Review. *IEEE Transactions on Software Engineering* 40, 3 (March 2014), 282–306. DOI:http://dx.doi.org/10.1109/TSE.2013.56

[16] Dimitra Giannakopoulou and Klaus Havelund. 2001. *Runtime Analysis of Linear Temporal Logic Specifications.* Technical Report August. Research Institute for Advanced Computer Science. 1–11 pages. http://www.riacs.edu/research/technical_reports/TR_pdf/TR_01.21.pdf

[17] Stijn Goedertier, Jan Vanthienen, and Filip Caron. 2015. Declarative business process modelling: principles and modelling languages. *Enterprise Information Systems* 9, 2 (2015), 161–185. DOI:http://dx.doi.org/10.1080/17517575.2013.830340

[18] G. Goth. 2008. Ultralarge Systems: Redefining Software Engineering? *Software, IEEE* 25, 3 (May 2008), 91–94. DOI:http://dx.doi.org/10.1109/MS.2008.82

[19] Henri Habrias and Marc Frappier. 2006. *Software Specification Methods - An Overview Using a Case Study.* ISTE Ltd., London, UK.

[20] David Harel. 1987. Statecharts: a visual formalism for complex systems. *Science of Computer Programming* 8, 3 (1987), 231–274. DOI:http://dx.doi.org/10.1016/0167-6423(87)90035-9

[21] Zbigniew Huzar, Ludwik Kuzniarz, Gianna Reggio, and Jean Louis Sourrouille. 2005. Consistency problems in UML-based software development. In *International Conference on UML Modeling Languages and Applications (UML) Satellite Activities – Revised Selected Papers*. Springer, Berlin, Heidelberg, 1–12. DOI:http://dx.doi.org/10.1007/978-3-540-31797-5_1

[22] Martin Ivarsson and Tony Gorschek. 2011. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering* 16, 3 (2011), 365–395. DOI:http://dx.doi.org/10.1007/s10664-010-9146-4

[23] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.

[24] Barbara Kitchenham and Pearl Brereton. 2013. A systematic review of systematic review process research in software engineering. *Information and Software Technology* 55, 12 (2013), 2049–2075. DOI:http://dx.doi.org/10.1016/j.infsof.2013.07.010

[25] Barbara Kitchenham, David Budgen, and Pearl Brereton. 2015. *Evidence-Based Software Engineering, Empirical SE, Software Design.* CRC Press, Boca Raton, FL. 399 pages.

[26] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering.* Technical Report EBSE 2007-001. Keele University and Durham University Joint Report.

[27] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Inf. Softw. Technol.* 51, 1 (Jan. 2009), 7–15. DOI:http://dx.doi.org/10.1016/j.infsof.2008.09.009

[28] Francisco J. Lucas, Fernando Molina, and Ambrosio Toval. 2009. A systematic review of UML model consistency management. *Inf. Softw. Technol.* 51, 12 (Dec. 2009), 1631–1645. DOI:http://dx.doi.org/10.1016/j.infsof.2009.04.009

[29] Frank D. Macías-Escrivá, Rodolfo Haber, Raul Del Toro, and Vicente Hernandez. 2013. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications* 40, 18 (Dec. 2013), 7267–7279. DOI:http://dx.doi.org/10.1016/j.eswa.2013.07.033

[30] MathWorks. 2015. Stateflow®: Model and simulate decision logic using state machines and flow charts. http://www.mathworks.com/products/stateflow. (2015). (Last accessed: April 20, 2017).

[31] Tom Mens and Pieter Van Gorp. 2006. A Taxonomy of Model Transformation. *Electron. Notes Theor. Comput. Sci.* 152 (March 2006), 125–142. DOI:http://dx.doi.org/10.1016/j.entcs.2005.10.021

[32] Gero Mühl, Ludger Fiege, and Peter Pietzuch. 2006. *Distributed Event-Based Systems.* Springer, Secaucus, NJ, USA. 406 pages.

[33] OASIS. 2007. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (April 2007). (Last accessed: April 20, 2017).

[34] OMG. 2011. Business Process Model and Notation (BPMN) Version 2.0.2. http://www.omg.org/spec/BPMN/2.0.2/PDF. (2011). (Last accessed: April 20, 2017).

[35] OMG. 2011. OMG Unified Modeling Language^{TM} (OMG UML), Superstructure, Version 2.4.1. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF. (2011). (Last accessed: April 20, 2017).

[36] Nayan B. Ruparelia. 2010. Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes* 35, 3 (May 2010), 8–13. DOI:http://dx.doi.org/10.1145/1764810.1764814

[37] August-Wilhelm Scheer and Markus Nüttgens. 2000. ARIS Architecture and Reference Models for Business Process Management. In *Business Process Management, Models, Techniques, and Empirical Studies*. Springer, London, UK, 376–389.

[38] Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Kwiatkowska, John Mcdermid, and Richard Paige. 2012. Large-scale Complex IT Systems. *Commun. ACM* 55, 7 (July 2012), 71–77. DOI:http://dx.doi.org/10.1145/2209249.2209268

[39] George Spanoudakis and Andrea Zisman. 2001. *Handbook of Software Engineering and Knowledge Engineering*. World Scientific, Singapore, Chapter Inconsistency management in software engineering: Survey and open research issues, 329–380.

[40] Thomas Stahl and Markus Völter. 2006. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, Hoboken, NJ, USA.

[41] Muhammad Usman, Aamer Nadeem, Tai-hoon Kim, and Eun-suk Cho. 2008. A Survey of Consistency Checking Techniques for UML Models. In *International Conference on Advanced Software Engineering and Its Applications (ASEA)*. IEEE, Hainan Island, 57–62. DOI:http://dx.doi.org/10.1109/ASEA.2008.40

[42] Wil M. P. van der Aalst. 2013. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering* 2013 (2013), 1–37. DOI:http://dx.doi.org/10.1155/2013/507984

[43] Ragnhild Van der Straeten. 2005. *Inconsistency Management in Model-Driven Engineering*. Ph.D. Dissertation. Vrije University Brussel, Dept. of Computer Science.

[44] Axel van Lamsweerde, Emmanual Letier, and Robert Darimont. 1998. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Trans. Softw. Eng.* 24, 11 (Nov. 1998), 908–926. DOI:http://dx.doi.org/10.1109/32.730542

[45] He Zhang, Muhammad Ali-Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Information and Software Technology* 53, 6 (2011), 625–637. DOI:http://dx.doi.org/10.1016/j.infsof.2010.12.010

A OVERVIEW OF SELECTED PRIMARY STUDIES

Table 16. Overview of 96 Selected Primary Studies

SID	Studied Artifacts	Semantic Domains	Checking Techniques	Studied Domains
S1	Statechart, Activity Diagram (fUML)	Process Algebra (CSP)	Model Checking	fUML
S2	Sequence Diagram, Statechart	State Transition (Transition Matrix)	Specialized Algorithm (Transition Set/Matrix, Super State Analysis)	UML
S3	Statechart	State Transition (Z)	Theorem Proving	UML
S4	Disjunctive Modal Transition Systems (DMTS)	Temporal Logic (LTL), State Transition	Model Checking+Specialized Algorithm (synthesis)	General
S5	Modal Transition System	State Transition, Quantified Boolean formulae (QBF)	Specialized Algorithm	General
S6	Statechart, Sequence Diagram	Petri Net (Generalized Stochastic PNs)	Specialized Algorithm (Analysable PNs)	UML
S7	Process Model	Guard-Action-Trigger (based on Event-Condition-Action)	Specialized Algorithm (Guard-Action-Trigger)	BPM+SOA
S8	Process Model (BPEL)	Process Algebra (D-LOTOS), Durational Action Timed Automata (DATA)	Model Checking	BPM+SOA
S9	Message Sequence Chart (MSC)	State Transition (Timed Message-Passing Automata)	Model Checking	UML
S10	Object Behavior Logic Models	State Transition (OBL)	Logical Inference	Object-Oriented Design
S11	Modal Transition Systems	Temporal Logic (CTL), State Transition	Model Checking, Specialized Algorithm (Repair)	General
S12	SCR Behavioral Requirements, Control-flow based PDL	State Transition, Graph (DFG)	Specialized Algorithm (Dataflow analysis)	SCR/PDL
S13	SDL, Message Sequence Chart (MSC)	Temporal Logic (LTL), State Transition (Finite State Automata)	Model Checking (SPIN)	Distributed Systems
S14	Service Behavioral Interface (STS)	State Transition (Symbolic Transition System STS)	Model Checking (Maude)	SOC/SOA
S15	Statechart, Sequence Diagram (UML-RT)	Process Algebra (CSP)	Model Checking	Embedded/Real-time
S16	Activity Diagram, Visual Contract	State Transition, Graph	Model Checking	BPM+SOA
S17	Activity Diagram	Typed Graph	Specialized Algorithm (Graph Transformation)	UML
S18	Activity Diagram	Temporal Logic, State Transition	Model Checking	UML
S19	Workflow	Ontology, Description Logic (DL)	Ontology Logic Reasoning	Workflow
S20	Activity Diagram (UML-MARTE)	State Transition (Time Transition System TTS)	Specialized Algorithm	Real-time
S21	Sequence Diagram, Statechart (UML/SPT)	Temporal Logic (CTL), Timed Automata	Specialized Algorithm (Schedulability Analysis)	Embedded/Real-time
S22	Statechart	Temporal Logic (CTL), State Transition	Model Checking (AGAVE)	UML
S23	Statechart	Temporal Logic (ACTL), Automata (Hierarchical Automata)	Model Checking (JACK)	UML
S24	Statechart, Sequence Diagram	State Transition (State Machine)	Specialized Algorithm	UML
S25	Sequence Diagram, Use Case	Modal Sequence Diagram (based on Harel's LSC)	Specialized Algorithm (Playout + Synthesis)	UML
S26	Process Model (BPMN)	Description Logic, GRL (i*+NFR)	Specialized Algorithm	BPM
S27	Live Sequence Chart (LSC)	Global System Automaton	Specialized Algorithm	UML

(Continued on next page)

Table 16. Overview of 96 Selected Primary Studies

SID	Studied Artifacts	Semantic Domains	Checking Techniques	Studied Domains
S28	Use Case, Activity Diagram, Collaboration Diagram	State Transition (Typed Graph)	Specialized Algorithm (Graph Transformation)	UML
S29	Statechart	Process Algebra (CSP)	Model Checking (FDR)	UML
S30	Sequence Diagram, Interface Automaton	State Transition (Interface Automata)	Specialized Algorithm (Existential Consistency)	Embedded/Real-time
S31	Sequence Diagram w. Timing Constraints	State Transition (Real-time Interface Automata RIA)	Specialized Algorithm	Embedded/Real-time
S32	Statechart, Sequence Diagram	Temporal Logic (LTL), State Transition	Model Checking	Software Architecture
S33	Statechart, LTS	State Transition (Labeled Transition System)	Model Checking	UML
S34	Sequence Diagram, Activity Diagram	Temporal Logic of Actions (cTLA)	Model Checking	UML
S35	Statechart (Timed Resource-Oriented Statechart TRoS)	Process Algebra (ACSR, for Real-time)	Specialized Algorithm	Embedded/Real-time
S36	Statechart, Sequence Diagram	Timed Automata	Model Checking	Real-time
S37	Sequence Diagram	State Transition (csTS, Modal TS)	Specialized Algorithm (synthesis)	UML
S38	Statechart, Sequence Diagram (UML-RT)	State Transition	Specialized Algorithm	Embedded/Real-time
S39	Process Model (BPMN)	Temporal Logic, Petri Nets	Model Checking (GROOVE)	BPMN
S40	Statechart	Process Algebra (Pi-Calculus)	Specialized Algorithm (Weak Bisimulation)	UML
S41	Statechart, Sequence Diagram	Process Algebra (Pi-Calculus)	Specialized Algorithm (Open Bisimulation)	UML
S42	Sequence, Collaboration Diagram	Real-time Action Logic (RAL)	Logical Inference	Real-time
S43	Modal Transition System	State Transition	Specialized Algorithm	General
S44	Label Transition System, I/O FSM	State Transition, Modal Logic	Model Checking + Specialized Algorithm (Bit Vectors)	UML
S45	Process Model, Domain Ontology	Kripke Structure, Description Logic (DL)	Model Checking	BPM
S46	Statechart	State Transition (Automata)	Model Checking	UML
S47	Sequence Diagram, Statechart	Temporal Logic, State Transition (SMV Language)	Model Checking	BPM+SOA
S48	Statechart	State Transition, Operational Semantics	Specialized Algorithm (Assertion Checking)	UML
S49	Sequence Diagram	Trace Semantics	Specialized Algorithm	UML
S50	Statechart	Temporal Logic, State Transition (Kripke Structure)	Model Checking	Embedded/Real-time
S51	Abstract, Executable Process Model (BPEL)	Petri Net, Communication Graph	Specialized Algorithm (Simulation)	BPM
S52	Stateflow (Simulink)	Process Algebra (Circus:Z, CSP, Guarded Commands)	Specialized Algorithm (Simulation)	Simulink
S53	Activity Diagram	State Transition	Model Checking (NuSMV)	UML
S54	Process Model	Temporal Logic (CTL), State Transition (LTS)	Model Checking	BPM
S55	Statechart	State Transition	Specialized Algorithm (Matching+Merging, Bisimulation)	UML
S56	Statechart	State Transition (B)	Theorem Proving	UML
S57	Statechart	Temporal Logic, State Transition (PROMELA)	Model Checking	Adaptive System
S58	Statechart, Class Diagram	Process Algebra (CSP)	Model Checking (FDR)	UML
S59	Statechart	Process Algebra (CSP)	Model Checking	UML
S60	Activity Diagram, Statechart	State Transition	Specialized Algorithm (Rules)	BPM
S61	Statechart	First Order Logic	Logical Inference (CrocoPat)	UML

(Continued on next page)

Table 16. Overview of 96 Selected Primary Studies

SID	Studied Artifacts	Semantic Domains	Checking Techniques	Studied Domains
S62	Modal Transition System	Process Algebra (Modal Pi-Calculus)	Specialized Algorithm	Modal Transition Systems
S63	Statechart, Sequence Diagram	State Transition	Model Checking	UML
S64	Object Behavior Diagram	Petri Net (Object Behavior Diagram)	Specialized Algorithm	BPM
S65	Statechart	State Transition (Symbolic Transition System STS)	Specialized Algorithm	UML
S66	Statechart (Invariant Statechart)	State Transition	Specialized Algorithm (State Invariant)	UML
S67	Sequence Diagram, Statechart	First Order Logic	Logical Inference, Theorem Proving	UML
S68	Use Case, Activity Diagram, Statechart, Sequence Diagram	Petri Net (Colored PN)	Specialized Algorithm (PN consistency)	UML
S69	Statechart	State Transition (Timed Automata)	Specialized Algorithm (Reachability Analysis+Constraint Solving)	Real-time
S70	Stateflow (Simulink)	Graph (Binary Decision Diagram BDD)	Model Checking (Salsa)	Simulink
S71	Process Model	Petri Net	Specialized Algorithm	BPM
S72	Process Model (BPEL)	Petri Nets, BPEL Program Dependence Graph (BPD)	Specialized Algorithm	BPM
S73	Statechart	Petri Net (Object Behavior Diagram)	Specialized Algorithm (Observation Consistency)	UML
S74	Activity Diagram	Petri Net (Instantialble PN)	Specialized Algorithm	UML
S75	Use Case, Statechart	Graph	Specialized Algorithm (Graph Search)	UML
S76	Process Model (BPEL), Service Behavior (Message)	Petri Net	Specialized Algorithm (Conformance Checking)	SOC/SOA
S77	Use Case, Sequence Diagram, Statechart	State Transition (Petri Net)	Specialized Algorithm (Systhesis, Invariant)	UML
S78	Statechart, Sequence Diagram	State Transition, Traces	Model Checking	UML
S79	Workflow	Formal Workflow, First Order Logic	Specialized Algorithm (Rules)	Workflow
S80	Process Model (BPEL 2.0)	Process Algebra (Pi-Calculus)	Specialized Algorithm (Bisimulation)	BPM+SOA
S81	Process Model	Graph	Specialized Algorithm (Matching, Edit Distance)	BPM
S82	Process Model	Petri Net	Specialized Algorithm (Causal Behavioral Profiles)	BPM
S83	Process Model (BPMN,BPEL,EPC)	Petri Net	Specialized Algorithm (Behavioral Profiles)	BPM
S84	Statechart, Sequence Diagram	Temporal Logic, State Transition (Extended Sequence Diagram)	Model Checking	SOC/SOA
S85	Activity Diagram	Process Algebra (CSP)	Model Checking (FDR)	UML
S86	Statechart, Sequence Diagram	Petri Net (Extended Colored PN)	Specialized Algorithm (Coverability Checking)	UML
S87	Statechart	Process Algebra (CSP), Abstract Machine Notation (B)	Model Checking	UML
S88	WS-CDL, BPEL	Process Algebra (CSP)	Model Checking	BPM+SOA
S89	Process Model (ebXML BPSS, BPEL)	Process Algebra (Pi-Calculus)	Specialized Algorithm (Trace Refinement CSP)	BPM+SOA
S90	Statechart, Sequence Diagram	Automata (Split Automata)	Model Checking	UML
S91	Statechart, Message Sequence Chart (MSC)	Automata	Model Checking	UML
S92	Statechart, Sequence Diagram	Propositional Logic	Model checking (SAT)	UML

(Continued on next page)

Table 16. Overview of 96 Selected Primary Studies

SID	Studied Artifacts	Semantic Domains	Checking Techniques	Studied Domains
S93	Statechart	Temporal Logic (LTL), Automata (Extended Hierarchical Automata)	Model Checking	UML
S94	Statechart	Temporal Logic (CTL), State Transition (Kripke Structure)	Model Checking	UML
S95	Modal Transition System	State Transition	Specialized Algorithm	Modal Transition Systems
S96	Statechart	Process Algebra (CSP)	Model Checking	UML

B SELECTED PRIMARY STUDIES

Table 17. List of Selected Primary Studies

ID	Full Reference
S1	Abdelhalim, I. , Schneider, S. , and Treharne, H. **Towards a Practical Approach to Check UML/fUML Models Consistency using CSP.** *13th International Conference on Formal Methods and Software Engineering (ICFEM)*, Durham, UK, 2011, Springer, 33–48.
S2	Alanazi, M. N. and Gustafson, D. A. **Super State Analysis for UML State Diagrams.** *WRI World Congress on Computer Science and Information Engineering (CSIE)- Volume 07*, 2009, IEEE, 560–565.
S3	Amálio, N. , Stepney, S. , and Polack, F. **Formal Proof from UML Models.** *6th International Conference on Formal Engineering Methods (ICFEM)*, Seattle, WA, USA, 2004, Springer, 418–433.
S4	Beneš, N. , Çerná, I. , and Křetínský, J. **Modal Transition Systems: Composition and LTL Model Checking.** *9th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, Taipei, Taiwan, 2011, Springer, 228–242.
S5	Beneš, N. , Křetínský, J. , and Larsen, K. G. **Refinement Checking on Parametric Modal Transition Systems.** *Acta Informatica*, Vol. 52, No. 2-3 (2015), 269–297.
S6	Bernardi, S. , Donatelli, S. , and Merseguer, J. **From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models.** *Third International Workshop on Software and Performance (WOSP)*, 2002, ACM, 35–45.
S7	Bhuiyan, J. , Nepal, S. , and Zic, J. **Checking Conformance between Business Processes and Web Service Contract in Service Oriented Applications.** *Australian Software Engineering Conference (ASWEC)*, 2006, IEEE, 80–89.
S8	Chama, I. E. , Belala, N. , and Saidouni, D.-E. **FMEBP: A Formal Modeling Environment of Business Process.** *20th International Conference on Information and Software Technologies (ICIST)*, Druskininkai, Lithuania, 2014, Springer, 211-223.
S9	Chandrasekaran, P. and Mukund, M. **Matching Scenarios with Timing Constraints.** *4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, Paris, France, 2006, Springer, 98–112.
S10	Chang, K. , Kung, D. C. , and Hsia, P. **OBL: A Formal Deduction Method for Object-Oriented Systems.** *23rd International Computer Software and Applications Conference (COMPSAC)*, 1999, IEEE, 450–455.
S11	Chatzieleftheriou, G. , Bonakdarpour, B. , and Smolka, S. A. **Abstract Model Repair.** *4th International Symposium on NASA Formal Methods (NFM)*, Norfolk, VA, USA, 2012, Springer, 341–355.
	(Continued on next page)

Table 17. List of Selected Primary Studies

ID	Full Reference
S12	Chechik, M. and Gannon, J. **Automatic Analysis of Consistency between Requirements and Designs.** *IEEE Trans. Softw. Eng.*, Vol. 27, No. 7 (2001), 651–672.
S13	D'Souza, D. and Mukund, M. **Checking Consistency of SDL+MSC Specifications.** *10th International SPIN Workshop on Model Checking Software*, Portland, OR, USA, 2003, Springer, 151–166.
S14	Durán, F. , Ouederni, M. , and Salaün, G. **Checking Protocol Compatibility using Maude.** *Electronic Notes in Theoretical Computer Science*, Vol. 255 (2009), 65–81.
S15	Engels, G. , Küster, J. M. , and Heckel, R. **A methodology for specifying and analyzing consistency of object-oriented behavioral models.** *8th European Software Engineering Conference, Held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, Vienna, Austria, 2001, ACM, 186–195.
S16	Engels, G. , Güldali, B. , and Soltenborn, C. **Assuring Consistency of Business Process Models and Web Services Using Visual Contracts.** *Applications of Graph Transformations with Industrial Relevance*, 17–31, (2008).
S17	Ermel, C. , Gall, J. , and Lambers, L. **Modeling with plausibility checking: inspecting favorable and critical signs for consistency between control flow and functional behavior.** *14th International Conference on Fundamental Approaches to Software Engineering (FASE): part of the joint European Conferences on Theory and Practice of Software (ETAPS)*, Saarbrücken, Germany, 2011, Springer, 156–170.
S18	Eshuis, R. and Wieringa, R. **Tool support for verifying UML activity diagrams.** *IEEE Transactions on Software Engineering (TSE)*, Vol. 30, No. 7 (2004), 437-447.
S19	Fan, S. , Zhao, J. L. , and Dou, W. **A framework for transformation from conceptual to logical workflow models.** *Decis. Support Syst.*, Vol. 54, No. 1 (2012), 781–794.
S20	Ge, N. , Pantel, M. , and Crégut, X. **Time properties dedicated transformation from UML-MARTE activity to time transition system.** *SIGSOFT Softw. Eng. Notes*, Vol. 37, No. 4 (2012), 1.
S21	Gherbi, A. and Khendek, F. **Consistency of UML/SPT models.** *13th international SDL Forum conference on Design for dependable systems (SDL)*, Paris, France, 2007, Springer, 203–224.
S22	Ghezzi, C. , Menghi, C. , and Molzam Sharifloo, A. **On Requirement Verification for Evolving Statecharts Specifications.** *Requirements Engineering*, Vol. 19, No. 3 (2014), 231–255.
S23	Gnesi, S. , Latella, D. , and Massink, M. **Model checking UML Statechart diagrams using JACK.** *4th IEEE International Symposium on High-Assurance Systems Engineering*, 1999, , 46-55.
S24	Graaf, B. and van Deursen, A. **Model-Driven Consistency Checking of Behavioural Specifications.** *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, 2007, IEEE, 115–126.
S25	Greenyer, J. and Frieben, J. **Consistency Checking Scenario-based Specifications of Dynamic Systems by Combining Simulation and Synthesis.** *Fourth Workshop on Behaviour Modelling - Foundations and Applications (BM-FA)*, Kgs. Lyngby, Denmark, 2012, ACM, 2:1–2:9.
S26	Grøner, G. , Asadi, M. , and Mohabbati, B. **Validation of user intentions in process orchestration and choreography.** *Information Systems*, Vol. 43, No. 0 (2014), 83 - 99.
S27	Harel, D. and Kugler, H. **Synthesizing State-Based Object Systems from LSC Specifications.** *International Journal of Foundations of Computer Science*, Vol. 13, No. 01 (2002), 5-51.
S28	Hausmann, J. H. , Heckel, R. , and Taentzer, G. **Detection of conflicting functional requirements in a use case-driven approach: a static analysis technique based on graph transformation.** *24th International Conference on Software Engineering (ICSE)*, Orlando, Florida, 2002, ACM, 105–115.

(Continued on next page)

Table 17. List of Selected Primary Studies

ID	Full Reference
S29	Heckel, R. and Küster, J. M. **Behavioral Constraints for Visual Models.** *Electronic Notes in Theoretical Computer Science*, Vol. 50, No. 3 (2001), 257–265.
S30	Hu, J. , Yu, X. , Wang, L. , et al.**Scenario-Based Specifications Verification for Component-Based Embedded Software Designs.** *International Conference on Parallel Processing Workshops (ICPPW)*, 2005, IEEE, 240–247.
S31	Hu, J. , Yu, X. , and Zhang, Y. **Checking Component-Based Embedded Software Designs for Scenario-Based Timing Specifications.** *International Conference Embedded and Ubiquitous Computing (EUC)*, Nagasaki, Japan, 2005, Springer, 395–404.
S32	Inverardi, P. , Muccini, H. , and Pelliccione, P. **Automated Check of Architectural Models Consistency Using SPIN.** *16th IEEE International Conference on Automated Software Engineering (ASE)*, 2001, IEEE, 346–349.
S33	Junwei, D. , Zhongwei, X. , and Meng, M.**Verification of Scenario-Based Safety Requirement Specification on Components Composition.** *2008 International Conference on Computer Science and Software Engineering (CSSE)- Volume 02*, 2008, IEEE, 686–689.
S34	Kaliappan, P. and Koenig, H. **An Approach to Synchronize UML-Based Design Components for Model-Driven Protocol Development.** *IEEE 34th Software Engineering Workshop (SEW)*, Limerick, Ireland, 2011, IEEE, 27–35.
S35	Kim, J. , Kang, I. , and Choi, J.-Y. **Formal synthesis of application and platform behaviors of embedded software systems.** *Software & Systems Modeling*, Vol. 14, No. 2 (2015), 839-859.
S36	Knapp, A. , Merz, S. , and Rauh, C. **Model Checking - Timed UML State Machines and Collaborations.** *7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2 (FTRTFT)*, 2002, Springer, 395–416.
S37	Krka, I. and Medvidovic, N. **Component-Aware Triggered Scenarios.** *IEEE/IFIP Conference on Software Architecture (WICSA)*, Sydney, NSW, Australia, 2014, IEEE, 129–138.
S38	Küster, J. M. and Stroop, J. **Consistent Design of Embedded Real-Time Systems with UML-RT.** *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, Magdeburg, Germany, 2001, IEEE, 31-40.
S39	Kwantes, P. M. , Gorp, P. V. , and Kleijn, J. **Towards Compliance Verification Between Global and Local Process Models.** *8th International Conference on Graph Transformation ICGT, Held as Part of STAF 2015*, L'Aquila, Italy, 2015, Springer, 221–236.
S40	Lam, V. S. W. and Padget, J. **Consistency Checking of Statechart Diagrams of a Class Hierarchy.** *19th European conference on Object-Oriented Programming (ECOOP)*, Glasgow, UK, 2005, Springer, 412–427.
S41	Lam, V. S. W. and Padget, J. **Consistency checking of sequence diagrams and statechart diagrams using the π-calculus.** *5th International Conference on Integrated Formal Methods (IFM)*, Eindhoven, The Netherlands, 2005, Springer, 347–365.
S42	Lano, K. **Formal Specification using Interaction Diagrams.** *Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, 2007, IEEE, 293–304.
S43	Larsen, K. G. , Nyman, U. , and Wąsowski, A. **On Modal Refinement and Consistency.** *18th International Conference on Concurrency Theory (CONCUR)*, Lisbon, Portugal, 2007, Springer, 105–119.
S44	Lee, J.-D. , Jung, J.-I. , and Lee, J.-H. **Verification and conformance test generation of communication protocol for railway signaling systems.** *Comput. Stand. Interfaces*, Vol. 29, No. 2 (2007), 143–151.
S45	Letia, I. A. and Goron, A. **Model checking as support for inspecting compliance to rules in flexible processes.** *Journal of Visual Languages & Computing*, Vol. 28, No. C (2015), 100 - 121.

(Continued on next page)

Table 17. List of Selected Primary Studies

ID	Full Reference
S46	Diego, L. , Istvan, M. , and Mieke, M. **Automatic verification of UML statechart diagrams using the SPIN model-checker.** *Formal Aspects of Computing*, Vol. 11, No. 637-664 (1999), Springer.
S47	Li, B. , Zhou, Y. , and Pang, J. **Model-Driven Automatic Generation of Verified BPEL Code for Web Service Composition.** *2009 16th Asia-Pacific Software Engineering Conference (APSEC)*, 2009, IEEE, 355–362.
S48	Liu, S. , Liu, Y. , and Sun, J. **USMMC: A Self-contained Model Checker for UML State Machines.** *9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, Saint Petersburg, Russia, 2013, ACM, 623–626.
S49	Lu, L. and Kim, D.-K. **Required Behavior of Sequence Diagrams: Semantics and Conformance.** *ACM Trans. Softw. Eng. Methodol.*, Vol. 23, No. 2 (2014), 15:1–15:28.
S50	Majzik, I. , Micskei, Z. , and Pintér, G. **Development of model based tools to support the design of railway control applications.** *26th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, Nuremberg, Germany, 2007, Springer, 430–435.
S51	Martens, A. **Consistency between Executable and Abstract Processes.** *IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE)*, 2005, IEEE, 60–67.
S52	Miyazawa, A. and Cavalcanti, A. **Refinement-oriented Models of Stateflow Charts.** *Science of Computer Programming*, Vol. 77, No. 10-11 (2012), 1151–1177.
S53	Muram, F. U. , Tran, H. , and Zdun, U. **Automated Mapping of UML Activity Diagrams to Formal Specifications for Supporting Containment Checking.** *11th International Workshop on Formal Engineering Approaches to Software Components and Architectures (FESCA)*, Grenoble, France, 2014, arXiv.org, 1–16.
S54	Nagel, B. , Gerth, C. , and Engels, G. **Ensuring Consistency Among Business Goals and Business Process Models.** *17th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2013, IEEE, 17–26.
S55	Nejati, S. , Sabetzadeh, M. , and Chechik, M. **Matching and Merging of Statecharts Specifications.** *29th International Conference on Software Engineering (ICSE)*, Minneapolis, MN, 2007, IEEE, 54–64.
S56	Ossami, D. D. O. , Jacquot, J.-P. , and Souquières, J. **Consistency in UML and B Multi-view Specifications.** *5th International Conference on Integrated Formal Methods (IFM)*, Eindhoven, The Netherlands, 2005, Springer, 386–405.
S57	Ramirez, A. J. and Cheng, B. H. C. **Verifying and Analyzing Adaptive Logic through UML State Models.** *International Conference on Software Testing, Verification, and Validation (ICST)*, 2008, IEEE, 529–532.
S58	Rasch, H. and Wehrheim, H. **Checking Consistency in UML Diagrams: Classes and State Machines.** *6th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, Paris, France, 2003, Springer, 229–243.
S59	Roscoe, A. W. and Wu, Z. **Verifying statemate statecharts using CSP and FDR.** *8th International Conference on Formal Methods and Software Engineering (ICFEM)*, Macao, China, 2006, Springer, 324–341.
S60	Ryndina, K. , Küster, J. M. , and Gall, H. **Consistency of business process models and object life cycles.** *International Conference on Models in Software Engineering (MoDELS)*, Genoa, Italy, 2006, Springer, 80–90.
S61	Sabetzadeh, M. , Nejati, S. , and Easterbrook, S. **Global Consistency Checking of Distributed Models with TReMer+.** *30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, 2008, ACM, 815–818.
	(Continued on next page)

Table 17. List of Selected Primary Studies

ID	Full Reference
S62	Sassolas, M. , Chechik, M. , and Uchitel, S. **Exploring inconsistencies between modal transition systems.** *Software and Systems Modeling (SoSyM)*, Vol. 10, No. 1 (2010), 117–142.
S63	Schäfer, T. , Knapp, A. , and Merz, S. **Model Checking UML State Machines and Collaborations.** *Electronic Notes in Theoretical Computer Science*, Vol. 55, No. 3 (2001), 357 - 369.
S64	Schrefl, M. and Stumptner, M. **Behavior-consistent specialization of object life cycles.** *ACM Trans. Softw. Eng. Methodol.*, Vol. 11, No. 1 (2002), 92–148.
S65	Schwarzl, C. and Peischl, B. **Static- and Dynamic Consistency Analysis of UML State Chart Models.** *13th International Conference on Model Driven Engineering Languages and Systems (MODELS) - Part I*, Oslo, Norway, 2010, Springer, 151–165.
S66	Sekerinski, E. **Verifying Statecharts with State Invariants.** *13th IEEE International Conference on on Engineering of Complex Computer Systems (ICECCS)*, 2008, IEEE, 7–14.
S67	Shan, L. and Zhu, H. **A Formal Descriptive Semantics of UML.** *10th International Conference on Formal Methods and Software Engineering (ICFEM)*, Kitakyushu-City, Japan, 2008, Springer, 375–396.
S68	Shinkawa, Y. **Inter-Model Consistency in UML Based on CPN Formalism.** *13th Asia Pacific Software Engineering Conference (APSEC)*, 2006, IEEE, 411–418.
S69	Shu, G. , Li, C. , Wang, Q. , et al.**Validating Objected-Oriented Prototype of Real-Time Systems with Timed Automata.** *13th IEEE International Workshop on Rapid System Prototyping (RSP'02)*, 2002, IEEE, 99–106.
S70	Sims, S. , Cleaveland, R. , and Butts, K. **Automated Validation of Software Models.** *16th IEEE International Conference on Automated Software Engineering (ASE)*, San Diego, California, 2001, IEEE, 91–96.
S71	Smirnov, S. , Farahani, A. Z. , and Weske, M. **State propagation in abstracted business processes.** *9th International Conference on Service-Oriented Computing (ICSOC)*, Paphos, Cyprus, 2011, Springer, 16–31.
S72	Song, W. , Zhang, W. , and Zhang, G. **Quantifying Consistency Between Conceptual and Executable Business Processes.** *IEEE International Conference on Services Computing (SCC)*, 2013, IEEE, 9–16.
S73	Stumptner, M. and Schrefl, M. **Behavior consistent inheritance in UML.** *19th International Conference on Conceptual Modeling (ER)*, Salt Lake City, Utah, USA, 2000, Springer, 527–542.
S74	Thierry-Mieg, Y. and Hillah, L.-M. **UML behavioral consistency checking using instantiable Petri nets.** *Innovations in Systems and Software Engineering (ISSE)*, Vol. 4, No. 3 (2008), 293-300.
S75	Truong, N.-T. , Tran, T.-M.-T. , and To, V.-K. **Checking the Consistency between UCM and PSM Using a Graph-Based Method.** *First Asian Conference on Intelligent Information and Database Systems (ACIIDS)*, 2009, IEEE, 190–195.
S76	van der Aalst, W. M. P. , Dumas, M. , and Ouyang, C. **Conformance Checking of Service Behavior.** *ACM Transactions on Internet Technology (TOIT)*, Vol. 8, No. 3 (2008), 1–30.
S77	van Hee, K. , Sidorova, N. , and Somers, L. **Consistency in model integration.** *Data & Knowledge Engineering*, Vol. 56, No. 1 (2006), 4–22.
S78	Wang, H. , Feng, T. , and Zhang, J. **Consistency check between behaviour models.** *IEEE International Symposium on Communications and Information Technology (ISCIT)*, 2005, IEEE, 486-489.
S79	Wang, H. J. and Zhao, J. L. **Constraint-centric Workflow Change Analytics.** *Decis. Support Syst.*, Vol. 51, No. 3 (2011), 562–575.

(Continued on next page)

Table 17. List of Selected Primary Studies

ID	Full Reference
S80	Weidlich, M. , Decker, G. , and Weske, M. **Efficient Analysis of BPEL 2.0 Processes Using p-Calculus.** *Second IEEE Asia-Pacific Service Computing Conference (APSCC)*, 2007, IEEE, 266–274.
S81	Weidlich, M. , Dijkman, R. , and Mendling, J. **The ICoP Framework: Identification of Correspondences Between Process Models.** *22nd International Conference on Advanced Information Systems Engineering (CAiSE)*, Hammamet, Tunisia, 2010, Springer, 483–498.
S82	Weidlich, M. , Polyvyanyy, A. , and Mendling, J. **Causal Behavioural Profiles - Efficient Computation, Applications, and Evaluation.** *Fundam. Inf.*, Vol. 113, No. 3-4 (2011), 399–435.
S83	Weidlich, M. , Mendling, J. , and Weske, M. **Efficient Consistency Measurement Based on Behavioral Profiles of Process Models.** *IEEE Trans. Softw. Eng.*, Vol. 37, No. 3 (2011), 410–429.
S84	Xie, Y. , Du, D. , and Liu, J. **Towards the Verification of Services Collaboration.** *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2009, IEEE, 428–433.
S85	Xu, D. , Miao, H. , and Philbert, N. **Model Checking UML Activity Diagrams in FDR.** *8th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, Shanghai, China, 2009, IEEE, 1035 - 1040.
S86	Yao, S. and Shatz, S. M. **Consistency Checking of UML Dynamic Models Based on Petri Net Techniques.** *15th International Conference on Computing (CIC)*, 2006, IEEE, 289–297.
S87	Yeung, W. L. **Checking Consistency between UML Class and State Models Based on CSP and B.** *Journal of Universal Computer Science (JUSC)*, Vol. 10, No. 11 (2004), 1540–1559.
S88	Yeung, W. L. **CSP-Based Verification for Web Service Orchestration and Choreography.** *Simulation*, Vol. 83, No. 1 (2007), 65–74.
S89	Yeung, W. L. **A Formal Basis for Cross-Checking ebXML BPSS Choreography and Web Service Orchestration.** *IEEE Asia-Pacific Services Computing Conference (APSCC)*, 2008, IEEE, 524–529.
S90	Zhao, X. , Long, Q. , and Qiu, Z. **Model Checking Dynamic UML Consistency.** *8th International Conference on Formal Methods and Software Engineering (ICFEM)*, Macao, China, 2006, Springer, 440–459.
S91	Knapp, A. and Wuttke, J. **Model Checking of UML 2.0 Interactions.** *Proceedings of the 2006 International Conference on Models in Software Engineering (MoDELS)*, Genoa, Italy, 2006, Springer, 42–51.
S92	Kaufmann, P. , Kronegger, M. , Pfandler, A. , Seidl, M. , and Widl, M. **A SAT-Based Debugging Tool for State Machines and Sequence Diagrams.** *7th International Conference on Software Language Engineering (SLE)*, Västerås, Sweden, 2014, Springer, 21–40.
S93	Dong, W. , Wang, J. , Qi, X. , and Qi, Z. **Model checking UML statecharts.** *8th Asia-Pacific Software Engineering Conference (APSEC)*, Macao, China, 2001, IEEE, 363 - 370.
S94	Zhao, Q. , and Krogh, B. H. **Formal Verification of Statecharts Using Finite-State Model Checkers.** *EEE Transactions on Control Systems Technology*, Vol. 14, 2006, IEEE, 943–950.
S95	Fischbein, D. and Uchitel, S. **On Correct and Complete Strong Merging of Partial Behaviour Models.** *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, New York, NY, USA, 2008, ACM, 297–307.
S96	Zhang, S. J. and Liu, Y. **An Automatic Approach to Model Checking UML State Machines.** *4th International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C)*,Washington, DC, USA, 2010, IEEE, 1–6.

C DATA EXTRACTION: DATA ITEMS AND ENCODING

Table 18. Evidence of Timing Support

Level	Description
1	Not considered
2	Implicit using of underlying timing model or rules, for instance, providing a way to represent temporal information using timers, temporal logics, CSP, etc.
3	Explicit timing model and analysis, for instance, by using explicitly timed models or real-time constraints

Table 19. Evidence of Inconsistency Handling (adapted from [39])

Level	Description
1	Not mentioned / Not considered
2	Systematic inconsistency diagnosis
3	Identifying handling actions
4	Evaluating costs and risks
5	Automated action selection and execution

Table 20. Evidence of Automation Support

Level	Description
1	Manual: requiring human interactions or manually specifying rules, constraints
2	Semi-automated: assuming existing input models, partially human interaction
3	Fully automated

Table 21. Evidence of Development Tool Support and Integration

Level	Description
1	Not mentioned / Not considered
2	Proposed/planned integration
3	Fully implemented integration

Table 22. Evidence of Tool Support for Consistency Checking

Level	Description
1	Not mentioned / Not considered
2	Only using existing tools / libraries
3	Prototypes (including using existing tools / libraries)

Table 23. Level of Empirical Evidence (adapted from [1, 23])

Level	Description
1	No evidence.
2	Evidence obtained from demonstration or working out toy examples.
3	Evidence obtained from expert opinions or observations.
4	Evidence obtained from academic studies, e.g., controlled experiments
5	Evidence obtained from industrial studies.
6	Evidence obtained from industrial practice.

Table 24. Type of Study and Evaluation (adapted from [7])

Type	Description
Rigorous analysis (RA)	Rigorous derivation and proof, suited for formal model
Case study (CS)	An empirical inquiry that investigates a contemporary phenomenon within its real-life context; when the boundaries between phenomenon and context are not clearly evident; and in which multiple sources of evidence are used
Discussion (DC)	Provided some qualitative, textual, opinion
Example (EX)	Authors describing an application and provide an example to assist in the description, but the example is "used to validate" or "evaluate" as far as the authors suggest
Experience Report (ER)	The result has been used on real examples, but not in the form of case studies or controlled experiments, the evidence of its use is collected informally or formally
Field study (FS)	Controlled experiment performed in industry settings
Laboratory experiment with human subjects (LH)	Identification of precise relationships between variables in a designed controlled environment using human subjects and quantitative techniques
Laboratory experiment with software subjects (LS)	A laboratory experiment to compare the performance of newly proposed system with other existing systems
Simulation (SI)	Execution of a system with artificial data, using a model of the real word
Not mentioned	*These types of studies have been excluded*

Table 25. Rigor (adapted from [22])

Aspect	**Description**		
	Level=Weak	**Level=Medium**	**Level=Strong**
Context described (C)	There appears to be no description of the context in which the evaluation is performed.	The context in which the study is performed is mentioned or presented in brief but not described to the degree to which a reader can understand and compare it to another context.	The context is described to the degree where a reader can understand and compare it to another context.
Study design described (S)	There appears to be no description of the design of the presented evaluation.	The study design is briefly described, e.g., "ten students did step 1, step 2 and step 3".	The study design is described to the degree where a reader can understand, e.g., the variables measured, the control used, the treatments, the selection/sampling used etc.
Validity discussed (V)	There appears to be no description of any threats to validity of the evaluation.	The validity of the study is mentioned but not described in detail.	The validity of the evaluation is discussed in detail where threats are described and measures to limit them are detailed.