HAL
open science

A Research Landscape on Formal Verification of Software Architecture Descriptions

Camila Araujo, Everton Cavalcante, Thais Batista, Marcel Oliveira, Flavio Oquendo

To cite this version:

Camila Araujo, Everton Cavalcante, Thais Batista, Marcel Oliveira, Flavio Oquendo. A Research Landscape on Formal Verification of Software Architecture Descriptions. IEEE Access, 2019, 7, pp.171752-171764. 10.1109/ACCESS.2019.2953858 . hal-02570179

HAL Id: hal-02570179

https://hal.science/hal-02570179v1

Submitted on 4 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received August 27, 2019, accepted November 8, 2019, date of publication November 15, 2019, date of current version December 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2953858

A Research Landscape on Formal Verification of Software Architecture Descriptions

CAMILA ARAÚJO[1,2], EVERTON CAVALCANTE[2], THAIS BATISTA[2], MARCEL OLIVEIRA[2], AND FLAVIO OQUENDO[3]

[1]State University of Rio Grande do Norte, Natal, Brazil
[2]DIMAp, Federal University of Rio Grande do Norte, Natal 59064-741, Brazil
[3]IRISA-UMR CNRS/Université Bretagne Sud, 56017 Vannes, France

Corresponding author: Camila Araújo (cmlaraujo@gmail.com)

This work was supported in part by the INES 2.0, in part by the FACEPE under Grant APQ-0399-1.03/17, in part by the CAPES under Grant 88887.136410/2017-00, and in part by the CNPq under Grant 465614/2014-0. The work of T. Batista was supported by the CNPq under Grant 308274/2016-4.

ABSTRACT One of the many different purposes of software architecture descriptions is contributing to an early analysis of the architecture with respect to quality attributes. The critical nature of many software systems calls for formal approaches aiming at precisely verifying if their designed architectures can meet important properties such as consistency, completeness, and correctness. In this context, it is worthwhile investigating the role of architecture descriptions to support the formal verification of software architectures to ensure their quality, as well as how such a process happens and is supported by existing languages and verification tools. To evaluate the research landscape on this subject, we have carried out a systematic mapping study in which we collected and analyzed studies available at the literature on formal verification of architecture descriptions. This work contributes with (i) a structured overview and taxonomy of the current state of the art on this topic and (ii) the elicitation of important issues to be addressed in future research.

INDEX TERMS Architecture description, formal verification, property specification, software architectures, systematic mapping.

I. INTRODUCTION

Software architectures play a significant role in the development of software-intensive systems as they contribute to the achievement of both business goals and quality requirements. A way of making software architectures more concrete is through architecture descriptions expressed in some architecture description language (ADL) [1]. These architecture descriptions can be used as models at design time or runtime as well as support software architecture documentation, maintenance, evaluation, and evolution. Classically, ADLs have been classified into three broad categories: (i) formal, i.e., notations with precise (often mathematically-based) syntax and semantics that support automated architectural analysis; (ii) semi-formal, i.e., notations with well-defined syntax, but lack of complete semantics; and (iii) informal, ad-hoc box-and-lines diagrams that cannot be formally analyzed and limit the usefulness of the architecture description [2], [3].

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

One of the major challenges in the design of software-intensive systems is related to their architectural analysis, i.e., the activity of discovering important system properties using architectural models even before its implementation [4]. Performing such an activity as early as possible is essential to avoid possible incorrectness, inconsistencies, and undesirable issues until later phases of the development process when a correction will be surely costly. This is imperative mainly when dealing with the critical nature of many complex systems, whose envisioned architecture must be verified with respect to their correctness and fulfillment of required behavior and properties of interest. For this purpose, formal architecture descriptions are highly desirable as means of better supporting automated architectural analysis, acknowledged as an important activity in software industry [5], [6]. The main advantage of adopting a formal approach is precisely determining if a software system can satisfy properties and constraints related to requirements and check the accuracy and correctness of architectural designs.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

The literature reports the existence of many approaches related to the formal verification of software architectures having their architecture descriptions as one of their main elements and some works have attempted to provide a comprehensive overview of the research landscape on this topic. Tsai and Xu [7] compared formal verification tools applied over software architecture specifications, whereas Dobrica and Nimelä [8] and Babar and Gorton [9] proposed frameworks aimed to compare existing architectural analysis and evaluation approaches, but not specifically focusing on formal methodologies. More recently, Zhang *et al.* [10] performed a valuable literature review on the formal verification of software architectures, but they (i) did not specifically considered architecture descriptions, (ii) focused only on model checking as formal verification technique, and (iii) conducted their study in an ad-hoc way, thus not following a systematic, well-defined procedure. Despite its relevance, this body of work has now almost ten to twenty years and other approaches on the formal software architecture verification have been proposed since then.

Given the importance of formal verification in the software architecture context as means of contributing to ensure quality in the contemporary software systems, an extended, up-to-date literature overview is quite relevant. Such an overview can enable both researchers and practitioners to critically reflect on the current state of the art, identify important issues to drive future research, and understand how existing and future research can be suitable for technical, business, and organizational needs. To reach this goal, we have performed a *systematic mapping study* (or shortly *systematic mapping*) on approaches for the formal verification of architecture descriptions. A systematic mapping is well-established form of secondary study aimed to obtain a comprehensive overview of a research topic, identify research gaps, and collect evidences to commission future research through a systematic, rigorous procedure of collection, selection, and analysis of available primary studies [11], [12].

The goal of this systematic mapping is threefold: (i) to provide an overview of the research on the formal verification of software architecture descriptions; (ii) to understand features of languages used to describe software architectures and properties of interest to be formally verified; and (iii) to understand how software architecture formal verification has been conducted in this context. For this purpose, studies published in journals, conferences, and workshops were collected from electronic databases, systematically analyzed, and selected as best fitting a set of inclusion and exclusion criteria. In this paper, we report (i) the main findings of the performed systematic mapping, (ii) a structured overview and taxonomy of the current state of the art on this topic, and (iii) a discussion on important issues to be addressed by future work.

The remainder of this paper is structured as follows. Section II provides the background of this work. Section III presents the methodology adopted in this work in terms of the research questions to be answered and study search and selection strategies, as well as details on how the relevant primary studies were selected. Section IV provides a synthesis resulting from the analysis of the studies as answers to the research questions. Section V presents a taxonomy for the formal verification of software architecture descriptions. Section VI discusses some important issues that can drive further research. Section VII raises some threats to empirical validity. Finally, Section VIII summarizes the main findings of this study along with some concluding remarks.

II. BACKGROUND

ADLs emerged since the 1990s mainly resulting from the research devoted to the problem of providing more precise ways to characterize the structure and behavior of software architectures as well as derive properties on these structures. Malavolta *et al.* [5] conducted a survey on the use of ADLs in industry to understand the practitioners' point of view regarding strengths, limitations, and requirements of current languages. They observed that ADLs should support (i) the definition of functional and non-functional properties, (ii) formal semantics for improving precision and allowing for automated analysis, (iii) both graphical and textual representations for easing the communication among stakeholders and users of architecture descriptions, and (iv) features such as simplicity and intuitiveness.

Recent studies on the use of ADLs by practitioners have confirmed and complemented some findings brought by Malavolta *et al.* [5]. Ozkaya [13] analyzed more than one hundred ADLs with respect to a set of requirements considered as crucial for practitioners. Some important findings provided by his study are: (i) a small part of existing ADLs considers non-functional requirements despite their relevance to ensure quality in software systems; (ii) formal semantics are supported by almost half of the existing notations, especially process algebras; (iii) exhaustive model checking is the most preferred automated analysis method, enabled by formal ADLs; and (iv) consistency has been considered as the most relevant property in automated verification of architectural models. In another survey with practitioners, Ozkaya [6] observed that they follow the common sense about the criticism on informal notations, which fail to support complex design decisions. Although formal notations could fill this gap, users do not often use them due to the significant learning curve, lack of knowledge among stakeholders, low popularity in industry, and weak support with respect to tools and guidance. In spite of the inherent difficulty of pursuing formal methods, such a recent research has acknowledged the role of formal verification of architectural models as means of precisely determining if a software system can satisfy properties related to user requirements.

As reported by Zhang *et al.* [10], one of the most used techniques for formally verifying software architectures is model checking, an exhaustive, automatic verification technique whose general goal is to check if an architectural specification satisfies architectural properties. These authors analyzed a set of model checking techniques to formally verify software

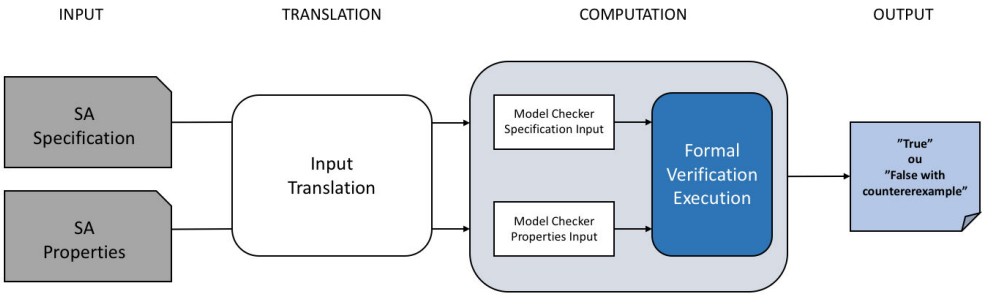


FIGURE 1. A process to formally verifying software architecture properties – adapted from Zhang *et al.* [10].

architectures and came up with a framework to classify and compare them. Figure 1 illustrates a process to formally verifying software architectures with model checking. This process takes as inputs specifications of both software architecture (e.g., a description in an ADL) and architectural properties expressed in some notation. If these specifications are not formally described, then a translation to a formal notation is first required. Next, the architectural properties regarding the software architecture under consideration are formally verified. The verification returns true, if the properties are satisfied, or false when a given property is violated.

III. RESEARCH METHODOLOGY

The systematic mapping study presented in this paper was conducted by following well-defined guidelines established in the literature [11], [12]. The basic steps are: (i) *planning*, which yields a protocol defining the research questions to be answered, the search strategy to be adopted, the criteria to be used for selecting primary studies, and the methods for extracting and synthesizing data; (ii) *execution*, in which primary studies are identified, selected, and evaluated according to the protocol; and (iii) *reporting* (or *analysis*), which aggregates information extracted from the relevant primary studies considering the research questions and outlines conclusions from them.

Research questions. Aiming at finding primary studies with approaches on the formal verification of software architecture descriptions, we have proposed the following research questions (RQs):

RQ1: What languages are used towards supporting verification, their characteristics, and supported views?
RQ2: What languages are used to specify architectural properties, their characteristics, and supported views?
RQ3: Which type of formal verification is addressed by the existing approaches?

Search strategy. To retrieve primary studies, we have used an automated search process performed over four popular electronic publication databases, namely ACM Digital Library, IEEEXplore, ScienceDirect.com, and Scopus. Based on the defined research questions, three main terms were initially identified, namely *architecture description language*, *formal language*, and *software architecture*. In addition, possible variations such as synonyms and singular/plural forms were considered, thereby resulting in the following search string:

```
(architecture description language OR
ADL) AND (formal language OR
specification language OR formal
semantics OR formal verification OR
formal analysis) AND software
architecture
```

The main terms were connected using the `AND` logical operator while the possible variations and synonyms were connected using the `OR` logical operator.

Selection criteria. Selection criteria were used to evaluate each primary study according to the defined research questions. The main goal was including studies that would be potentially relevant to answer the research questions and excluding the ones that would not contribute to answer them.

Three inclusion criteria (ICs) were considered:

IC1: The study proposes an approach to formally specify or verify software architectures.
IC2: The study proposes a framework or a tool to support the formal verification of software architectures.
IC3: The study analyzes software architecture verification approaches.

Seven exclusion criteria (ECs) were also established:

EC1: The study does not proposes an approach to formally specify or verify software architectures.
EC2: The study does not concern languages, techniques or approaches to formally specify or verify software architectures.
EC3: The study mentions a formal software architecture verification technique, but it does not detail the verification process itself.
EC4: The study is a previous version of a more recent study on the same research.
EC5: The study does not have an abstract or the full text is not available.
EC6: The study is a table of contents, foreword, tutorial, editorial, keynote talk or summary of conference/workshop.

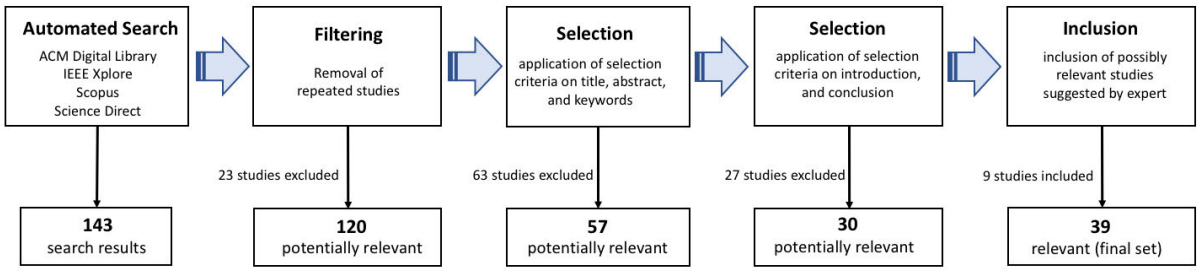


FIGURE 2. Phases for selecting the relevant primary studies.

EC7: The study is not written in English, which is the most common language in scientific papers.

In this work, a given primary study was considered as relevant if it has not met any exclusion criterion and met at least one inclusion criterion.

Data extraction and synthesis. To extract data from the selected primary studies as well as synthesize the results, a data extraction sheet was built with items related to the research questions and other relevant information. Besides basic information such as title and publication year and venue, extracted data concerned information about: (i) *architecture description*, in terms of used languages, visual support, formalism degree, and supported architectural views; (ii) *property specification*, in terms of used languages, type of architectural property, and expressed quality attributes or other properties of interest; (iii) the *translation process* from a given notation to a formal one, concerning automation degree and translation target language; and (iv) *formal verification process*, i.e., which type of verification is performed and if it is supported by some tool.

Selection process. As this study was carried out in December 2018, we considered primary studies published until then, thus covering a time window in which it would be possible to find consolidated research on the Software Architecture discipline. During the search process, the search string has undergone minor changes to make it compatible with the specificities of each database engine. Afterwards, the automated search procedure was performed over each electronic database according to the adapted search string. The automated search was limited to title, abstract, and keyword fields.

After retrieving the studies from the electronic databases, the selection process was conducted in five phases. In the first phase, results were unified into a single repository and duplicates were removed. The second phase encompassed reading title, abstract, and keywords of the retrieved studies, which were filtered according to the defined selection criteria (ICs/ECs). The third phase encompassed reading both introduction and conclusion of the studies and a new application of the selection criteria. The fourth phase consisted of fully reading the studies and filling out the data extraction sheet. At last, expert[1] suggestions were considered about the inclusion of possibly relevant studies. Figure 2 depicts the execution of these phases, which resulted in 39 studies selected as relevant (see Appendix A).

[1] An expert is a person with lengthy experience in both theory and practice regarding software architecture descriptions and formalisms.

IV. RESULTS

In this section, we summarize the results of the performed systematic mapping considering the research questions and data extracted/synthesized from the analyzed primary studies. We first present a brief overview of the selected primary studies (Section IV-A) and then the answers to each research question encompassing architecture descriptions, property specification, and formal verification processes (Sections IV-B to IV-D).

A. OVERVIEW OF SELECTED PRIMARY STUDIES

Distribution along the years. Figure 5 presents the distribution of publications along 24 years of research (1994-2018). It is possible to notice an average of 2.64 studies per year, but with a varying behavior in the research community, i.e., the number of studies increases and decreases along the
years.

Publication venues. Figure 4 illustrates the publication venues of the selected primary studies. Most papers were published in conferences (30/39), followed by journals (7/39). The low number of workshop papers (2/39) may be an indicator of the fact that the proposals have undergone a more solid evaluation, in line with the effort required for proposing solutions in this context. We also noticed that the selected studies are spread across 33 different venues, thus indicating that there is no main conference or workshop where studied related to the formal verification of architectural descriptions would be published.

Validation/evaluation methods. We have noticed that almost 92% of studies present some method of validation. However, more than half of these studies (20/39) merely give an example illustrating a possible scenario in which the proposed approach could be applied. Other forms of evaluation include case studies in real-world scenarios (13/39) as well as controlled experiments (3/39). Therefore, it is possible to conclude that stronger methods are required to validate/evaluate the proposed approaches as means of

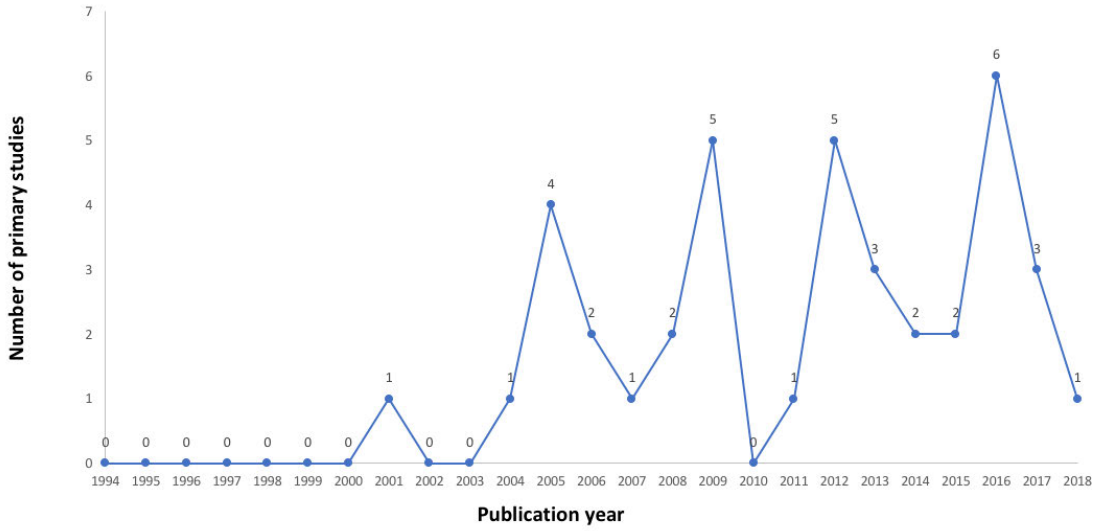


FIGURE 3. Publication of the selected primary studies along the years.

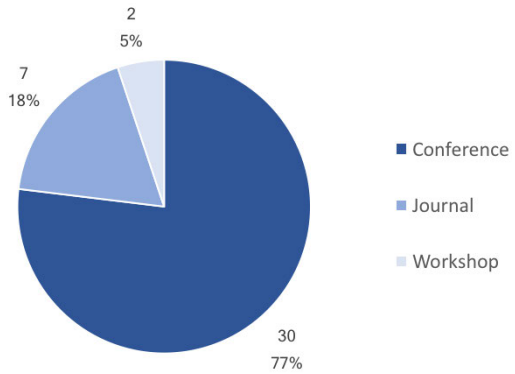


FIGURE 4. Venue types where the selected primary studies were published.

providing robust evidence about their efficiency, effectiveness, and feasibility.

B. ARCHITECTURE DESCRIPTION LANGUAGES

One of the goals of our study was identifying what ADLs have been used in the primary studies for architecture description towards supporting verification, their characteristics, and supported views. By analyzing data extracted from the selected studies, we have identified several languages and notations for describing software architectures, with no proposal being widely used towards a formal verification process. AADL and EAST-ADL have stood out among the used notations (reported in three studies each) due to their use in critical-domain embedded systems, which typically require formally verifying their software architectures. AADL has been used to describe architectural models in chaotic systems (S7), systems based on synchronous programming modules (S16), and real-time embedded systems (S39). In turn, EAST-ADL has been used to describe automotive embedded systems (S9, S12, S19) and support the verification of properties related to fault tolerance and correctness.

Besides identifying the ADLs used in the selected studies, we were interested into understanding their characteristics in terms of (i) representation, whether textual, visual or both, (ii) semantic formalism, and (iii) encompassed architectural viewpoints. We found 59% of the selected primary studies (23/39) using a visual notation to ease architectural modeling. In particular, UML/SysML profiles have stood out as means of visually representing software architectures in a fifth of these studies (S9, S12, S19, S21, S24, S28, S31, S34, S35). Such a representativeness of UML profiles is in line with the findings from Malavolta *et al.* [5] and Ozkaya [6] regarding the preference of practitioners to use this type of representation to support architecture description.

We have found 32 studies using an approach to describe software architectures. These approaches can be characterized in terms of formal ADLs (e.g., S16, S28), semi-formal ADLs (such as the ones based on UML (S9, S19) and XML (S5, S11)) or non-ADL formal notations (e.g., S2, S17), i.e., the direct use of mathematical formalisms without abstractions over them to describe a software architecture. We also realized that most studies have preferred to use visual notations likely due to reasons similar to those previously evidenced by Malavolta *et al.* [5] and Ozkaya [6], [13], such as the lower learning curve on describing software architectures with these languages.

Another investigated feature regarding the existing software architecture description notations stands for the architectural viewpoints that they can represent. Ninety-five percent of the selected studies (37/39) encompass both structural and behavioral concerns about the architecture whereas the other ones focus only on the behavioral viewpoint. Software industry practitioners are indeed interested in archi-

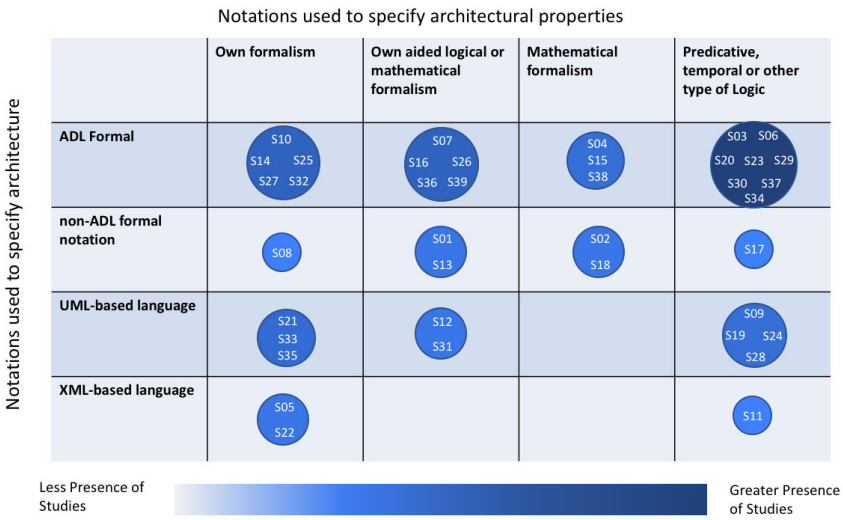


FIGURE 5. Publication of the selected primary studies along the years.

tecture description approaches supporting multiple viewpoints according to their needs [5]. Therefore, the expressive percentage of studies representing both structural and behavioral concerns in architecture description towards their formal verification is aligned with the interest of industry.

> **Main findings (RQ1).** There is no widely used ADL towards a formal verification process, even though there is a preference for approaches that use visual notation to facilitate architectural modeling, especially UML/SysML profiles. Most studies address both structural and behavioral concerns about software architectures when considering their formal verification.

C. SPECIFICATION OF ARCHITECTURAL PROPERTIES

We were interested into identifying what notations have been used in the primary studies to specify architectural properties. We have noticed that there is no single notation with wide use for specifying architectural properties. Most authors have preferred using their own formalisms (28% $\approx$ 11/39 of the selected studies) or combining them with some existing mathematical formalism (23% $\approx$ 9/39 of the selected studies). In turn, a third of the primary studies have used predicate, temporal or other type of logic. The particular interest in temporal logics as a formalism to express architectural properties is also evidenced by Zhang *et al.* [10] since most of these properties are temporal, i.e., they are qualified and grounded in a sequence of states of the system over time.

Based on the information about the notations for specifying both software architectures and their properties, we drawn a possible relationship among the used formalisms. As shown in Figure 5, we noticed a high concentration of studies associating the architectural description with an ADL and the specification of properties with mathematical formalisms, especially the logic-based ones.

We were also interested into understanding some features of the notations used to specify architectural properties in terms of the concerned viewpoints, whether structural or behavioral. We expected that most studies would claim to consider properties from both viewpoints since (i) previous studies have regarded this as an important gap in the past of ADLs [14] and (ii) supporting these multiple viewpoints is an expectation and current practice in industry [5]. However, the collected data contradicted our belief as most studies (27/39) address only behavioral concerns regarding architectural properties.

Another investigated feature about the notations used for specifying properties is related to what architectural properties or quality attributes can be specified with these languages. We have clustered the several properties found in three main categories. The first one was addressed by 19 studies and refers to properties related to reliability concerns. The second category refers to important properties related to concurrent systems, such as fairness, safety, liveness, deadlock-freedom, criticality, and priority, as concerned by 21 studies. The third category is related to properties specifically observed in real-time systems, which were addressed in five studies. It is possible to notice that the properties falling into these three categories are often found in critical systems, so that it is not by chance that the effort invested in formal approaches and verification techniques is larger in this class of systems. Safety standards to critical software such as DO178B/C for avionic systems [15], IEC 62304 for medical systems [16], and IEC 62279 [17] for rail systems strongly recommend adopting formal verification techniques as means of ensuring safety in these systems.

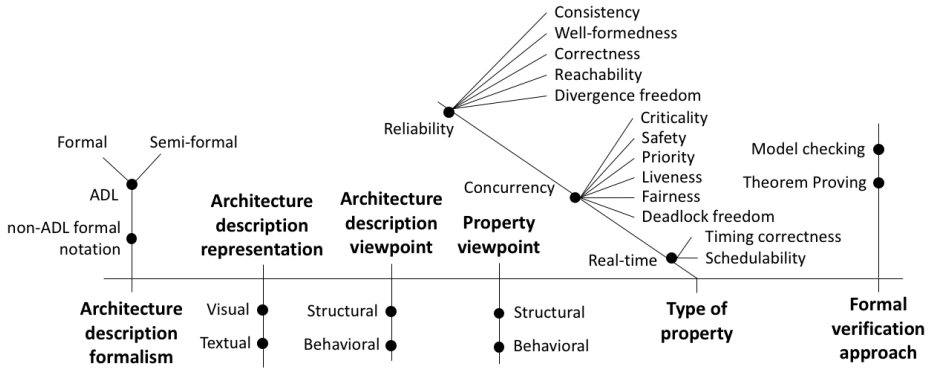


FIGURE 6. A multifaceted taxonomy for the formal verification of software architecture descriptions.

> **Main findings (RQ2).** There is no single notation widely used to specify architectural properties. Most authors have preferred to use their own formalisms, sometimes combining them with some existing mathematical formalisms. Most of the found studies considered only behavioral concerns related to architectural properties.

D. FORMAL VERIFICATION APPROACHES

At last, we aimed to identify the most used techniques in the formal verification of software architectures and if they have tool support. In their literature review, Zhang *et al.* [10] identified model checking as a frequently used formal verification technique. This was confirmed in our study as 77% of the analyzed studies (30/39) have applied such an approach to verify if a given software architecture satisfies architectural properties of interest. In a second place, two studies have used theorem proving to support formal verification and two other studies have used it along with model checking as complementary approaches. There are some possible reasons explaining such a prevalence of model checking. First, model checking is fully automated as opposed to theorem proving, which normally requires human guidance. Second, theorem proving requires much more effort from users, thus limiting its usage to experts and proficients in proving. Third, a software architecture (and its description) is one of the earliest models of a software system and hence it is a suitable input to model checking.

We also noticed that formal verification is often supported by one or more tools, either using model checking or theorem proving. Well-known tools such as Uppaal, PAT, and FDR have been used by a third of the selected primary studies. Zhang *et al.* [10] state that formal verification approaches in software architecture often make use of mature existing verification mechanisms such as FDR and UPPAAL, which have been successfully used in both academic and industry settings. Possible reasons in favor of these tools are (i) the ability of FDR to handle complex, industrial-sized model checking with respect to safety and liveness conditions and (ii) the high scalability and usefulness of UPPAAL to model, validate, and verify real-time systems and properties of interest. In turn, PAT comes with an user-interactive tool to verify temporal properties.

> **Main findings (RQ3).** Model checking showed to be the most commonly used formal verification technique. Formal verification is often supported by well-known tools such as UPPAAL, PAT, and FDR.

V. A TAXONOMY FOR THE FORMAL VERIFICATION OF SOFTWARE ARCHITECTURE DESCRIPTIONS

Taxonomies have been directly or indirectly used in the Software Architecture community to mature the knowledge field, in particular as means of understanding and classifying existing work. One of the contributions of this paper is a taxonomy arisen from the analysis of the studies selected in our systematic mapping study aiming at capturing important aspects related to the formal verification of software architecture descriptions. The proposed taxonomy is not solely intended to summarize the current state of the art on this topic, but it can also provide a basis to classify existing approaches and understanding their characteristics. Furthermore, the taxonomy can also serve as a reasoning framework towards proposing novel or improved approaches in this context.

Due to the multiplicity of perspectives under which existing work could be classified, our taxonomy has been structured upon the principles of the faceted analysis approach [18], one of the most used classification structures for taxonomies in Software Engineering [19]. In multifaceted taxonomies, there are more than one perspective (facet) to view and classify a given entity, so that each facet is independent and can have its own classes.

Figure 6 presents our taxonomy for the formal verification of software architecture descriptions. Such a taxonomy encompasses six main axes: (i) the *architectural description formalisms*, whether a typical ADL or a non-ADL formal notation; (ii) the *architecture description representation*, whether using visual and/or textual notations; (iii) *architecture description viewpoints*; (iv) *property viewpoints* of

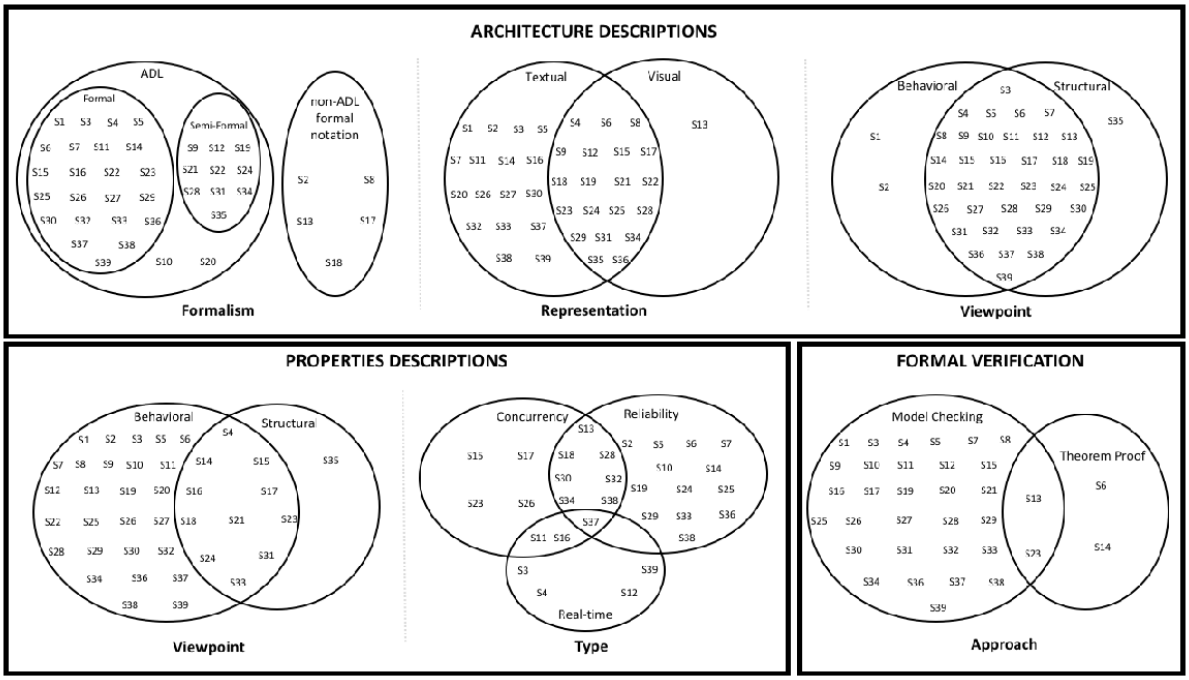


FIGURE 7. **Classification of the selected primary studies according to the taxonomy axes.**

interest; (v) the *types of properties* commonly found in formal architectural analysis approaches; and (vi) the *formal verification approach* for software architecture descriptions. The categories within the proposed taxonomy are not disjoint and other ones not currently covered can be easily included. As matter of future work, we intend to validate our taxonomy with respect to reliability and usefulness by surveying experts or conducting experimental studies.

The utility of a taxonomy can be demonstrated or exemplified by classifying existing literature [19]. Figure 7 presents a classification of the selected primary studies according to the main axes defined in our multifaceted taxonomy. It is important to mention that some studies do not present clear, sufficient information about their proposal, so that we have preferred leaving them unclassified for the sake of reliability and aiming to avoid any sort of misunderstanding. For example, one study does not restrict the ADL to be used, so that it is not possible to classify it into one of the categories of the *architecture description representation* axis. Nine studies do not detail what properties are specified by the proposed approach, thus hampering the classification of these studies with respect to the *type of property* axis. In turn, five studies do not report the approach used for formal verification purposes and hence they cannot be classified into any category of the *formal verification approach* axis.

VI. FUTURE DIRECTIONS IN RESEARCH AND DEVELOPMENT

Based on the analysis of the selected studies and the obtained findings, we came up with three potentially relevant directions for research and development regarding the formal verification of software architectures. As there is a growing need for verifying non-functional properties at the architectural level, we briefly discuss these issues with particular focus on scalability and dynamicity concerns. Another discussed issue is related to the use of semi-formal ADLs for verification purposes and the criticism from practitioners concerning the usability of formal ADLs.

Addressing scalability concerns in formal verification of software architectures. As reported in Section IV-D, model checking has been the most used verification technique in the analyzed studies, thereby confirming findings provided by Zhang *et al.* [10] almost ten years ago. Despite its wide and successful use, model checking has well-known limitations with respect to scalability. Traditional model checking approaches are not exempted from the exponential growth of the state space, the so-called state space explosion problem. This makes such a technique to be a prohibitive choice in many cases due to unneglectable execution time, computational resources, and effort from architects, important reasons that often hinder the adoption of formal-based techniques in industry [5]. Aiming at overcoming these limitations, alternative techniques such as assume-guarantee model-checking [20], compositional model-checking [21], [22], parallel model-checking [23], statistical model checking [24], and probabilistic model checking [25] have been proposed in the last years concerning affordable, computationally efficient approaches to rigorously verify properties of general specification. Despite our study did not have a specific research question about scalability issues on formal verification approaches for software architecture descriptions, this was a point that called our attention since our findings revealed a major use of model checking (77% of the selected studies). Among the 30 studies using model

checking as formal verification technique, only three studies (S10, S24, S27) have concerned the scalability of their approaches. The exploration of strategies aimed to provide model checking techniques with more scalability when verifying complex, critical can drive further research in this direction.

Addressing dynamicity concerns in formal verification of software architectures. Dynamicity is increasingly becoming an intrinsic property of the contemporary systems, which operate on environments that are highly dynamic, subjected to a number of changes. Software architectures for these systems hence need to be dynamic to accommodate such changes during runtime, ideally with minimum or no disruption as it is the case of certain safety- and mission-critical systems. The inherent characteristics of dynamic software architectures challenge activities such as formal architecture description and verification of architectural properties. Therefore, languages tailored to support both architecture description and property specification must consider the creation, interconnection or removal of architectural elements at runtime, which exist at a given instant in time and no longer exist at another. On the one hand, most of the existing formal ADLs have limitations with respect to the description of dynamic software architectures. On the other hand, the notations available in the literature to formally express architectural properties are not able to cope with these characteristics. We found only two studies (S23, S29) with proposals on the formal verification of properties of dynamic software architectures, thus revealing the scarcity of studies on this topic.

Addressing usability for formal ADLs. As formal ADLs provide unambiguous semantics, they are essential for supporting verification of architectures. However, Malavolta *et al.* [5] report that these notations are rarely used mainly due to the lack of mature tools and the need of specialized competencies for using them. Practitioners have indeed considered formal ADLs as quite difficult to use, besides requiring a high learning curve and huge effort to specify a system. They also argue that the supported automated analysis provided by formal ADLs is not compatible to the level of effort required for specifying a system using such ADLs. Therefore, an important research direction to foster the use of formal ADLs is the development of easy-to-use tools associated to formal ADLs aiming at facilitating the specification of architectures and allowing for early formal verification, thus positioning such ADLs in the mainstream of the architectural process.

VII. THREATS TO VALIDITY

To ensure high quality and scientific value to the findings of our systematic mapping study, a protocol was established a priori with clear research questions and explicit criteria to evaluate and select primary studies. This protocol was strictly followed according to well-accepted guidelines for systematic literature reviews and mapping studies [11], [12]. Nonetheless, potential threats to validity may affect the obtained results. In this section, we discuss some of these limitations and how we have attempted to mitigate them.

External validity. The most significant threat to external validity of this study is related to its incompleteness as relevant studies may have been missed. To reduce this threat, we have considered four of the most relevant available sources in Software Engineering. However, there are still limitations. First, some studies may have been missed due to technical limitations of the automated search engines themselves. Second, the selected databases do not represent an exhaustive list of publication sources, so that other databases might also be included. Third, we have not performed snowballing [26], a technique that allows checking the reference lists of the read studies aiming to find additional studies not retrieved in the automated search procedure. Fourth, grey literature (e.g., white papers, non peer-reviewed papers, etc.) was not herein considered as source of studies, but we deem that this does not constitute an additional threat as peer-review processes are a standard requirement of high quality publications.

Construct validity. To mitigate potential threats to construct validity, we established a well-defined protocol that guided all activities performed in this study. The selection of the primary studies according to the documented inclusion and exclusion criteria was rigorously carried out to ensure that they were indeed suited to answer the research questions.

Conclusion validity. Bias on data extraction may result in inaccuracy of the extracted data items and not all studies sufficiently and clearly describe information extracted as data items to support answering the research questions. To mitigate potential threats to conclusion validity, we have striven to reduce these bias by strictly adhering to the established protocol. Furthermore, the taxonomy resulted from the analysis of the selected studies could be another source of threat to the conclusion validity of our study as it has not been evaluated with other experts yet.

VIII. CONCLUSION

This paper presented the results of a systematic mapping study aimed to provide a panorama of the current state of the art on the role of architecture descriptions and property specification in supporting the formal verification of software architectures, an important activity to ensure the quality of software systems. We have systematically analyzed 39 primary studies found in electronic publication databases to (i) provide an overview of the research on this topic, (ii) understand features of languages used to describe software architectures and related properties of interest to be formally verified, and (iii) understand how software architecture formal verification has been conducted in this context.

Our analyses of the selected studies have resulted in several findings about the current state of the art on the formal verification of architecture descriptions:

- at least two studies about this topic have been published per year (on average), possibly indicating that it continues being a research topic of interest;
- the research landscape has shown to be quite fragmented, with studies published in a variety of venues;
- there is no reference language used to formally describe software architectures and specify architectural properties, thus resulting in a plethora of used languages and notations;
- visual and semi-formal notations have appeared as means of supporting software architecture description despite requiring an additional process to translate the produced models towards formal verification;
- supporting multiple viewpoints (in particular the structural and behavioral ones) has been considered as a relevant concern and hence addressed by both architecture description and property specifications;
- properties related to reliability, concurrency, and real-time characteristics have been addressed in the studies as a reflex of the relevance of these concerns in many critical systems;
- model checking has been the most used technique to support the formal verification of architectural properties despite its well-known limitations with respect to scalability; and
- verification mechanisms traditionally found in the formal methods field has been applied to software architectures over the years.

Besides the analysis of these studies and resulted findings, an important contribution brought by this paper is a taxonomy representing the research landscape on this topic and that can be used to further classify existing and future approaches as well as understand their characteristics. We have also discussed some promising directions in research and development in this context, such as (i) proposing alternative approaches to ease the architectural analysis activity, (ii) addressing dynamicity as an important concern in the architectural life cycle of the contemporary software systems, and (iii) putting effort on an attempt to make formal ADLs and related approaches more usable, mainly in industry.

APPENDIX A
SELECTED STUDIES

S1 M. Auguston, "Software architecture built from behavior models," ACM SIGSOFT Software Engineering Notes, vol. 34, no. 5, 2009.

S2 A. Bracciali, A. Brogi, and F. Turini, "A framework for specifying and verifying the behaviour of open systems," Journal of Logic and Algebraic Programming, vol. 63, no. 2, pp. 215-240, 2005.

S3 C. Braga, F. Chalub, and A. Sztajnberg, "A formal semantics for a Quality of Service contract language," Electronic Notes in Theoretical Computer Science, vol. 203, no. 7, pp. 103-120, 2009.

S4 L. Brim, I. Černá, P. Vařeková, and B. Zimmerova, "Component-interaction automata as a verification-oriented component-based system Specification," in Proceedings of the 2005 Conference on Specification and Verification of Component-based Systems. USA: ACM, 2005.

S5 L. Chen, L. Huang, C. Li, L. Wu, and W. Luo, "Design and safety analysis for system architecture: A Breeze/ADL-based approach," in Proceedings of 38th IEEE Annual Computer Software and Applications Conference. USA: IEEE, 2014, pp. 261-266.

S6 X. Ling, "A categorical approach for modeling and verifying dynamic software architecture," in Proceedings of the 7th IEEE International Conference on Software Security and Reliability Companion. USA: IEEE, 2013, pp. 168-175.

S7 D. De Niz, "Architectural concurrency equivalence with chaotic models," in Proceedings of the 5th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software. USA: IEEE, 2008, pp. 57-67.

S8 A. T. E. Dib and Z. Sahnoun, "Model checking of multi agent system architectures using BigMC," in Proceedings of the 2015 Federated Conference on Computer Science and Information Systems. USA: IEEE, 2015, pp. 1717-1722.

S9 E. P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson, "ViTAL: A verification tool for EAST-ADL models using UPPAAL PORT," in Proceedings of the 17th IEEE International Conference on Engineering of Complex Computer Systems. USA: IEEE, 2012, pp. 328-337.

S10 J. M. Franco, R. Barbosa, and M. Zenha-Rela, "Automated reliability prediction from formal architectural descriptions," in Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture. USA: IEEE, 2012, pp. 302-309.

S11 Y. Fu, "Modeling, validation and automated composition of Web services," in Proceedings of the 6th International Conference on Web Engineering. USA: ACM, 2006, pp. 217-224.

S12 A. Goknil, J. Suryadevara, M. A. Peraldi-Frati, and F. Mallet, "Analysis support for TADL2 timing constraints on EAST-ADL models," in K. Drira, Ed. Proceedings of the 7th European Conference on Sofware Architecture. Lecture Notes in Computer Science, vol. 7957. Germany: Springer Berlin Heidelberg, 2013, pp. 89-105.

S13 X. He, H. Yu, T. Shi, J. Ding, and Y. Deng, "Formally analyzing software architectural specifications using SAM," Journal of Systems and Software, vol. 71, no. 1-2, pp. 11-29, 2004.

S14 D. Hemer and D. Yulin, "Specifying software architectures using a formal-based approach," in Proceedings

of the 19th Australian Software Engineering Conference. USA: IEEE, 2008, pp. 279-288.

S15 H. Hoyos, R. Casallas, and F. Jimenez, "HiLeS-T:: An ADL for early requirement verification of embedded systems," in Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems. USA: ACM, 2012, pp. 7-12, 2012.

S16 E. Jahier, N. Halbwachs, and P. Raymond, "Synchronous modeling and validation of priority inheritance schedulers," in M. Chechik and M. Wirsing, Eds. Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, vol. 5503. Germany: Springer Berlin Heidelberg, 2009, pp. 140-154.

S17 C. Jerad and K. Barkaoui, "On the use of rewriting logic for verification of distributed software architecture description based LfP," in Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping. USA: IEEE, 2005, pp. 202-208.

S18 A. Kamandi and J. Habibi, "Toward a new analyzable architectural description language based on OSAN," in Proceedings of the 2nd International Conference on Software Engineering Advances. USA: IEEE, 2007, pp. 2-7.

S19 E. Y. Kang, L. Ke, M. Z. Hua, and Y. X. Wang, "Verifying automotive systems in EAST-ADL/Stateflow using UPPAAL," in Proceedings of the 2015 Asia-Pacific Software Engineering Conference. USA: IEEE, 2016, pp. 143-150.

S20 M. Khalgui, H.-M. Hanisch, and A. Gharbi, "Model-checking for the functional safety of control component-based heterogeneous embedded systems," in Proceedings of the 2009 IEEE Conference on Emerging Technologies and Factory Automation. USA: IEEE, 2009, pp. 1-10.

S21 L. Lima, A. Miyazawa, A. Cavalcanti, M. Cornelio, J. Iyoda, A. Sampaio, R. Hains, A. Larkham, and V. Lewis, "An integrated semantics for reasoning about SysML design models using refinement," Software and Systems Modeling, vol. 16, no. 3, pp. 875-902, 2017.

S22 R. Maraoui and E. Cariou, "A mediation based approach for formal verification of Web services Composition", in Proceedings of the 2017 International Conference on Engineering & MIS. USA: IEEE, 2017, pp. 1-6.

S23 R. Mateescu and F. Oquendo, "π-AAL: An architecture analysis language for formally specifying and verifying structural and behavioural properties of software architectures," SIGSOFT Software Engineering Notes, vol. 31, no. 2, pp. 1-19, 2006.

S24 S. Mesli-Kesraoui, D. Kesraoui, F. Oquendo, A. Bignon, A. Toguyeni, and P. Berruet, "Formal verification of software-intensive systems architectures described with piping and instrumentation diagrams", in B. Tekinerdogan, U. Zdun, and A. Babar. Proceedings of the 10th European Conference on Software Architecture. Lecture Notes in Computer Scince, vol. 9839. Switzerland: Springer International Publishing AG, 2016, pp. 210-226.

S25 F. Oquendo, "Case study on formally describing the architecture of a software-intensive system-of-systems with SosADL," in Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics. USA: IEEE, 2017, pp. 2260-2266.

S26 M. Ozkaya and C. Kloukinas, "Towards Design-by-Contract based software architecture design," in Proceedings of the 12th IEEE International Conference on Intelligent Software Methodologies, Tools and Techniques. USA: IEEE, 2013, pp. 157-164.

S27 M. Ozkaya and C. Kloukinas, "Architectural specification and analysis with XCD - The Aegis Combat System case study," in Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development. Portugal: SciTePress, 2014, pp. 368-375.

S28 M. Ozkaya and M. A. Kose, "SAwUML - UML-based, contractual software architectures and their formal analysis using SPIN," Computer Languages, Systems and Structures, vol. 54, pp. 71-94, 2018.

S29 E. Cavalcante, J. Quilbeuf, L. M. Traonouez, F. Oquendo, T. Batista, and A. Legay, "Statistical model checking of dynamic software architectures", in B. Tekinerdogan, U. Zdun, and A. Babar. Proceedings of the 10th European Conference on Software Architecture. Lecture Notes in Computer Scince, vol. 9839. Switzerland: Springer International Publishing AG, 2016, pp. 185-200.

S30 A. Rademaker, C. Braga, and A. Sztajnberg, "A rewriting semantics for a software architecture description language," Electronic Notes in Theoretical Computer Science, vol. 130, pp. 345-377, 2005.

S31 D. Regep and F. Kordon, "Lfp: A specification language for rapid prototyping of concurrent systems," Proceedings of the 12th International Workshop on Rapid System Prototyping. USA: IEEE, 2001, pp. 90-96.

S32 S. Song, J. Zhang, Y. Liu, M. Auguston, J. Sun, J. S. Dong, and T. Chen, "Formalizing and verifying stochastic system architectures using Monterey Phoenix," Software and Systems Modeling, vol. 15, no. 2, pp. 453-471, 2016.

S33 S. R. Taoufik, B. M. Tahar, S. Layth, and K. Mourad, "Towards a formal approach for the verification of SCA/BPEL software architectures," in Proceedings of the 8th International Conference on Information, Intelligence, Systems & Applications. USA: IEEE, 2017.

S34 S. R. Taoufik, B. M. Tahar, and K. Mourad, "Behavioral verification of UML2.0 software architecture," in Proceedings of the 12th International Conference on Semantics, Knowledge and Grids. USA: IEEE, 2017, pp. 115-120.

S35 S. R. Taoufik, B. M. Tahar, K. Mourad, and M. Faouzi, "UML2.0 formalization and Acme verification of the qualitative properties of software architectures," in Proceedings of the 25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises. USA: IEEE, 2016, pp. 192-197.

S36 G.-q. Zhang, H.-j. Shi, and M. Rong, "Mismatch detection of asynchronous Web services with timed constraints," in Proceedings of the 2011 IEEE Asia-Pacific Services Computing Conference. USA: IEEE, 2011, pp. 251-258.

S37 J. Zhang, Y. Liu, M. Auguston, J. Sun, and J. S. Dong, "Using Monterey Phoenix to formalize and verify system architectures," in Proceedings of the 19th Asia-Pacific Software Engineering Conference. USA: IEEE, 2012, pp. 644-653.

S38 J. Zhang, Y. Liu, J. Sun, J. S. Dong, and J. Sun, "Model checking software architecture design," in Proceedings of the 2012 IEEE International Symposium on High Assurance Systems Engineering. USA: IEEE, 2012, pp. 193-200.

S39 Z. Yang, K. Hu, D. Ma, and L. Pi, "Towards a formal semantics for the AADL behavior annex," in Proceedings of the 2009 Design, Automation & Test in Europe Conference & Exhibition. USA: IEEE, 2009, pp. 1166-1171.

REFERENCES

[1] *Systems and Software Engineering—Architecture Description*, document ISO/IEC/IEEE 42010:2011(E), Jan. 2011.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Reading, MA, USA: Addison-Wesley, 2013.

[3] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2011.

[4] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Hoboken, NJ, USA: Wiley, 2010.

[5] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 869–891, Jun. 2013.

[6] M. Ozkaya, "Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling?" *Inf. Softw. Technol.*, vol. 95, pp. 15–33, Mar. 2018.

[7] J. J. P. Tsai and K. Xu, "A comparative study of formal verification techniques for software architecture specifications," *Ann. Softw. Eng.*, vol. 10, nos. 1–4, pp. 207–223, Nov. 2000.

[8] L. Dobrica and E. Niemelä, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, Jul. 2002.

[9] M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," in *Proc. 11th Asia–Pacific Softw. Eng. Conf.*, Nov./Dec. 2004, pp. 600–607.

[10] P. Zhang, H. Muccini, and B. Li, "A classification and comparison of model checking software architecture techniques," *J. Syst. Softw.*, vol. 83, no. 5, pp. 723–744, 2010.

[11] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research—A participant-observer case study," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 638–651, Jun. 2011.

[12] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. 12th Int. Conf. Eval. Assessment Softw. Eng.*, U.K. BCS, 2008, pp. 68–77.

[13] M. Ozkaya, "The analysis of architectural languages for the needs of practitioners," *Softw. Pract. Exper.*, vol. 48, pp. 985–1018, May 2018.

[14] R. Hilliard and T. Rice, "Expressiveness in architecture description languages," in *Proc. 3rd Int. Workshop Softw. Archit.*, 1998, pp. 65–68.

[15] *Software Considerations Airborne Syst. Equip. Certification*, document, RTCA RTCA/DO-178C, 2012.

[16] A. Coronato, *Engineering High Quality Medical Software: Regulations, Standards, Methodologies And Tools For Certification*. London, U.K.: IET, 2018.

[17] *Railway Applications—Communication, Signalling And Processing Systems—Software For Railway Control And Protection Systems*, IEC 62279:2015, 2015.

[18] B. H. Kwasnik, "The role of classification in knowledge representation and discovery," *Library Trends*, vol. 48, no. 1, pp. 22–47, 1999.

[19] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method," *Inf. Softw. Technol.*, vol. 85, pp. 43–59, May 2017.

[20] C. S. Păsăreanu, M. B. Dwyer, and M. Huth, "Assume-guarantee model checking of software: A comparative case study," in *Theoretical and Practical Aspects of SPIN Model Checking* (Lecture Notes in Computer Science), D. Dams, R. Gerth, S. Leue, and M. Massink, Eds., Berlin, Germany: Springer-Verlag, 1999, pp. 168–183.

[21] M. V. M. Oliveira, P. Antonino, R. Ramos, A. Sampaio, A. Mota, and A. W. Roscoe, "Rigorous development of component-based systems using component metadata and patterns," *Formal Aspects Comput.*, vol. 28, no. 6, pp. 937–1004, 2016.

[22] M. S. C. Filho, M. V. M. Oliveira, A. Sampaio, and A. Cavalcanti, "Compositional and local livelock analysis for CSP," *Inf. Process. Lett.*, vol. 133, pp. 21–25, May 2018.

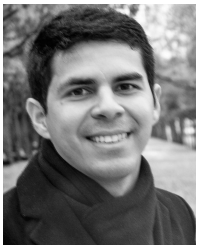
[23] L. Brim and J. Barnat, "Tutorial: Parallel model checking," in *Model Checking Software* (Lecture Notes in Computer Science), vol. 4595, D. Bošnački and S. Edelkamp, Eds. Berlin, Germany: Springer, 2007, pp. 2–3.

[24] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Runtime Verification* (Lecture Notes in Computer Science), vol. 17, H. Barringer, Eds. Berlin, Germany: Springer, 2010, pp. 122–135.

[25] M. Kwiatkowska, G. Norman, and D. Parker, "Advances and challenges of probabilistic model checking," in *Proc. 48th Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, Sep./Oct. 2010, pp. 1691–1698.

[26] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, 2014, p. 38.

CAMILA ARAÚJO received the M.Sc. degree in systems and computing from the Federal University of Rio Grande do Norte, Natal, Brazil, in 2006, where she is currently pursuing the Ph.D. degree in computer science. She is currently an Assistant Professor with the Computer Science Department, State University of Rio Grande do Norte, Natal. Her current research interests include software architecture, architectural description languages, formal verification, and property specification.



EVERTON CAVALCANTE received the Ph.D. degree in computer science from the Federal University of Rio Grande do Norte, Natal, Brazil, in 2016, and the Ph.D. degree in informatics from the University of Southern Brittany, France, in 2016. He is currently an Assistant Professor with the Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte. He has expertise in computer science with an emphasis on software architecture and distributed systems. His current research interests include middleware, cloud computing, ubiquitous computing, the Internet of Things, smart cities, software dynamic reconfiguration, architecture description languages, and systems-of-systems.

THAIS BATISTA received the Ph.D. degree in informatics from the Pontifical Catholic University of Rio de Janeiro, Brazil, in 2000. She did the Postdoctoral Research and the Senior Internship with Lancaster University, U.K., from 2004 to 2005 and from 2013 to 2014, respectively. She is currently a Full Professor with the Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil, and a Research Fellow Level 1D of the Brazilian National Council for Scientific and Technological Development. She has expertise in computer science with an emphasis on software architecture and distributed systems. Her current research interests include architecture description language, cloud computing, the Internet of Things, middleware, smart cities, and systems-of-systems.

MARCEL OLIVEIRA received the Ph.D. degree in computer science from the University of York, U.K., in 2005. He did the Postdoctoral Research with the Federal University of Pernambuco, Brazil, from 2011 to 2012. He is currently an Associate Professor with the Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil. He is currently a member of the Brazilian National Institute of Software Engineering and the Special Committee on Formal Methods of the Brazilian Computer Society. He has expertise in computer science with an emphasis on formal methods. His current research interests include refinement calculation and tactics, concurrency, semantics of formal languages, formal methods, and code synthesis from formal specifications.

FLAVIO OQUENDO received the Ph.D. and Research Direction Habilitation degrees from the University of Grenoble, France. He is currently a Full Professor of computing with IRISA, a joint research unit of CNRS, Université de Bretagne-Sud, France, where he leads the research on formal approaches for architecting software-intensive systems-of-systems. He has published over 200 refereed journals and conference papers in computer science and software engineering. His current research interests include formal languages, processes, and tools to support the efficient architecture of software-intensive systems-of-systems. They include formal description and development techniques for system-of-systems architecture, analysis, refinement, and evolution and their applications in industrial settings. He served on Program Committees of over 100 international conferences, chaired 10 of them, and acted as a Referee for over 20 international journals. In particular, he has been involved in the French, European, and IEEE/IFIP Conferences on Software Architecture, namely, CAL, ECSA, and WICSA (now ICSA). He was a recipient of the Research Excellence Award from the French Ministry of Research and Higher Education. He has served as a General Chair for ECSA, a Program Committee Chair for CAL, ECSA, and WICSA, a Steering Committee Chair for ECSA, and a Steering Committee Member for CAL, ECSA, and WICSA. He has acted as an Expert and an Evaluator for Research Projects in ICT for several Programs of the European Commission, in particular, FP7 and H2020 on Software-Intensive Systems-of-Systems and Trustworthy Software-Intensive Systems.