Model Checking Software Architecture Techniques

1st Tigau-Almasan Alexandra
Software Engineering
Babeș-Bolyai University
Cluj-Napoca, Romania

2nd Alexandru Stoica
Software Engineering
Babeș-Bolyai University
Cluj-Napoca, Romania

3rd Robert Corman
Software Engineering
Babeș-Bolyai University
Cluj-Napoca, Romania

4th Andreea Gorgos
Software Engineering
Babeș-Bolyai University
Cluj-Napoca, Romania

***Abstract*—The evaluation of quality requirements from a software architecture specification becomes more relevant and challenging during the design and coding process of our software products. Various techniques have been proposed for exhaustive and automatic verification of an architectural specification based on our architectural models, among them, model checking.**

The goal of this paper is to analyse and summarise a classification and comparison framework for model checking software architecture techniques proposed by *Pengcheng Zhang*, *Henry Muccini* and *Bixin Li*, in their article *"A classification and comparison of model checking software architecture techniques"*.

***Index Terms*—model-checking, software architecture, formal verification**

I. Introduction

Software Architecture (SA) plays an essential role in high-level descriptions of large-scale structures of software systems. It has emerged as a dominant method for documenting architectural decisions, predicting architectural qualities before implementing the system, and guiding the design and coding process. The primary usage of software architecture specification is to analyse and validate architectural choices, as an addition to traditional code-level analysis techniques.

In the past decade, formal modelling techniques have been developed to software architecture designs, which allow us to achieve precise specification and rigorous verification of the software specification and its properties [Zhang et al., 2010]. However, the current practice of software architecture modelling primary relies on diagrammatic notations and informal textual descriptions.

II. Motivation

Most of the software development activities rely on the selection of the right architecture, which satisfies specific functional and non-functional requirements [Zhang et al., 2010]. Therefore the verification of the software architecture's specification and its properties has a highly significant role in the development life-cycle of a product.

Automatic verification provides an efficient and effective method which determines whether a system can satisfy the set of requirements. Over the years, researchers proposed multiple analysis techniques and tools such as deadlock detection, model checking, testing, performance analysis et al.

Model-checking has the advantage of being fully automated [Zhang et al., 2010]. A checker receives a formal model of the software system architecture and the specification of a property to be checked and returns either true if the property holds, or false if the specification violates it.

III. Problem Statement

According to [Zhang et al., 2010] goal statement, the researchers seek to classify and compare multiple model checking of software architecture techniques. They develop a framework based on a set of classification entities and attributes and present a robust classification system of all proposed model checking techniques used on software architectures in the past two decades.

IV. Related Work

Over the past two decades, the research community proposed many model checking SA techniques, yet little work has been done on their classification and comparison.

A close related paper from Tsai and Xu (2000) [Tsai and Xu, 2000] compares formal verification techniques applied over software architecture specifications. In their paper, the authors translate the architectural specification into a labelled transition system model (which acts as a bridge between an architecture description language and an input language of any verification tool). The paper itself focuses on formal verification tools, not specifically on model checking software architectures, and does not cover recently proposed approaches since its publication in 2000.

Other papers such as [Dobrica and Niemela, 2002], [Babar et al., 2004] propose comparison frameworks for software architecture evaluation and analysis methods, both focused on quality attributes such as maintenance, reusability of an existent knowledge base, reliability et al.

V. Proposed Approach

In [Zhang et al., 2010] the authors classify 17 model checking approaches for software architecture, using a classification framework which focuses on two primary goals:

- understanding the primary artifacts and activities used when model checking software architectures by measuring the attributes of macro-entities such as ***the input data*** (SA specification and properties), ***the model checking engine***, and the input ***translation mechanism*** that connects the two components
- understanding the associated qualities associated with each technique by measuring the ***usability***, ***reliability***, ***scalability*** and ***expressiveness*** of each engine output

The framework itself evaluates the main activities in model checking SA, which according to the article, cluster into three macro-entities: input, translation and computation.

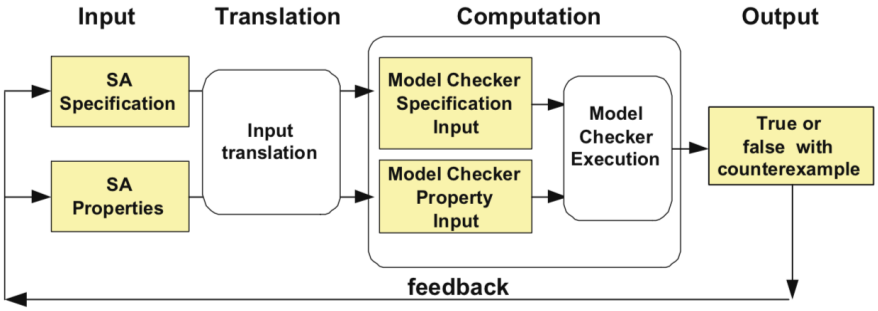

Fig. 1. Main Activities in Model Checking Software Architectures

Name	Year	Principal Investigator(s)	References
Wright	1997	Allen and Garlan	Allen and Garlan (1997, 1998)
Tsai	1997	Tsai	Tsai et al., 1997
CHAM	1998	Inverardi and Wolf	Compare et al., 1999; Corradini et al., 2006
Bose	1999	Bose	Bose 1999
Darwin/FSP	1999	Magee, Kramer, Giannakopoulou	Giannakopoulou, 1999; Magee et al., 1999
PoliS	1999	Ciancarini and Mascolo	Ciancarini and Mascolo, 1999; Ciancarini et al., 2000
Arcade	2001	Barber, Graser, and Holt	Barber et al., 2001
Archware	2001	Warboys and Oquendo	Oquendo et al., 2004; Oquendo, 2004; Mateescu and Oquendo, 2006
CHARMY	2001	Pelliccione, Inverardi, Muccini	Inverardi et al., 2001; Pelliccione et al., 2009
AEmilia	2002	Bernardo	Balsamo et al., 2003; Aldini et al., 2009
SAM	2002	He, Ding, Deng, Yu	Shi and He, 2003; Yu et al., 2004; He et al., 2004; He, 2005
Garlan et al.	2003	Garlan, Khersonsky, Kim	Garlan et al. (2003)
CBabel	2003	Braga and Sztajnberg	Braga and Sztajnberg, 2003; Rademaker et al., 2005; Braga et al., 2009
Zanolin et al.	2003	Zanolin, Ghezzi, Baresi	Zanolin et al. 2003
AutoFocus/Auto Focus 2	2005	Philipps, Slotosch	MunichUT, 2006
Fujaba	2005	Giese	Burmester et al., 2005; Becker et al., 2006; Burmester et al., 2007
Lfp	2005	Jerad and Barkaoui	Jerad and Barkaoui, 2005; Jerad et al., 2007; Jerad et al., 2008

Fig. 2. The 17 Surveyed Model Checking SA Approaches

We can decompose the input entity into two main entities: the specification of the software architecture and the specification of SA properties *(e.g. safety, liveness)* [Zhang et al., 2010]. Those entities are formally represented using architectural description languages and specific notations, and some ADLs offer stable tool support.

Most model checking engines used in the computation activity require a translation of the SA specification and SA properties. The translation process has different levels of automation *(e.g. no translation, partial translation and total translation)*, based on the available tool support.

For each model checking technique evaluation, the authors consider four characteristics: usability, reliability, scalability and expressiveness *(selected based on related studies)*. The framework evaluates the usability factor based on understandability, learnability and operability. The reliability entity classifies model checking SA approaches according to inception, development, refinement, and dormant. The authors analyse which type of properties can be checked by a model checking technique to calculate its expressiveness. The scalability factor considers two primary metrics: the model checking SA ability to specify complex architectures and the ability to model check complex inputs [Zhang et al., 2010].

VI. Validation of Approach

A. Goal 1 - Input, Translation and Computation Process

Architects can specify the software architecture specification in three ways: using formal specification languages (architecture description language - ADL or general formal languages), model-based diagrammatic notations (such as UML) and box-and-line notations.

In [Zhang et al., 2010], the researchers classify all model-checking techniques based on the method used to create the software architecture's specification *(see fig.3)* into two categories: formal SA specification and model-based specification.

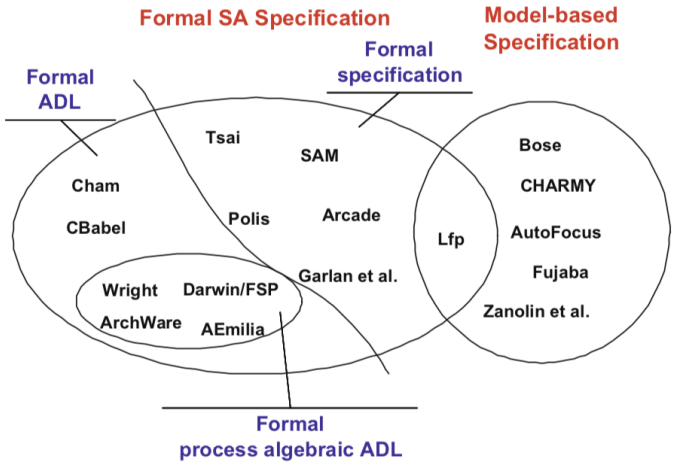

Fig. 3. Formal or Model-Based SA Specification

It is important to remark that some techniques such as Lfp can use both formal textual notations and model-based specification to combine the advantages of both categories. Most model-checking techniques use formal textual notations, while recent approaches use model-based specification.

The SA properties are typically specified using existing property specification languages, yet some offer support for

new formal property specification languages or graphical abstractions [Zhang et al., 2010] *(see fig.4)*. The author collected information on how each approach represents abstract concepts such as connectors, components, interfaces, types and instances, semantics, and configuration. When checking a SA property, some of those concepts play a crucial role in the formal validation process. Although many model checking techniques offer support for component specification, same lack support for connectors *(Tsai, CHAM et al.)*, interfaces *(SAM, CBabel et al.)*, and types *(Darwin, PoliS, Arcade)*.

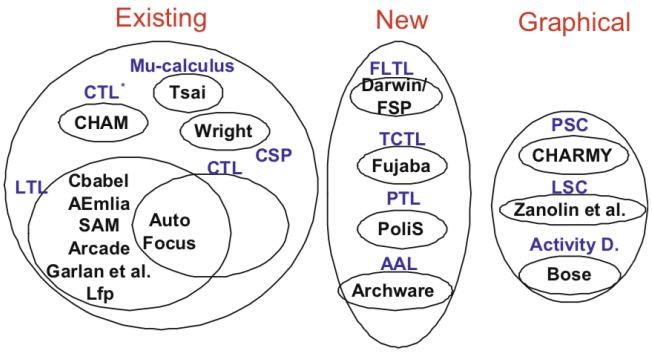

Fig. 4. Existing, New or Graphical Property Specification Languages

Our evaluated techniques use different engines to perform model checking. Bose, Arcade, CHARMY, and Zanolin et al. use SPIN as their model checking engine; thus, they all use Promela and LTL formulae as their input languages [Zhang et al., 2010]. Some approaches provide tool support (internal or public) for model checking, while others such as Zanolin have no tool support integrated.

If the model checking SA approach relies on external a model checking engine, then its SA specification or SA properties require translation from their input language; (*see fig.5*). Otherwise, when the SA approach supports its model checking engine, then its specification and properties are directly used as input for the engine).

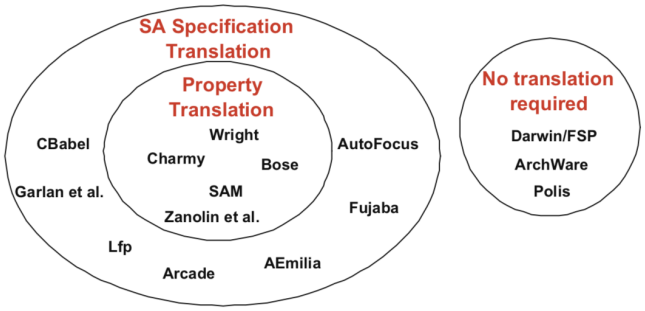

Fig. 5. Input Translation

B. Goal 2 - Usability, Reliability, Expressiveness, Scalability Summary

When evaluating the usability of model checking SA approaches, the paper [Zhang et al., 2010] focuses on four aspects:

- explicit configuration - Medvidovic and Taylor, 2000 remarks this feature to facilitate understandably of architectural structure in ADLs;
- graphical notations - sound graphical notations for expressing the architecture and architectural properties improve the usability of our approach;
- output comprehensibility
- degree of automation

The evaluation results concluded that AutoFOCUS is the most practical model checking SA approach, followed by CHARMY and Fujaba. CHAM and Lfp scored the lowest on this factor due to their implicit configuration, difficult output readability and low automation for model checking.

The reliability evaluation takes into consideration the level of maturity and validation of our model checking approach. Based on the presented data, model checking SA techniques such as Archware, CHARMY, et al. are reliable, since they are in a refinement stage of maturity and multiple industrial studies apply them [Zhang et al., 2010].

Support for hierarchical and compositional modelling influences the scalability of an approach, making techniques such as Darwin/FSP, Archware, AutoFOCUS, and Fujaba more scalable than others.

The expressiveness factor takes into consideration a few modelling support features for components, connectors, properties, interfaces, and types. Wright and Archware are the most expressive model checking SA techniques among those surveyed in the paper, being the only two supporting all those features [Zhang et al., 2010].

VII. Conclusions

Using their classification and comparison framework, the authors identify the direction of development for model checking techniques on software architectures.

Their analysis concluded that the model checking SA techniques have evolved from purely theoretical studies to practical approaches with mature tool support and readable output. Recently introduced techniques make use of model-based notations and do not rely on architecture description languages (ADLs). Modern model checking procedures can verify complex and domain-specific software architectures.

The authors believe that the focus in the industry is shifting from coding to modelling and that new model checking SA techniques should support automatic verification of SA specification against both system requirements and implementation.

New model checking approaches should be able to check dynamic and service-oriented software architectures because most of our systems are dynamically evolving (components are removed or plugged in based on our system's requirements).

	Usability				Reliability		Scalability			Expressiv.
	Is the SA configuration a first class element ?	Is the model checking SA description/ SA property modelling semantically sound?	Is the output comprehensible to a software architect?	Which is the degree of automation in model checking SA techniques?	What is the level of maturity	Has the method been validated?	Does the model checking SA technique support hierarchical modelling?	Is the SA specification textual?	Does the model checker support abstraction and minimization?	Can the desired properties be checked by the model-checking SA approach?
Wright	Explicit	Textual/Textual	difficult	medium	dormant	S & S	NO	Textua	Compression functions	Comp, Comp Int
Tsai	Implicit	Textual/Textual	difficult	none	dormant	S & S	No	Textual	None	None
CHAM	Implicit	Textual/Textual	difficult	none	development	S & S	No	Textual	None	Comp
Bose	Implicit	Graphical/ Graphical and sound	difficult	medium	dormant	S & S	No	Graphical	state compression, on-the fly, and POR	Comp, Con, Type
Darwin/ FSP	Implicit	Textual/Textual	difficult	high	development	M & I	Yes	Textual and Graphical	compositional minimization	Comp, Comp Int, Type
PoliS	Implicit	Textual/Textual	difficult	medium	dormant	S & S	No	Textual	supports local searches	Comp
Arcade	Implicit	Textual/Textual	easy	medium	dormant	S & S	No	Textual	state compression, on-the fly, and POR	None
Archware	Explicit	Textual/Textual	difficult	high	refinement	M & I	Yes	Textual	compositional on-the fly	all
CHARMY	Explicit	Graphical/ Graphical and sound	easy	medium	refinement	M & I	No	Textual and Graphical	state compression, on-the fly, and POR	Comp
AEmilia	Explicit	Textual/Textual	difficult	high	refinement	M & I	No	Textual	OBDD and BMC	Comp, Comp Int, Type
SAM	Implicit	Textual/Textual	difficult	medium	dormant	M & I	Yes	Graphical	abstractions and reductions	Comp, Con
Garlan et al.	Implicit	Textual/Textual	difficult	medium	dormant	S & S	No	Graphical	composition and abstract	Comp
CBabel	Implicit	Textual/Textual	difficult	high	dormant	S & S	No	Textual	BDD and on-the-fly	Comp, Type
Zanolin et al.	Implicit	Graphical/ Graphical and sound	difficult	low	dormant	S & S	No	Graphical	state compression, on-the fly, and POR	Comp, Con
Auto FOCUS	Explicit	Graphical and sound/ textual	easy	high	refinement	M & I	Yes	Textual and Graphical	OBDD and POR	Comp, Comp Int
Fujaba	Implicit	Graphical and sound/ textual	difficult	high	refinement	M & I	Yes	Textual and Graphical	memory reduction	Comp, Comp Int
Lfp	Implicit	Graphical/Textual	difficult	low	dormant	S & S	No	Textual and Graphical	on-the-fly computation and BDD	Comp, Con, Type

Fig. 6. Summary Goal 2 Framwork Attributes

VIII. Personal Conclusions

The article offers two primary contributions: a classification and comparison framework for model checking SA techniques which allows us to evaluate future findings and an excellent presentation of 17 model checking approaches for software architectures from which we can identify the advantages and disadvantages of using them in the industry.

Although some attributes used in the second goal of their framework's parameters are not explicitly justified, the study presents a robust classification system of all the proposed model checking techniques used on software architectures in the past two decades. Their work offers sharp clarity on the development direction of those techniques; and emphasises the primary features that should be taken into consideration when developing new model checking SA techniques.

References

[Babar et al., 2004] Babar, M. A., Gorton, I., and Gorton, I. (2004). Comparison of scenario-based software architecture evaluation methods. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC '04, pages 600–607, Washington, DC, USA. IEEE Computer Society.

[Dobrica and Niemela, 2002] Dobrica, L. and Niemela, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653.

[Tsai and Xu, 2000] Tsai, J. J. and Xu, K. (2000). A comparative study of formal verification techniques for software architecture specifications. *Annals of Software Engineering*, 10(1):207–223.

[Zhang et al., 2010] Zhang, P., Muccini, H., and Li, B. (2010). A classification and comparison of model checking software architecture techniques. *Journal of Systems and Software*, 83(5):723 – 744.