Towards CPS Verification Engineering

Andreas Müller
andreas.mueller@fh-hagenberg.at
University of Applied Sciences
Upper Austria
Hagenberg, Austria

Stefan Mitsch
smitsch@cs.cmu.edu
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Werner Retschitzegger
werner.retschitzegger@jku.at
Department of Cooperative Information Systems
Johannes Kepler University
Linz, Austria

Wieland Schwinger
wieland.schwinger
Department of Cooperative Information Systems
Johannes Kepler University
Linz, Austria

Abstract

While formal verification techniques are inevitable to ensure safety of critical cyber-phyical systems (CPS), engineering techniques to support the *design* and *analysis* of such CPS are still in their infancy. Therefore, we take a first step towards the provision of appropriate engineering techniques for CPS verification, by providing an extensive evaluation of the current state of the art, identifying challenges not yet tackled by existing approaches and by proposing a research roadmap intended to pave the way towards a fully supported engineering process for CPS verification models.

CCS Concepts

• Computer systems organization → Embedded and cyber-physical systems; **• Computing methodologies → Model verification and validation**.

ACM Reference Format:
Andreas Müller, Stefan Mitsch, Werner Retschitzegger, and Wieland Schwinger. 2020. Towards CPS Verification Engineering. In *The 22nd International Conference on Information Integration and Web-based Applications & Services (iiWAS '20), November 30-December 2, 2020, Chiang Mai, Thailand.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3428757.3429146

1 Introduction

The high safety standards imposed on cyber-physical systems (CPS) urgently require *formal verification techniques* (e.g., deductive verification [49] or reachability analysis [16]). Safety verification at scale calls for component-based *hybrid system models* [42], combining the hybrid nature of discrete controllers and physical processes, along with contracts describing *initial system states* (e.g., vehicle stopped initially) and *desired target states* (e.g., vehicle does not collide).

Problem. To address the complexity of verification, *design techniques* for obtaining more manageable components based on *abstraction* (i.e., reducing model details) and *decomposition* (i.e., partitioning models) are essential [15]. In order to ensure that these components still represent the relevant characteristics of the CPS under construction, additional *analytic techniques* are crucial, not least since formal verification is costly due to human interaction. These techniques being derived from the fundamental activities in SW engineering (SWE) [52] are central in a *CPS verification model engineering process* and comprise three phases (cf. Fig. 1): (i) a *modeling phase* in terms of decomposition driven by partition and reuse based on a system description leading to a component-based model, (ii) a *validation phase* leading, in a model-driven and data-driven way, to a validated model and (iii) a *verification phase* leading to a verified model which finally ends up, on basis of appropriate model transformations, in an executable controller of the CPS in question.

CPS Verification Engineering vs. SWE. Traditional SWE techniques are, however, inapplicable without severe adaptations, as engineering of CPS verification models differs in several key aspects [32]: Goal of the process is not a comprehensive system model, but a model that can be formally verified. As a result, processed artifacts (i) comprise intertwined discrete and continuous dynamics (*hybrid system models*), (ii) are highly non-deterministic, (iii) are vigorously abstracted to facilitate verification, (iv) target all system relevant actors, not only the ones being engineered (e.g., obstacle-avoiding robot and obstacles), and (v) must be unambiguously defined following explicitly defined formal semantics. Lack of support in current CPS engineering techniques for these unique features makes it far from trivial to create *valid component-based hybrid system verification models*, and presents one of the biggest bottlenecks for applying formal methods to CPS [51].

Contribution and Paper Structure. Taking a first step towards the provision of appropriate engineering techniques for CPS verification, the contribution of this paper is as follows. Section 2 provides an extensive evaluation of the current state of the art, based on a systematic set of criteria and identifies challenges not yet tackled by existing approaches. Section 3 proposes a research roadmap in order to overcome these challenges and finally, Section 4 concludes the paper with a brief resumé.

This material is based on research sponsored by the Austrian Science Fund (FWF) P28187-N31 and by the AFOSR under grant number FA9550-16-1-0288. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

iiWAS '20, November 30-December 2, 2020, Chiang Mai, Thailand
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8922-8/20/11.
https://doi.org/10.1145/3428757.3429146

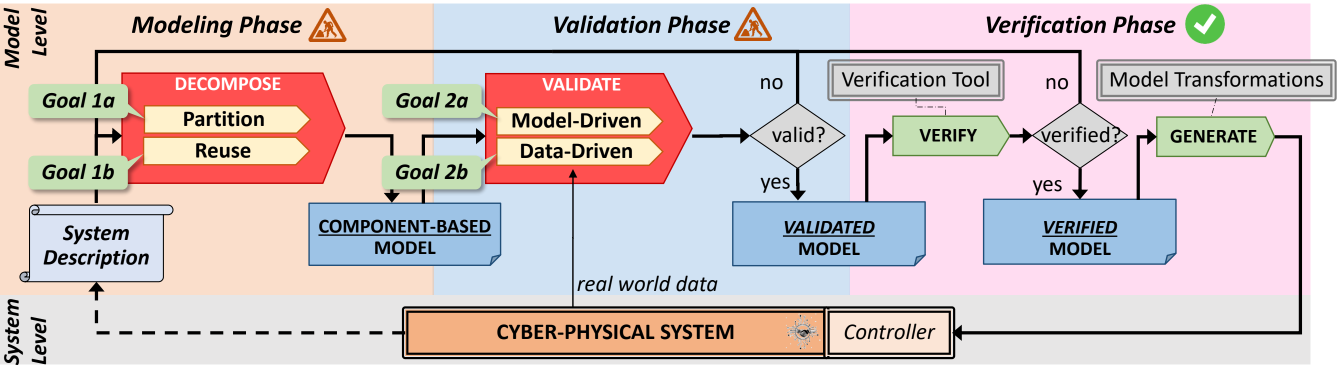


Figure 1: CPS verification model engineering process: modeling, validation, verification

2 State of Research

In order to give an in-depth overview about the state of research, approaches tackling different aspects of verification engineering of CPS are evaluated on basis of a systematic set of comparison criteria (see Fig. 2) which were carefully assembled in a bottom-up manner on basis of the characteristics of the different approaches. As intertwined cyber- and physical aspects are inherent to CPS, we did not consider solely discrete or continuous approaches (e.g., [21], supporting validation of continuous power systems), but rather focus on related work covering engineering of both aspects, usually formalized in *hybrid system models*. In the following, we first start with a brief discussion of the wider research context for engineering CPS verifications, followed by an evaluation of approaches for *decomposition* and finally *validation* of verification models.

2.1 CPS Engineering & Component-based SWE

CPS engineering is increasingly adopting paradigms from traditional *component-based software engineering* (CBSE) [52, 55]. Despite inherent differences as already discussed above, CBSE provides valuable input like component identification (e.g., [13, 33]), component reuse (e.g., [14]), code search engines (e.g., [5, 22, 50]), pragmatic software reuse (e.g., [20, 36]) and various software metrics (e.g., [22, 28, 29, 39, 53]). However, compositional formalisms necessary for describing desired overall behavior of CPS for verification are less common and often limited to linear continuous dynamics (e.g., assume-guarantee reasoning for hybrid automata [2, 15, 19]).

2.2 Decomposition of Verification Models

Approaches targeting the decomposition of verification models are evaluated in the following by distinguishing between those, acting in a *top-down* manner by *partitioning* the system into parts and those adhering to a *bottom-up* strategy by *reusing* existing components from a library such that they comprise the system behavior.

2.2.1 Partitioning-based Approaches.

Kekatos et al. [27] translate Simulink models to SpaceEx verification models building a network of hybrid automata. Simulink blocks containing non-linearities are over-approximated by piecewise affine approximations, possibly resulting in numerous locations (e.g., sinus approximated by 40 locations). Further partitioning besides over-approximation is not supported.

Kebir et al. [26] identify components from object-oriented code with two bottom-up decompositions: (i) automated clustering with a fitness function based on coupling and cohesion, and (ii) user-guided component identification of desired size around pre-selected key entities. Additionally, heuristics identify interfaces (e.g., based on method cohesion). Despite its focus on discrete software only, some techniques like clustering, key entities or heuristics provide a valuable input for further research regarding the partitioning of verification models for hybrid systems.

Oehlerking and Theel [46] automatically decompose hybrid automata into strongly connected components preserving stability (i.e., minor changes of input values lead to appropriate changes in output values) rather than safety, with extensions to stochastic hybrid systems [47]. They, however, neither consider reuse, nor specify or decompose external behavior.

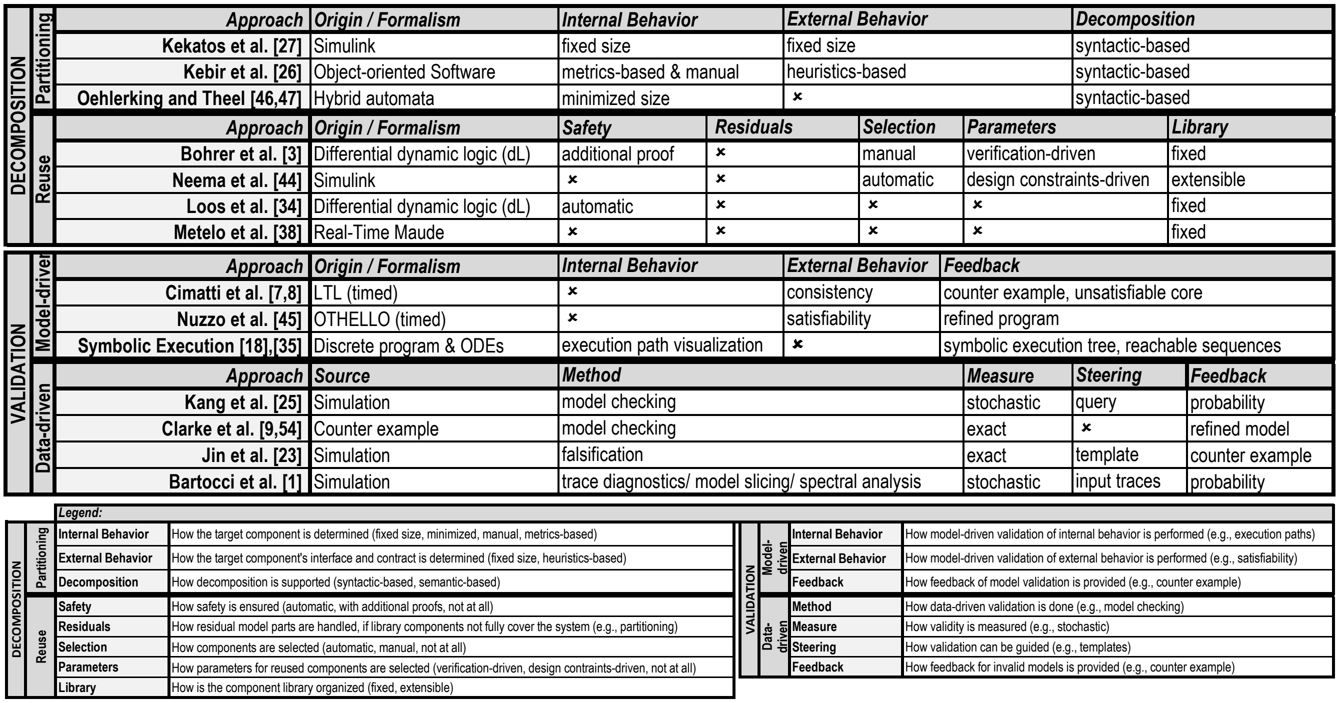
Summary. Current approaches do not focus on (i) verification models, and (ii) partitioning of internal or external component behavior based on formal semantics with verification complexity in mind.

2.2.2 Reuse-based Approaches.

Bohrer et al. [3] build hybrid systems models of roller coasters from components of individual track pieces. Parametric line segments/arcs are pre-verified to support certain properties (e.g., bounds on max. acceleration) and can be repeatedly instantiated manually to build a 2D model of tracks. There is, however, no focus on supporting automatic component selection, i.e., in their terms, to find lines/arcs that together model a track spline, but their components could serve as evaluation benchmark for further research.

Neema et al. [44] propose component-based model synthesis by design-space exploration. Components are put in a refinement hierarchy (i.e., design space), each providing several implementations. Goal is to find instances that form complete coverage of the system, obeying the hierarchy and design constraints (e.g., max. implementation cost). Although not focused on verification models and safety properties, its methods for finding specific implementations in large libraries turn out useful.

Loos et al. [34] focus on reuse of models and prove safety of a distributed car control system using quantified differential logic [48], so verify a property for any number of cars on a highway (e.g., avoid



		Approach	Origin / Formalism	Internal Behavior	External Behavior	Decomposition
DECOMPOSITION	Partitioning	Kekatos et al. [27]	Simulink	fixed size	fixed size	syntactic-based
		Kebir et al. [26]	Object-oriented Software	metrics-based & manual	heuristics-based	syntactic-based
		Oehlerking and Theel [46,47]	Hybrid automata	minimized size	✗	syntactic-based

		Approach	Origin / Formalism	Safety	Residuals	Selection	Parameters	Library
DECOMPOSITION	Reuse	Bohrer et al. [3]	Differential dynamic logic (dL)	additional proof	✗	manual	verification-driven	fixed
		Neema et al. [44]	Simulink	✗	✗	automatic	design constraints-driven	extensible
		Loos et al. [34]	Differential dynamic logic (dL)	automatic	✗	✗	✗	fixed
		Metelo et al. [38]	Real-Time Maude	✗	✗	✗	✗	fixed

		Approach	Origin / Formalism	Internal Behavior	External Behavior	Feedback
VALIDATION	Model-driven	Cimatti et al. [7,8]	LTL (timed)	✗	consistency	counter example, unsatisfiable core
		Nuzzo et al. [45]	OTHELLO (timed)	✗	satisfiability	refined program
		Symbolic Execution [18],[35]	Discrete program & ODEs	execution path visualization	✗	symbolic execution tree, reachable sequences

		Approach	Source	Method	Measure	Steering	Feedback
VALIDATION	Data-driven	Kang et al. [25]	Simulation	model checking	stochastic	query	probability
		Clarke et al. [9,54]	Counter example	model checking	exact	✗	refined model
		Jin et al. [23]	Simulation	falsification	exact	template	counter example
		Bartocci et al. [1]	Simulation	trace diagnostics/ model slicing/ spectral analysis	stochastic	input traces	probability

Legend:

DECOMPOSITION	Partitioning	Internal Behavior	How the target component is determined (fixed size, minimized, manual, metrics-based)
		External Behavior	How the target component's interface and contract is determined (fixed size, heuristics-based)
		Decomposition	How decomposition is supported (syntactic-based, semantic-based)
	Reuse	Safety	How safety is ensured (automatic, with additional proofs, not at all)
		Residuals	How residual model parts are handled, if library components not fully cover the system (e.g., partitioning)
		Selection	How components are selected (automatic, manual, not at all)
		Parameters	How parameters for reused components are selected (verification-driven, design contraints-driven, not at all)
		Library	How is the component library organized (fixed, extensible)

VALIDATION	Model-driven	Internal Behavior	How model-driven validation of internal behavior is performed (e.g., execution paths)
		External Behavior	How model-driven validation of external behavior is performed (e.g., satisfiability)
		Feedback	How feedback of model validation is provided (e.g., counter example)
	Data-driven	Method	How data-driven validation is done (e.g., model checking)
		Measure	How validity is measured (e.g., stochastic)
		Steering	How validation can be guided (e.g., templates)
		Feedback	How feedback for invalid models is provided (e.g., counter example)

Figure 2: State of research

crashes). Similarly, **Metelo et al. [38]** target the n-reservoir problem, where water tank components constantly drain water filled by a single hose, assuring a min. water level verified after composition using linear hybrid automata. However, the approaches are strictly limited to components of uniform shape and a fixed component structure—the user merely chooses the number of components (i. e., components cannot be composed or reused in later models).

Summary. Current approaches (i) are restricted to specific scenarios with limited libraries of components having fixed and uniform structure, (ii) mostly select component candidates manual, while composition is partly automated, and (iii) do not focus on incomplete component coverage and the resulting residual models.

2.3 Validation of Verification Models

Even after decomposition, formal verification is costly, often requiring human guidance. To prevent futile or vacuous proofs, prior *validation techniques* can check if a model adequately represents reality and is fit for verification. In the following, validation approaches are distinguished weather they ensure that the model (i) satisfies *modeling requirements*, i.e., analyze *how* the model is expressed (e.g., contradiction-free), i.e., *model-driven* approaches and (ii) satisfies *domain requirements*, i.e., captures *what* the model has to express (i.e., system under consideration), i.e., *data-driven* approaches.

2.3.1 Model-Driven Approaches.

Cimatti et al. [7, 8] introduce the OTHELLO language to automatically validate system behavior requirements with an SMT solver by integrating temporal logic with hybrid aspects. Besides counterexamples, diagnostic information w.r.t. failed validation attempts is provided, comprising an unsatisfiable core to identify the causal requirements. Even though formalized requirements can be interpreted as specifications of desired system behavior, the approach is limited to temporal properties and does not aim at validation of hybrid models and associated contracts.

Nuzzo et al. [45] present the CHASE framework for CPS requirement engineering by discrete time models, components and contracts. CHASE bases on linear temporal logic (LTL) to analyze consistency and compatibility of contracts. An external tool validates a specified property and synthesizes a Python implementation from consistent specifications. The validation of hybrid system components and contracts with continuous dynamics is not an issue.

Hentschel et al. [18] propose a symbolic execution approach for discrete Java programs, allowing validation of their behavior by full execution path exploration [4, 30]. Similarly, **Majumdar et al. [35]** present a symbolic execution approach for closed-loop CPS, by discretizing the behavior of a linear plant, yielding a discrete-time sequence of states instead of continuous evolution. However, neither approach focuses on hybrid systems verification models. Rather they act as potential starting points for a model-driven validation approach.

Summary. Current approaches (i) mostly focus on contract validation based on designated specification formalization, but (ii) do not focus on verification models that comprise internal and external behavior. Feedback varies by approaches by giving insight into possible response directions. Finally, symbolic execution approaches do not focus on hybrid system verification models, and their symbolic execution trees require adaptation to CPS.

2.3.2 Data-Driven Approaches.

Kang et al. [25] use UPPAAL-SMC [12], a statistical model checking tool for timed systems, to check system properties with limited degree of confidence by monitoring simulation runs in the automotive domain. Validation feedback depends on queries (e.g., probability estimate and comparison), guiding the process. They, however, do not use external data (e.g., from a dedicated simulation model) to validate if a hybrid verification model behaves as intended.

Clarke et al. [9], Stursberg et al. [54] present a counterexample-guided abstraction refinement approach for hybrid systems by creating abstractions thereof and validating that they actually behave as the original system. If so, it suffices to verify the abstraction. Spurious counterexamples (do not fulfill specification, but are not accepted by the original model) result in a model refinement, which is possible since all locations of the original model are mapped onto single states in the abstraction. Their validation of models, however, is limited to closely related models and abstractions, and does not consider independent data that cannot be directly mapped to the original model.

Jin et al. [23] target model specification synthesis and simulation for data-driven validation. A manually provided template formula in temporal logic, expressing the target property (e.g., start moving in y seconds with speed x) together with the simulation data is used to find, based on falsification, values satisfying the template, thus fulfilling the target property. With this approach it is, however, not possible to ensure that a model behaves as indicated by data obtained from an external source.

Bartocci et al. [1] use a three step approach to localize faults in Simulink/Stateflow models: (i) simulate the model guided by tests containing input traces and localize erroneous parts, (ii) use trace diagnostic to identify causal variables, and (iii) use these variables to find components potentially involved in specification violation. Additional use of external data is not tackled in their approach.

Summary. Current data-driven validation approaches (i) mostly base their analysis on data created by simulation of the model itself and (ii) are not focused on ensuring that the targeted model actually behaves as intended based on external data, but rather that it fulfills specific defined properties. This goes inline when taking a look into current *CPS simulation* approaches (e.g., [24, 31]), requiring manual behavior inspection, instead of providing automated techniques comparing simulations/data with the modeled behavior to ensure that a verification model behaves as desired.

3 Research Roadmap

The following roadmap is intended to address the challenges in the modeling- and validation phase of the engineering process of Fig. 1 as identified in our state-of-the-art survey in the previous section. Overall, our findings have revealed that, while the *verification phase* is well supported through formal verification tools (e.g., [17]) and proof-aware model transformation methods (e.g., [43]), further research is necessary in order to facilitate *modeling* and *validation* by putting forward (i) *design techniques* for *decomposition of verification models* and (ii) *analytic techniques* for their *validation* to support the *entire* CPS verification model engineering process. In particular, the following research goals and according challenges are considered to be of major importance:

Goal 1a: Partitioning Mechanisms for Decomposition. Techniques to partition system behavior should be developed which are able to deal with both, implicit and explicit links within the intertwined discrete controllers and physical processes. Complementing these techniques, an appropriate contract mechanism is needed in order to facilitate formal verification of the resulting hybrid system components while preserving overall system correctness.

Goal 1b: Reuse Mechanisms for Decomposition. Techniques to identify pre-verified components are necessary which comprise parts of the required system behavior. In addition, smart selection mechanisms have to be provided for component coupling and for matching component parameters such that system safety is provably preserved. Finally, component and contract candidates should be at least semi-automatically outlined for the residual model.

Challenges. It has to be emphasized that, identifying component candidates is especially crucial, since the goal is not only to facilitate reuse with increased cohesion and reduced coupling, but also to reduce verification complexity; this simultaneously asks for (i) substantial adjustments of component contracts to support inter-component communication, (ii) *explicit links* (e.g., communication, sensing) and *implicit links* through physical phenomena (e.g., time, motion), and (iii) strict formal requirements (e.g., no overlapping component variables). Naturally, a necessary prerequisite for reuse mechanisms are *component libraries*, which have been introduced in previous work, e.g., [6] (collection of Simulink models) and our hybrid systems component library targeting various domains, like traffic control [40], vehicle cruise control [41], robotics and train control [42].

Goal 2a: Model-driven Validation Mechanisms. Techniques that expose the hybrid discrete-continuous model behavior during a proof should be put forward in order to, e.g., visualize model traces and map model elements to proof steps to foster user interaction. Moreover, concepts are needed allowing to analyze model quality attributes, absent additional external data, to determine, e.g., if certain assumptions are satisfiable.

Goal 2b: Data-driven Validation Mechanisms. Techniques to compare models to external data for compliance with domain requirements have to be developed, allowing, e.g., real-world test data and simulations. These techniques should subsequently also allow to localize and improve unsatisfactory model parts.

Challenges. Checking *modeling requirements* demands comprehensive analysis of the model's internal and external behavior, challenged by the infinite number of system states arising from the combined cyber- and physical behavior. Checking *domain requirements* must base on supplemental CPS data (e.g., obtained by simulation or testing), as verification models tend to be vigorously abstracted. Such data is, however, discrete in nature (e.g., recorded at discrete time intervals), requiring to bridge the gap to hybrid verification models by discovering how discretized data can be mapped to the component behavior while retaining its contracts, facilitating error localization in invalid models. Techniques for fast comparisons between a model and a system, can be found in the field of artificial intelligence, where *hybrid system diagnosis* [37] aims at discovering errors in systems by comparing the observed behavior of a system to a predicted behavior described by a model (e.g., [10, 11]).

4 Resumé

With the research goals presented in this paper, we hope to: (i) conceptualize novel design and analytic engineering techniques to reduce verification effort, partly based on existing SWE techniques being vigorously extended towards the characteristics of CPS, (ii) demonstrate applicability with a prototypical tool suite supporting these techniques, and (iii) gain insights into the relationship between modularity, reusability, and functional suitability in CPS verification.

References

[1] Ezio Bartocci et al. 2018. Localizing Faults in Simulink/Stateflow Models with STL. In *Proc. Hybrid Systems: Computation and Control Conf.* ACM, 197–206.

[2] Sergiy Bogomolov et al. 2014. Assume-Guarantee Abstraction Refinement Meets Hybrid Systems. In *Hardware and Software: Verification and Testing - 10th Int. Haifa Verification Conf. (LNCS)*, Eran Yahav (Ed.). Springer, 116–131.

[3] Rose Bohrer et al. 2018. CoasterX: A Case Study in Component-Driven Hybrid Systems Proof Automation. In *6th IFAC Conf. on Analysis and Design of Hybrid Systems (IFAC-PapersOnLine)*, Alessandro Abate et al. (Eds.). Elsevier, 55–60.

[4] Robert S. Boyer et al. 1975. SELECT - A Formal System for Testing and Debugging Programs by Symbolic Execution. *SIGPLAN Not* 10, 6 (1975), 234–245.

[5] Gianluigi Caldiera and Victor R. Basili. 1991. Identifying and Qualifying Reusable Software Components. *IEEE Computer* 24, 2 (1991), 61–70.

[6] Shafiul Azam Chowdhury et al. 2018. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In *Proc. Conf. on Softw. Eng.* ACM.

[7] Alessandro Cimatti et al. 2009. Requirements Validation for Hybrid Systems. In *Computer Aided Verification Conf. (LNCS)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer.

[8] Alessandro Cimatti et al. 2012. Validation of requirements for hybrid systems: A formal approach. *ACM Trans. Softw. Eng. Methodol.* 21, 4 (2012), 22:1–22:34.

[9] Edmund M. Clarke et al. 2003. Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement. In *Tools and Algorithms for the Construction and Analysis of Systems Conf. (LNCS)*, Hubert Garavel and John Hatcliff (Eds.). Springer.

[10] Matthew J. Daigle et al. 2015. A Structural Model Decomposition Framework for Hybrid Systems Diagnosis. In *Proc. of the 26th Int. Workshop on Principles of Diagnosis*. CEUR-WS.org, 201–208.

[11] Matthew J. Daigle et al. 2018. Diagnosis of Hybrid Systems Using Structural Model Decomposition. In *Fault Diagnosis of Hybrid Dynamic and Complex Systems*, Moamar Sayed-Mouchaweh (Ed.). Springer, Cham, 179–207.

[12] Alexandre David et al. 2011. Time for Statistical Model Checking of Real-Time Systems. In *Computer Aided Verification - 23rd Int. Conf., Proc. (LNCS)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, 349–355.

[13] Dejan Desovski and Bojan Cukic. 2007. A Component-Based Approach to Verification and Validation of Formal Software Models. In *Architecting Dependable Systems IV (LNCS)*. Springer.

[14] William B. Frakes and Kyo Kang. 2005. Software Reuse Research: Status and Future. *IEEE Trans. Software Eng.* 31, 7 (2005), 529–536.

[15] Goran Frehse et al. 2004. Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In *IEEE Conf. on Decision and Control, CDC*, Vol. 1.

[16] Goran Frehse et al. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Computer Aided Verification Conf. Proc. (LNCS)*. Springer.

[17] Nathan Fulton et al. 2015. KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In *Conf. on Automated Deduction, Proc. (LNCS)*. Springer.

[18] Martin Hentschel et al. 2014. Symbolic Execution Debugger (SED). In *Runtime Verification, Proc. (LNCS)*. Springer.

[19] Thomas A. Henzinger et al. 2001. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *Hybrid Systems: Computation and Control, Proc. (LNCS)*, Vol. 2034. Springer.

[20] Reid Holmes and Robert J. Walker. 2012. Systematizing pragmatic software reuse. *ACM Trans. Softw. Eng. Methodol.* 21, 4 (2012), 20:1–20:44.

[21] Zhenyu Huang et al. 2006. Model validation with hybrid dynamic simulation. In *2006 IEEE Power Engineering Society General Meeting*.

[22] Oliver Hummel. 2008. *Semantic component retrieval in software engineering*. Ph.D. Dissertation. Univ. of Mannheim, Germany. https://madoc.bib.uni-mannheim.de/1883/

[23] Xiaoqing Jin et al. 2013. Mining requirements from closed-loop control models. In *Hybrid systems: computation and control, Proc.* ACM.

[24] T. Junjie et al. 2012. Cyber-physical systems modeling method based on Modelica. In *2012 IEEE Int. Conf. on Software Security and Reliability Companion*. 188–191.

[25] Eun-Young Kang et al. 2017. Verification and Validation of a Cyber-Physical System in the Automotive Domain. In *IEEE Conf. on Softw. Quality, Reliability and Security Companion*. IEEE.

[26] Selim Kebir et al. 2012. Quality-Centric Approach for Software Component Identification from Object-Oriented Code. In *Joint IEEE/IFIP Conf. on Softw. Architecture & Europ. Conf. on Softw. Architecture*. IEEE.

[27] Nikolaos Kekatos et al. 2017. Constructing verification models of nonlinear Simulink systems via syntactic hybridization. In *IEEE Conf. on Decision and Control*. IEEE.

[28] Marcus Kessel and Colin Atkinson. 2015. Measuring the Superfluous Functionality in Software Components. In *ACM SIGSOFT Symp. on Component-Based Softw. Eng.*, Philippe Kruchten et al. (Eds.). ACM.

[29] Marcus Kessel and Colin Atkinson. 2016. Ranking software components for reuse based on non-functional properties. *Inf. Sys. Frontiers* 18, 5 (2016), 825–853.

[30] James C. King. 1976. Symbolic Execution and Program Testing. *Commun. ACM* 19, 7 (1976), 385–394.

[31] Xenofon D. Koutsoukos et al. 2018. SURE: A Modeling and Simulation Integration Platform for Evaluation of Secure and Resilient Cyber-Physical Systems. *IEEE Proc.* 106, 1 (2018).

[32] E.A Lee. 2008. Cyber Physical Systems: Design Challenges. In *IEEE Symp. on Object Oriented Real-Time Distrib. Comp. (ISORC 2008)*.

[33] Keith Levi and Ali Arsanjani. 2002. A goal-driven approach to enterprise component identification and specification. *Commun. ACM* 45, 10 (2002), 45–52.

[34] Sarah M. Loos et al. 2011. Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified. In *Symposium on Formal Methods (LNCS)*. Springer.

[35] Rupak Majumdar et al. 2012. CLSE: Closed-Loop Symbolic Execution. In *NASA Formal Methods Symp. Proc. (LNCS)*. Springer.

[36] Soha Makady and Robert J. Walker. 2013. Validating pragmatic reuse tasks by leveraging existing test suites. *Softw., Pract. Exper.* 43, 9 (2013), 1039–1070.

[37] Sheila A. McIlraith et al. 2000. Hybrid Systems Diagnosis. In *Hybrid Systems: Computation and Control*. Springer.

[38] Andre Metelo et al. 2018. Towards the Modular Specification and Validation of Cyber-Physical Systems: A Case-Study on Reservoir Modeling with Hybrid Automata. In *Computational Science and Its Applications (LNCS)*. Springer.

[39] Marko Mijač and Zlatko Stapic. 2015. Reusability Metrics of Software Components: Survey. In *Centrl. Europ. Conf. on Inform. and Intell. Sys.* Univ. Zagreb.

[40] Andreas Müller et al. 2015. Verified Traffic Networks: Component-based Verification of Cyber-Physical Flow Systems. In *18th Int. Conf. on Intelligent Transportation Systems*. 757–764.

[41] Andreas Müller et al. 2016. A Component-Based Approach to Hybrid Systems Safety Verification. In *Integrated Formal Methods - 12th Int. Conf., Proc. (LNCS)*, Erika Ábrahám and Marieke Huisman (Eds.). Springer, 441–456.

[42] Andreas Müller et al. 2018. Tactical contract composition for hybrid system component verification. *STTT* 20, 6 (2018), 615–643.

[43] Andreas Müller, Stefan Mitsch, Wieland Schwinger, and André Platzer. 2018. A Component-Based Hybrid Systems Verification and Implementation Tool in KeYmaera X (Tool Demonstration). In *Cyber Physical Systems. Model-Based Design - 8th International Workshop. Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 11615. Springer, 91–110.

[44] Sandeep Neema et al. 2003. Constraint-Based Design-Space Exploration and Model Synthesis. In *Embedded Software, Conf. (LNCS)*, Rajeev Alur and Insup Lee (Eds.). Springer.

[45] Pierluigi Nuzzo et al. 2018. CHASE: Contract-based requirement engineering for cyber-physical system design. In *Design, Automation & Test in Europe Conf. & Exhib.* IEEE.

[46] Jens Oehlerking and Oliver E. Theel. 2009. Decompositional Construction of Lyapunov Functions for Hybrid Systems. In *Hybrid Systems: Computation and Control, Proc. (LNCS)*. Springer.

[47] Jens Oehlerking and Oliver E. Theel. 2009. A Decompositional Proof Scheme for Automated Convergence Proofs of Stochastic Hybrid Systems. In *Automated Technology for Verification and Analysis, Proc. (LNCS)*. Springer.

[48] André Platzer. 2012. A Complete Axiomatization of Quantified Differential Dynamic Logic for Distributed Hybrid Systems. *Logical Methods in Computer Science* 8, 4 (2012).

[49] André Platzer. 2018. *Logical foundations of cyber-physical systems*. Springer, Cham, Switzerland.

[50] Steven P. Reiss. 2009. Semantics-based code search. In *Conf. on Softw. Eng., Proc.* IEEE, 243–253.

[51] Kristin Yvonne Rozier. 2016. Specification: The Biggest Bottleneck in Formal Methods and Autonomy. In *Verified Software. Theories, Tools, and Experiments Conf. (LNCS)*.

[52] Ian Sommerville. 2011. *Software engineering* (9. ed.). Pearson, Boston, MA.

[53] Kathryn T. Stolee et al. 2016. Code search with input/output queries: Generalizing, ranking, and assessment. *Journal of Systems and Software* 116 (2016), 35–48.

[54] Olaf Stursberg et al. 2003. Specification-Guided Analysis of Hybrid Systems Using a Hierarchy of Validation Methods. *IFAC Proc. Volumes* 36, 6 (2003), 289–294.

[55] Tassio Vale et al. 2016. Twenty-eight years of component-based software engineering. *Journal of Systems and Software* 111 (2016), 128–148.