On the State of the Art in Verification and Validation in Cyber Physical Systems

Xi Zheng, Christine Julien, Miryung Kim, Sarfraz Khurshid

The Center for Advanced Research in Software Engineering
Department of Electrical and Computer Engineering,
The University of Texas at Austin
{jameszhengxi, c.julien}@utexas.edu, {miryung, khurshid}@ece.utexas.edu

TR-ARiSE-2014-001

ARiSE
Advanced Research in Software Engineering

On the State of the Art in Verification and Validation in Cyber Physical Systems

Xi Zheng, Christine Julien, Miryung Kim, Sarfraz Khurshid
The Center for Advanced Research in Software Engineering
Department of Electrical and Computer Engineering, The University of Texas at Austin
{jameszhengxi, c.julien}@utexas.edu, {miryung, khurshid}@ece.utexas.edu

ABSTRACT

It is widely held that debugging cyber-physical systems (CPS) is challenging. However, few empirical studies quantitatively and qualitatively capture the state of the art and the state of the practice in debugging CPS and analyze what major research gaps remain. This paper presents an empirical study of verification and validation in CPS through three complementary methods: a structured on-line survey of CPS developers and researchers, semi-structured interviews with professional CPS developers from various backgrounds, and a qualitative analysis of state of the art in research related to CPS testing. We find that traditional verification and validation methodologies are not sufficient for cyber-physical systems, and we identify several potential avenues for future work. Our key findings include: (i) many CPS developers do not use traditional verification and validation methodologies and rely heavily on trial and error; (ii) simulation alone is not enough to capture dangerous bugs in CPS; (iii) it is widely acknowledged that the main challenges in CPS debugging are related to models of software systems, models of physics, and integration of cyber and physics models. These findings aid in identifying research directions to address the identified key challenges in CPS verification and validation.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Real-time and embedded systems; D.2.4 [**Software/Program Verification**]: Assertion checkers, Validation

Keywords

cyber-physical systems; verification & validation; debugging

1. INTRODUCTION

Cyber-Physical Systems (CPS) feature a tight coupling between physical processes and software components [44] and execute in varying spatial and temporal contexts exhibiting multiple behavioral patterns between runs [81]. CPS is widely used in biomedical and healthcare systems, autonomous vehicles, smart grid, and many other industrial applications [44, 64, 81]. Over the years, systems and control engineers have made significant progress in developing system science and engineering methods and tools (e.g., time and frequency domain methods, state space analysis, filtering, prediction, optimization, robust control, and stochastic control) [7]. In parallel, computer science and software engineering researchers have made breakthroughs in software verification and validation (e.g., systematic testing, formal methods). However, as our studies in this paper demonstrate, guaranteeing the correctness of CPS remains an astounding challenge.

There are insufficient methods for investigating the impact of the environment, or *context*, on a CPS [72]. External conditions, which are often hard to predict, can invalidate estimates (even worst-case ones) of the safety and reliability of a system. Modeling any CPS is further hampered by the complexity of modeling *both* the cyber world (e.g., software, network, and computing hardware) and the physical environment (physical processes and their interactions) [48]. Simplified models failing to anticipate that components fail dependently are easily invalidated by the interdependencies among the cyber and physical.

In a 2007 DARPA Urban Challenge Vehicle, a bug undetected by more than 300 miles of test-driving resulted in a near collision. An analysis of the incident found that, to protect the steering system, the interface to the physical hardware limits the steering rate to low speeds [55]. When the path planner produces a sharp turn at higher speeds, the vehicle physically cannot follow, and this unpredicted situation caused the bug. This analysis also concluded that, although simulation-centric tools are indispensable for rapid prototyping, design, and debugging, these tools are limited in providing correctness guarantees. In some mission-critical industries (e.g., medical devices), correctness is satisfied by the documentation for code inspections, static analysis, module-level testing, and integration testing [38]. These tests are largely open-loop tests that do not consider the context of the patient [38]. Such a lack of true correctness guarantees could easily cause the Therac-25 disaster [46] to reoccur.

These challenges both in the research community and industry practice motivated us to conduct a survey and follow-up interviews with CPS experts to discover the state of the art and practice in tackling challenges in verification and validation of CPS. We address the following specific research questions: (1) What is the definition of software verification and validation from CPS developers' perspectives? (2) How familiar are CPS developers with traditional verification and validation methodologies and how do CPS developers use these methodologies? (3) What are the state of art methodologies in the verification and validation of CPS, in which scenarios do CPS developers find each of them most useful, and what are their limitations? (4) What are the main open challenges in verification and validation of CPS?

We conducted a survey with twenty-five CPS experts (Section 3); we interviewed five of these participants in more depth (Section 4). Our studies found the following:

- Many participants are not familiar with the concept of verification and validation, and half do not use traditional verification and validation methodologies (e.g., code coverage), which they found not applicable to CPS.
- Many participants use high level programming languages to develop CPS applications; resource constraints (in terms of CPU, memory and storage) are *not* a significant concern for CPS developers.
- Many participants are aware of formal methods but do not apply them much due to state explosion, a lack of bug finding support, and modeling issues regarding temporal aspects and physics.
- Simulation is commonly used in CPS; however, simulation tools are inaccurate and are therefore used mainly in prototyping during earlier stages of design.
- Trial and error is the main debugging methodology used by CPS developers though it suffers from incomplete coverage and is very expensive in terms of time and resources.
- Key open challenges in CPS verification and validation remain in tailoring models of software and models of physics,[1] and integration of these models with each other and into the verification and validation process.

While there are many anecdotes relating the challenges in verification and validation in CPS, empirical studies have not systematically assessed the research gaps in this domain comprehensively, unlike our study. To the best of our knowledge, our study is the first to quantitatively assess the state of the art and the state of the practice in this specific area; Our study also sets out to confirm or dispel some widely held notions regarding the development and debugging of CPS (Section 5). we couple our survey and interviews with an in depth literature review (Section 2), and the combination provides strong evidence of the above findings. Based on our study, we propose future research directions for verification and validation of CPS (Section 5).

2. STATE OF THE ART IN CPS VERIFICATION AND VALIDATION

Using models to design complex systems is the *de facto* standard in engineering disciplines [70]. In the context of CPS, similarly styled models provide a foundation to specify and verify correct system behaviors and are used as the basis of complex simulation tools, which are commonly used to assess a baseline for the behavior of a complex CPS. In this section, we review the state of the art from the models themselves, through formal CPS verification, and into simulation. We close with a look at more empirical approaches captured in CPS-targeted testing and debugging tools.

2.1 Models

A model of a CPS contains heterogeneous models of both dynamic physical processes and computations, networks, and software [23]. Due to close relationships between CPS and *hybrid systems*, many hybrid systems models are used in CPS either directly or through extensions [55].

Automata. The Automata model [36] represents the behavior of sequential systems without time constraints. Substantial later work has extended this idea into hybrid systems. *Timed automata* [4] and its derivatives capture the relative timing of events, enabling the specification of liveness, fairness, and non-determinism, which are all key concepts in CPS. *Input/Output Automata* [54] can capture asynchronous and partially synchronous systems, giving semi-autonomous components direct control over some actions, while also forcing them to be reactive to events in their surroundings. *Hybrid automata* [3] go a step further in expressively capturing both discrete and continuous dynamics. In a hybrid automata, a system is a finite automaton with a set of variables. At a given location, the values of the continuous variables change according to associated laws, and transitions between locations are protected by guarded assignments. The hybrid automata model and its extensions are widely used in hybrid systems modeling [9, 33, 53, 62]. While automata models are expressive and popular, from the perspective of everyday application developers, they are complex, unintuitive, and lack modularity and dynamic reconfiguration [82], which are all necessary for providing tool support to otherwise ordinary CPS application developers.

Petri nets. Petri nets [56] have been extended to hybrid systems, most notably through the addition of temporal aspects that associate time either with the places [73] or with the transitions [66]. In a *hybrid Petri net* [42], the state (i.e., values of the tokens) can be either discrete (e.g., integers) or continuous (e.g., real numbers). Though this is still far from capturing both discrete and continuous *dynamics*, it has served as a foundation for future models, e.g., *differential Petri nets* [22], in which the system's continuous dynamics can be at least partially represented via differential equations. In *hybrid stochastic Petri nets*, the discrete transitions may be immediate, stochastic, or deterministically timed. Petri net models tend to be more intuitive than automata models [19] while sharing similar expressiveness in terms of modeling capabilities for CPS. However, Petri net models still do not easily support modular analysis nor are they easily integrated with everyday CPS development.

Modeling languages. Modeling languages have a tendency to be more modular and more easily integrated into more user-facing tools. Introducing continuous statements (in the form of differential equations) into CSP leads to *hybrid CSP* [39]. Communications can assign values to continuous variables, potentially describing system evolution that is driven by events. Using hybrid CSP, one can model many key aspects of CPS, including synchronicity, timing of events, and discrete and continuous dynamics. CHARON [5] allows a hierarchical representation of hybrid systems, providing both faithful representations of discrete and continuous dynamics and a modular and compositional approach. CHARON does not support dynamic architectural reconfiguration, a limitation overcome by R-Charon [61]. SHIFT enables the specification and simulation of complex systems represented as hybrid automata; SHIFT captures the synchronicity and continuous dynamics of hybrid systems while also supporting compositional modeling and dynamic reconfiguration. The syntax is approachable (it is similar to C syntax), and SHIFT includes a compiler that can convert a SHIFT program into C code. However, none of the models consider some fundamental concepts in CPS (e.g., support for networking or performance analysis [81]).

Other models. Other models and modeling tools for hybrid systems provide inspirations for CPS. *Piecewise Deter-*

[1] In the realm of engineering of systems and machines, "systems models" is the appropriate term, but, to avoid confusion, we use "models of physics" in this paper.

ministic Markov Processes (PMDP) [52] and *Stochastic Hybrid Systems* (SHS) [37], for example, address uncertainty by brining randomness into either discrete transitions (in PDMP) or continuous states (SHS). *Autonomous Semantic Agents* (ASA) [26, 49] can capture semantic relationships among components in both the cyber and physical worlds and interdependencies across the cyber-physical boundary.

2.2 Formal Verification

It is exceedingly difficult to prove properties of CPS automatically because of the disconnect between formal techniques for the cyber portions and well-established engineering techniques for the physical pieces [17, 60]. However, significant progress has been made in formal verification that has the potential to change this landscape for CPS.

Deductive verification. One of the earliest approaches to hardware verification and deductive verification (i.e., theorem proving) uses formal logic for both implementation and specification [69]. PVS [58] has been used to create an inductive proof method for the parallel composition of hybrid systems [1]. STeP [10] has been used to prove correctness in hybrid systems using verification rules and diagrams that can automatically generate invariants. The μ-calculus has also been extended to reason about properties of hybrid systems, enabling deductive verification of hypotheses regarding a system's continuity or tolerance [20]. *Dynamic Logic* (DL) [30] is a rich formal system for reasoning about programs, and many variants have been developed for hybrid systems. For instance, *Quantified Differential Dynamic Logic* (QDDL) [63] introduces *quantified hybrid programs*, with quantified assignments and quantified differential equations systems for expressing distributed hybrid dynamics. The main disadvantage for applying deductive verification to CPS is that deriving a formal proof is overwhelmingly tedious [69]. This hurdle appears to be insurmountable among our practitioners as it requires domain experts with considerable mathematical experience. More critically, deductive verification cannot capture bugs that appear only at runtime (e.g., those that arise only when the software interacts with a physical environment).

Verification algorithms and model checking. The verification world is rife with highly capable model checkers. KRONOS [21] can verify whether a system modeled in timed automata satisfies given properties. UPPAAL [41], which has seen some use in verifying CPS, combines a symbolic verification algorithm to verify properties during state-space exploration, thereby ameliorating some of the scalability challenges that other model checkers experience. Real-Time PROMELA [79] allows specifications of concurrent systems to include information about clocks and time based on timed automata. HyTECH [32] is a symbolic model checker designed for linear hybrid automata that can compute parametric constraints to guarantee correctness, which is especially helpful for designing CPS applications. Probabilistic model checking (e.g., as in PRISM [34]) provides insight into stochastic effects of systems. Abstraction has been used to simplify hybrid systems to make them more amenable to model checking while maintaining sufficient detail to check critical properties [78]. Model reduction is widely used in control theory to eliminate spurious dynamic elements while identifying the key elements [18]. Though such a reduced order model makes the analysis tractable, error bounds are unquantified, which makes the verification unsafe. Model reduction techniques can be augmented by approximation that can provide bounds on the estimated error for the analysis [29]. The main strength of model checking is that verification can be fully automated. However, in general, model checking suffers from state-explosion, complexity in specifying input properties and creating the scripting environment, and inevitable loss of representativeness of code analyzed [8]. More relevantly, as with deductive verification, there are bugs in CPS that crop up only at run time and only in the presence of peculiarities of a particular physical deployment environment; such runtime bugs cannot be captured by model checking alone.

2.3 Simulation

From our on-line survey and interviews, we found that CPS developers use simulation widely but tend to build their own simulation tools targeting a specific domain or focus. General purpose simulation tools exist mostly for the physical parts of a CPS or for the networking components. LabVIEW [50], a graphical programming environment used by some CPS developers, comes with built-in simulation packages for simulating an analog computer, a continuous time process, or discrete events; the latter enables an interactive experimental environment to study processes with uncertainties. Many domain experts use Simulink [11], which provides solvers for modeling, simulating, and analyzing dynamic systems expressed in Matlab's weakly typed programming language. Breach [25] is a Matlab toolbox providing simulation-based techniques for the analysis of hybrid dynamic systems. Modelica [27] is a specialized object-oriented modeling language for complex and heterogeneous physical systems that supports ordinary differential equations, differential-algebraic questions, bond graphs, finite state automata, Petri nets, etc. Ptolemy [14] uses object-oriented syntax to model heterogeneous subsystems and to integrate these subsystems into a whole. HyVisual [45] is a simulator for continuous-time dynamic systems and hybrid systems that uses ordinary differential equations to define continuous dynamics and provides a numerical solver to simulate the dynamics. Test effectiveness can be improved by using simulation augmented with symbolic techniques, e.g., by combining numerical simulation with symbolic methods to compute a set of initial states that are equivalent to a given initial state and a simulation trajectory [6]. Co-simulation [28] allows the discrete (cyber) and continuous (physical) models to run on their respective simulators, managed by a coordinating co-simulation engine. All of these tools are in addition to the common discrete event simulators found in the networking community [31, 47, 80], which are also widely used by CPS developers. In general, while simulation models may provide very good representations of the real world and allow fast and cheap prototyping of CPS applications, they often fail to accurately represent the environment; these concerns are borne out by our studies reported in subsequent sections.

2.4 Testing and Debugging Tools

Though sensor networks and CPS are not exactly the same, several tools exist to support testing and debugging deployed wireless sensor networks, which provides some insight into directions and challenges for CPS. *Passive distributed assertions* [67] allows programmers to specify assertions that are preprocessed to generate instrumented code that passively transmits relevant messages as the assertions

Table 1: Summary of Survey Questions
(abbreviated; see `https://www.surveymonkey.com/s/MP7HP7W` for complete survey)

Background	What are your primary application domains of expertise (multiple selections allowed)? What are your primary roles (multiple selections allowed)? How many years of cyber-physical systems development experience do you have? What programming languages have you used in developing CPS applications?
Definition	What are the main differences between CPS and embedded systems? How do you define verification and validation (in general)?
Perceptions	Please rate agreement or disagreement (Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree) - Simulation alone is sufficient for verification and validation of CPS. - Formal methods for verification and validation of CPS is not tractable with respect to resources and time. - The state of the art of verification and validation of CPS involves repeatedly rerunning the system in a "live" deployment, observing its behavior, and tweaking the implementation (both hardware and software) to achieve the stated requirements.
Experience	What percentage of your work is devoted to verification and validation? How do you think code inspection can help with verification and validation? What testing methodologies do you employ during verification and validation of CPS? What model checker(s) do you employ during verification and validation of CPS? What simulation tools you used? Have you written assertions to aid in verification and validation of CPS?

are checked. Dustminer [40] collects system logs to look for sequences of events responsible for faulty interactions among sensors. CPS developers also need support for source-level debugging, for which a handful of tools exist in sensor networks. Clairvoyant [83] uses a debugger on each sensor node to instrument the binary code to enable GDB-like debugging behavior. MDB [76] provides the same style of behavior for *macroprograms* specified at the network level (instead of the node level). Envirolog [51] records all events labeled with programmer-provided annotations, allowing an entire execution trace to be replayed. *Declarative tracepoints* allows the programmer to insert checkpoints for specified conditions that occur at runtime. Sympathy [65] collects and analyzes a set of minimal metrics at a centralized sink node to enable fault localization across the distributed nodes. Finally, KleeNet [68] uses symbolic execution to generate distributed execution paths and cover low- probability corner-case situations. In summary, these tools and algorithm can tackle various similar issues in CPS, but none of them takes into account the nature of physical world with continuous dynamics as is demanded by general purpose CPS.

3. THE ON-LINE SURVEY

To ascertain both the states of the art and practice in CPS verification and validation, we sent an on-line survey to 82 CPS researchers and developers and received 25 responses. We solicited responses from individuals in two categories: those who actively publish work related to CPS development in relevant academic conferences and industry CPS practitioners. We reached experts from a wide range of subfields, including electrical, mechanical, chemical, and biological engineering and from computer science; 37.5% of them have expertise in Control Systems and AI, 37.5% of them in Networking, 16.7% of them in Cyber Security, 16.7% of them in Civil Engineering/Mechanical Engineering/Other Engineering, 37.5% in real-time systems, distributed systems, algorithm, verification, testing, and software engineering, in general. When asked about their primary role(s), 70.8% have roles as CPS modeling expert, designer and architect; 54.2% have roles in validation and verification, and 41.7% classified themselves as CPS developers. The participants had, on average, 8.35 years of software development experience and 6.69 years of experience in CPS applications.

The survey consisted of 32 multiple choice and free form questions designed to (1) understand the participant's definitions of CPS and verification and validation; (2) determine the participant's familiarity with existing techniques for verification and validation, and how they are applied to CPS; (3) to collect information about the techniques that actual CPS developers employ in verifying and validating their CPS systems; and (4) to collect information about the main challenges in verification and validation of CPS systems, from an "in the trenches" perspective. Table 1 shows an abbreviated version of the survey. The perception questions are based on claims made in various forms in the literature with regard to the applicability of verification and validation techniques available today; many of these claims are referenced in the previous background section. Our approach in the survey was *intentional*. We began by generating definitions of both *cyber- physical systems* and of *verification and validation*. We then built on this foundation to determine, in detail, the experts' various approaches to and perceptions of the wide array of CPS verification and validation techniques, running the gamut from "trial and error" to "complete formal modeling and verification."

3.1 General Background

Among CPS developers, there are strong opinions about appropriate programming languages. The programming language greatly influences the verification tools and techniques that can be applied; some techniques apply at the design level and are thus more general purpose, but others apply at the language level. Figure 1 shows the responses to the question "**What programming languages are you familiar with**?" These results mirror surveys of programming language adoption in general. Only one of our respondents was familiar with nesC, the programming language for TinyOS sensor network platforms. This is interesting given that that many CPS experts purportedly believe that high-level programming languages are not appropriate for cyber-physical style systems. A later question broached this question directly, when we asked participants to rate their agreement with, "**A programming language like Java is not applicable to systems with hard real-time constraints**." Figure 2 shows the results; we were surprised by the implication that many of the CPS experts we surveyed found Java to be reasonably appropriate for CPS development. Nearly a third of our respondents were also familiar with functional languages; such languages were originally used in the context of artificial intelligence. This further counters the col-

loquial claim that, in developing CPS, high-level languages are not desirable, which has a potential rippling effect on future research directions in software engineering for CPS. In our follow-up interviews, we found even stronger evidence for these findings (Section 4).

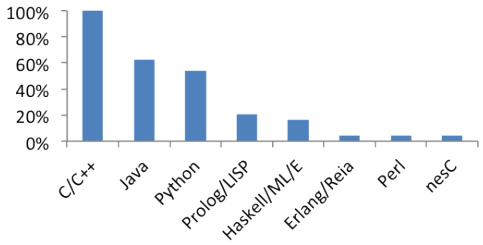

Figure 1: "What programming languages are you familiar with?"

We also asked the participants to express definitions of both cyber-physical systems and verification and validation in their own words. This is important in setting a foundation for the remainder of the survey responses. When we asked, "**In your opinion, what are the main differences between cyber-physical systems and conventional embedded systems**," most respondents' answers identified commonly cited key distinctions; the following responses were typical:

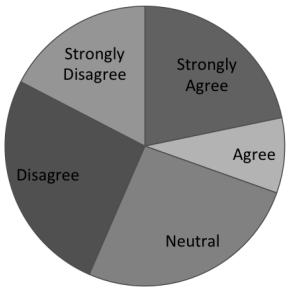

Figure 2: "A programming language like Java is not applicable to systems with hard real-time constraints."

"*embedded systems were mostly focused on software/hardware interacting with low level sensing and real-time control. CPS includes embedded systems but also networks, security, privacy, cloud computing, and even big data.*"

"*CPSs tend to focus more on the interplay between physical and virtual worlds, and the kind of applications possible with the observation (and modification) of the physical world done through devices embedded in the environment.*"

While CPS and embedded systems are often used interchangeably, CPS developers who are "in the trenches" clearly differentiate the two. These perspectives help identify key aspects of CPS that software engineering must address.

While the above gets at participants' individual definitions of CPS (which largely converge), we also wanted to understand CPS developers' perspectives on verification and validation. In response to "**How do you define verification and validation**," over half of the participants gave something quite similar to commonly accepted definitions (i.e., that verification establishes how well a software product matches its specification, while validation establishes how well that software product achieves the actual goal [12]). Many other respondents failed to correctly express the concepts. Intuitively, these results motivate the creation of easy-to-use tools that enable even CPS developers without a rigorous training in verification and validation to develop robust systems; such tools do not exist today.

3.2 Perceptions

One of the primary reasons we performed this survey was because there are several claims made about CPS and the development of CPS that are not widely supported by evidence one way or the other. We phrased these sometimes controversial points as questions about "perceptions" associated with various aspects of CPS verification and validation. We asked the participants to rate their level of agreement (or disagreement) with the statements using a five-point Likert scale.

One common concern is with respect to the wide use of simulation in CPS validation. It is often stated (and even empirically demonstrated [55]) that simulation does not sufficiently match a system's behavior in the real world. On the other hand, in more classical engineering domains, models based on first principles are, in fact, quite representative of the real world [75]. We asked our participants to rate their agreement with "**The use of simulation alone is sufficient for supporting verification and validation of cyber-physical systems.**" Given the variety of backgrounds among our participants, this question has the potential to tease out a potential dichotomy among CPS developers with different backgrounds. In fact, all but one of the survey respondents selected either "Disagree" or "Strongly disagree." The one respondent who selected "Strongly Agree" was also one of the four survey respondents who gave their primary area of expertise as "Civil Engineering/Mechanical Engineering/Other Engineering."

Another commonly held belief is that formal approaches have too high of an overhead to be practically applied in CPS [77]. When we asked the participants to rate "**The use of formal methods for verification and validation of cyber-physical systems is not tractable with respect to resources and time**," the diversity of answers was surprising, as was the apparent support for at least limited use of formal methods for CPS. Figure 3 shows the distribution of responses.

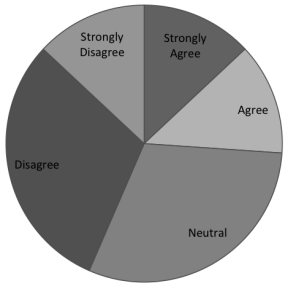

Figure 3: "The use of formal methods for verification and validation of cyber-physical systems is not tractable with respect to resources and time."

We and others have previously claimed that the most common approach to debugging cyber-physical systems requires a significant amount of "trial and error" [57, 64]. To evaluate this claim, we asked the participants' opinions regarding "**The current state of the art of verification and validation of cyber-physical systems involves repeatedly rerunning the system in a 'live' deployment, observing its behavior, and subsequently tweaking the implementation (both hardware and software) to adjust the system's behavior to achieve the stated requirement.**" 91.3% of the participants expressed either "Strongly Agree or Agree." The current "trial and error" processes are neither rigorous nor repeatable, but the extensive amount of *in situ* debugging that these responses demonstrate motivates better support for approaches to verification and validation that function "in the wild."

Our literature review found that approaches to CPS verification and validation tend to focus either on computational models or on models of physics. Rarely do the two con-

verge. The next question in our survey attempted to ascertain whether this is intentional or accidental. We asked the participants to rate their agreement with "**A lack of formal connection to models of physics is a key gap in the verification and validation of cyber-physical systems.**" We found that 69.6% of the respondents selected either "Strongly Agree" or "Agree," while 26.1% were "Neutral." This is corroborated by a second question, in which we asked the respondents whether they agreed with the statement, "**Since CPS has both cyber and physical parts, any approach for verification and validation would need to allow an engineer to in some way examine both parts at the same time**," to which only 27.3% of the respondents selected "Disagree" or "Strongly Disagree." These two results in conjunction indicate a potential need for more expressive and integrated models between cyber worlds and physical worlds.

3.3 Verification and Validation Experiences

The third section of our survey queried the participants about their use of verification and validation techniques, most specifically applied to CPS. As Figure 4 shows, more than 60% of the participants spent between 30-60% of the system's development time on debugging; more than 20% of the respondents spent *more* time than that. Clearly, debugging CPS is expensive and time consuming.

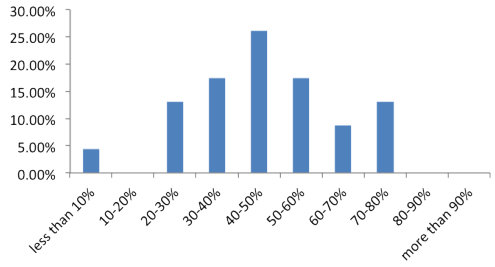

Figure 4: Project time spent in debugging

Only about half of the participants indicated that they employed *code inspection* as a verification and validation technique. Of the respondents who did use code inspection, they chose "detecting faults" as the primary purpose; of the respondents who did not use code inspection, the majority found it to be "not relevant". On balance, while code inspection is not universally used, it is believed by some developers to provide an important and useful tool to improving code quality and code understanding, which is known to lead to less error-prone implementations [74]. While this motivates better tool support for code inspection of CPS, it is not a clear significant concern of active CPS developers.

We received an evenly balanced response to "**Have you used systematic testing to aid in verification and validation?**" Participants who responded affirmatively reported improvement of code coverage, systematic review, and identification of corner cases. Respondents who have not used systematic testing gave pretty standard reasons:

"*lack of time and deep familiarity*"

"*no easily available tools.*"

CPS developers are not universally familiar with traditional systematic testing and tools. We believe that, though systematic testing is well established, there are research challenges in bridging the gap between existing techniques and CPS development (e.g., symbolic execution for CPS).

When we asked "**Have you used formal methods (e.g., model checking) to aid in verification and validation?**" the majority, in fact, replied affirmatively. When we followed up with the participants who had employed formal methods about the advantages, they reported the benefits of inferring useful pattern, complete testing, finding corner cases, and verifying key components. Some participants even reported a sense that model checking was becoming increasingly practical for real systems. Those who did not use model checking said that it is (for example):

"*overly complicated for most purposes; most bugs arise from time dependent interactions with physical systems.*"

"*not applicable to my domain; [...] too demanding and unreliable.*"

When we asked "**What specific model checker(s) do you employ during your verification and validation activities?)**" participants reported relatively high usage of Spin [35] (53.85%), NuSMV [15] (46.15%), and UPPAAL [41] (46.15%)), as shown in Figure 5. There was also substantial high use of other (mostly domain-specific) model checkers.

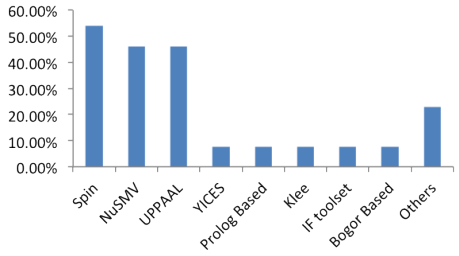

Figure 5: Model checkers used

From the free form responses, we noticed that participants gravitate towards general purpose model checkers for very small, very specific pieces of their systems. These model checkers do not enable combined reasoning about the cyber and physical portions of the systems, which is critical to complete and correct verification of CPS.

The responses to "**Have you used simulation to aid in verification and validation?**" were overwhelmingly positive; only one participant said "no." Participants reported the use of simulation for understanding the system, prototyping behavior, refining specifications, exploring configurations, and minimizing test effort as uses for simulation: also put some of quite useful quotes from these participants below.

"*can provide some preliminary confidence of the system*"

"*can help refine the specification and validate the system*"

"*allows assumptions made in modeling to be cross-validated against another source of ground truth*"

"*helps save and focus testing effort.*"

One participant astutely noted, "*modeling and simulation only goes so far. No one ever found oil by drilling through a map on a table.*" The one participant who did not rely on simulation stated that, "*Good enough simulation does not exist.*" The participants reported high usage of Simulink (61.1%), but also proprietary, in-house tools (77.8%), among many others, as shown in Figure 6.

Our survey results indicate that simulation is commonly used for verifying and validating CPS. However, the remark-

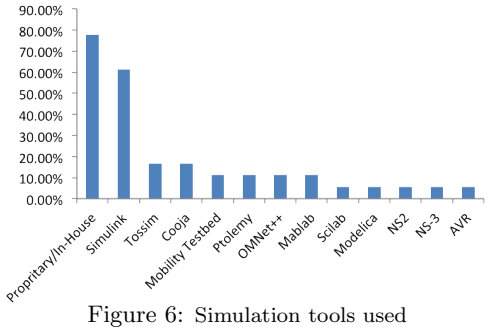

Figure 6: Simulation tools used

ably high ratio of in-house simulation is concerning because it naturally limits the reproducibility and generalizability of the results and implies that developers find that simulation tools in general are not sufficient.

A common and relatively straightforward approach to debugging at the source level is to augment the program with assertions that provide checkpoints on the program state throughout its execution. When we asked the participants about their use of assertions in CPS, the vast majority (more than 70%) had used them. The respondents used assertions primarily for bug detection and writing formal specifications, and, to a slightly lesser extent, for documenting assumptions. The participants stated that the use of assertions:

"*[provides] formal documentation [and] explicitly states otherwise implicit assumptions, often enabling to detect errors sooner and have a better indication of where a problem stems from*"

"*forces developer to write down (basically as part of the code) the expectation for correct behavior [with] the bonus of being able to 'execute' the assertion.*"

The two main reasons cited for not having used assertions in CPS development were concerns about performance and the difficulty in tracing assertions in deployed distributed systems. In general, these results bolster our hypothesis that assertions are a useful means to verify and validate CPS; future research that tailors assertions to particular challenges of CPS (e.g., distribution and physical aspects) may ameliorate some of the concerns.

Finally, we asked our respondents about their perception of remaining open challenges in verification and validation of CPS. Almost 50% of the participants reported issues with physics models, more than a quarter cited scalability issues, and nearly a quarter reported issues with a lack of systematic verification and validation methods. Figure 7 reports the complete results. Example statements include:

"*CPS models are fragile. We need verification and validation methods that scale with the complexity of the model and are robust to slight variations of the model.*"

"*Time plays a critical role and is misunderstood.*"

"*Impossible to fully understand environment dynamics.*"

The survey results indicate models of software systems, models of physics, and the integration of the two are major bottlenecks in verification and validation of CPS. Scalability issues of existing tools and techniques and a lack of a capability of directly verifying CPS code from simulation motivate new, tailored approaches that build on and complement the current state of practice. Before exploring what this might look like, we took a few of our survey participants and interviewed them in detail.

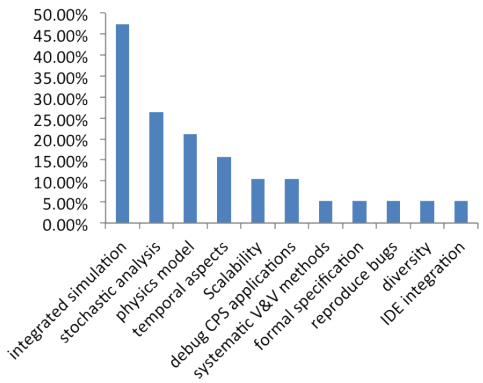

Figure 7: Main research challenges

4. INTERVIEWS

To more deeply examine the implications of several of the responses in the survey, we conducted follow-up interviews with five of the experts that completed the initial survey. We interviewed an expert in *autonomous vehicles*, another in closely related *autonomous robots*, one in *medical CPS*, one in *formal specification and algorithms*, and one in *unmanned aerial vehicles and wireless sensor networks*. We built the questions around trends we saw in the survey results, and the goal of the interviews was to try to confirm or dispel common conceptions about CPS development and debugging.

We were somewhat surprised to find that our survey respondents were not completely against using high-level programming languages like Java to build CPS. The interviews corroborated the survey results and, in fact, highlighted high-level languages that are popular among the CPS developers we interviewed. Specifically, when we asked, "**What are the main programming languages you used in developing CPS applications?**" our interviewees said, as typical examples:

"*For the high level, mostly it is Python. Low level is C and C++. In the middle is Java.*"

"*For wireless sensor networks, mainly C, nesC; for aerial drones, mainly C++, Java. We also extended C++, C, and Java with high level abstractions for specific needs.*"

Quite simply, while "low-level" languages (i.e., those that are closer to the hardware) are still popular among CPS developers, the perception that high-level languages are not used for CPS development does not universally hold.

We entered our studies with the perception that CPS developers are hampered by severe resource constraints of their deployment environments. Our analysis of the survey results hinted that this might not be the case. In our one-on-one interviews, we engaged the interviewees in conversations about the actuality of the resource constraints of their target platforms *and* their perceived impact of those constraints on the debugging task. The interview results indicate that CPS developers do not always perceive their target platforms to be resource constrained. Further, the CPS developers we interviewed did not perceive any resource constraints the platforms may have to be a significant impediment to development and debugging tasks. Some specific quotes include:

"*The computation platform is not resource constrained. We are not limited by this.*"

"*[The platforms] are much more advanced than a mote. We are not too worried about performance and resources.*"

"*We don't have concerns of resources in general. However, to me, wireless sensor networks are a specific type of CPS, the device nodes are for sure resource constrained. But for other types of CPS applications, it might not be the case.*"

This is an important finding in the sense that techniques for supporting development tasks for CPS (including those for verification and validation) often have a quite significant focus on resource constraint aspects; these efforts may, in fact, be misplaced or at least over-emphasized.

We know from our own experience and from reviewing the literature that simulation is commonly used in verification and validation in CPS. However, many researchers and practitioners discount the value of simulation results. During our interviews, we asked the interviewees about the simulation tools they use and their perception of the pros and cons of using simulation. The following are some samples of the resulting discussions:

"*These simulations are not very truthful. We used an in-house hybrid simulation tool, [but] even with this in-house simulation, we need real testing as the risk is too big for any undetected errors in autonomous vehicles.*"

"*I am not happy with [...] simulation tools as they are not accurate enough; they give you a basic sense of how the system would work, but actually making the simulation work requires [too much effort] to tune parameters.*"

"*We used simulation but not the major part of it. What you can test through simulation is only a very small fraction of the problems which can potentially come out when you deploy the system.*"

A common theme was the revelation that the primary simulation tools used were in-house simulators. Further, though simulation is commonly used in CPS verification and validation, it is far from being complete and perfect. From our interviews, we learned that simulation is increasingly likely to be used primarily only in the earlier stages of design to give a rough view of the system and its behavior.

Concerns about the applicability of model checking to the CPS appeared to crop up in our survey; our interviews delved deeper into the use of model checkers by our interview subjects. Specifically, we asked, "**How do you use model checkers in verifying and validating CPS applications?**" The responses we received, some of which follow, indicate that model checkers enjoy only a limited use by in-the-field CPS developers, usually employed to check only small pieces of the larger system in isolation:

"*We use a very simplistic model for partial ordered sets and use Spin to check it.*"

"*We would like to transform our questions into timed automata and feed the input into UPPAAL. But the model checking suffers from space explosion, and we have to restrict our input to very small set. It is not that useful [...] model checking [does not] fit our needs.*"

Further details of these discussions around model checking led to a desire by the interview subjects supplant model checking with more robust run-time verification that is both "on-line" and "incremental."

We asked our interviewees, generally, "**How do you test CPS applications?**" Across the board, the responses validated our view that trial and error is currently the most prevalent approach. These responses include:

"*We use simulation and trial and error to observe errors.*"

"*Mainly visual observation, look at what robots are doing, take videos and take sensor data reading. Basically it is trial and error.*"

"*Test software isolated from sensors and controller, then use trial and error to visually observe what is going on.*"

"*Visual observation. We collect traces and print out sensor values. We manually read the traces. It is trial and error.*"

"*Trial and error. We did apply some formal and unit testing at some point; it is pretty rare to find bugs by testing one unit at a time.*"

The majority of our survey respondents identified a lack of formal connection to models of physics as a key concern. We explored this gap more deeply in the interviews by asking the subjects about the software and physics models they employ for CPS development and what they viewed to be the pros and cons of using these models. We were surprised to find that CPS developers tend not to deeply consider (formal) models of physical systems during development. They also found the available software models to be inadequate. Some examples of their responses include:

"*The environment is not ideal, we created static physics models to handle noise. The models are still immature and fixed. We need on-line learning models.*"

"*We used physics models of motors [and a] flow dynamics model. We use these models to determine what forces to counteract using actuation.*"

"*We mainly used distribution models (e.g., partial order models, lattice models) to detect global [correctness] predicates. We abstract away the physics model.*"

Recent emerging work has demonstrated the use of model-driven development to automatically generate CPS software from heavily validated models [38, 59]. Our interviews attempted to ascertain a practitioner's view on the use of model-driven development for CPS. Model-driven development, though having a quite lengthy history, is far from mature, especially with respect to CPS. Some examples of our interviewees' responses include:

"*For simple problems, model-driven development might be possible. But for complex problems, [...] model-driven development is not very realistic.*"

"*[Model-driven development] can abstract away complexity. However, I am conservative about how the models can match the real environment [...] and how much effort it takes. The core challenge is how to automate the transition from software using the models and software working [in the real world].*"

"*I am not optimistic about this approach. To create models that are very accurate takes too long, which is not useful.*"

"*There is a significant gap between the perceived environment and the modeled environment. [You risk] building a*

complex model is more complex than the traditional programming task."

"*I don't believe in that.*"

To explore our subjects' opinions on future research directions for CPS development and debugging, we ask open-ended questions, "**What are your ideal testing tools for CPS that are not currently available?**" As the previously discussed results, both for the survey and for the interview were leading, the interview subjects described a need for highly faithful physics models, simple but expressive software modeling languages, and the integration of the two. The subjects also expressed a need for better simulators and debugging tools that give programmers greater visibility into the entire system's behavior (both cyber and physical) and better fault localization. The following offer some quotes from these responses:

"*high-fidelity simulation with on-line learning of models.*"

"*accurate run-time models of physics and software models to use for off-line development.*"

"*tools that can reproduce bugs. If it is available, we could throw some random errors into the model to check how the system reacts to these. It is also ideal to have multi-domain models for motors, mechanical systems.*"

"*[techniques for] formally specifying the system behaviors, automating the fault localization by specifying a syndrome (e.g., a pattern).*"

Finally, we also asked our interview subjects, "**How have you used assertions? What improvement you like to see for the use of assertions in CPS?**" Our interview results seem to confirm our intuition that assertions are a reasonable approach to debugging CPS, but that to make them even more appealing, especially to domain experts, the assertion framework should be complemented by CPS-specific features (e.g., temporal and physics aspects).

"*I used assertions to assert the effects of actuation, mainly used for debugging. We actually need to debug assertions, as assertion happens too quickly and it fails to observe the effects of actuation. In CPS, actuation latency is not taken care of by traditional assertions.*"

"*I used assertions to figure out errors. Since I could not step through code since the interaction with the physics, I find assertions is very useful in this regard.*"

5. FUTURE RESEARCH DIRECTIONS

Our goals were to discover CPS developers' perspectives on verification and validation, their application of existing and traditional techniques to verification and validation in building CPS, the tools and techniques that CPS developers find most and least useful, and the remaining open challenges in verification and validation of CPS. Along the way, we identified several preconceptions CPS experts have about developing and debugging CPS. Table 2 shows a summary of several of these common assumptions and misconceptions, including references to where they appear in the literature, and cross-references to our survey and interviews where we discuss how our investigation either confirmed or dispelled the notion.

From our analysis of both the survey and follow-up interviews, it is clear that, while CPS developers are aware of and sensitive to the importance of verification and validation in CPS, there remain significant concerns in both cyber and physical models, the scalability of model checking approaches, the applicability of simulators, and the inability of other approaches to incorporate the fundamental concerns of CPS (not the least of which is the need to for tools and techniques to be accessible to domain-experts who may not be programming experts). We are strongly motivated to move CPS debugging into world where the techniques are more rigorous and repeatable than the "trial and error" approaches that are the state of the art, while being sensitive to the way that these domain experts are inevitably going to develop their cyber-physical systems.

Trial and error. Because trial and error has become the primary and pervasive way developers verify CPS, it is unreasonable to expect a dramatic shift in the workflow that would require CPS developers to learn and integrate fundamentally different tools and techniques. For this reason, it seems reasonable to explore techniques to verification and validation that fit naturally within a trial and error model but address the concerns about such approaches (e.g., the inability of trial and error to cover all potential use cases or potentially encountered environmental conditions). Exploring the integration of assertions into the natural programming language environment [84] offers promise in this vein, since it does not force the developer to integrate a new tool suite. Given the responses in our survey and interviews, such techniques must be coupled with fault localization capabilities; it is simply not enough to inform the developer that a fault occurred, but it is extremely important in CPS to help identify the location or cause of the fault so that time is not wasted attempting to fix the incorrect thing. This is particularly important and difficult in CPS because the faults may come from either logical errors in the software, from incorrect or incomplete models of the physics of the system, or from encountering an environment that was unexpected. Tools for CPS verification and validation must therefore support testing and debugging *in situ*.

Models and model integration. As we covered in the literature review, there are many models of hybrid systems that may apply to CPS as well, but each of them provides only incomplete coverage of the challenges that emerge in CPS applications. CPS appears to require a hierarchy of models where an abstract model provides an overview of the entire system, including the cyber and physical components. Under this abstract model, there must also be traditional models of the software logic, models of the physical components (e.g., motors, drive trains, actuators, etc.), and models of properties of physics (e.g., models of friction for autonomous vehicles, models of ambient environments for smart homes, models of blood flow, etc.). The abstract model must support both continuous dynamics and discrete states and provide some form for specifying interaction that crosses from cyber to physical and back. Instead of representing raw sensor data at this abstract level, the model should reason about abstract state and remain agnostic to the specifics of how and in what format this information is captured and represented. In software engineering, models are traditionally relegated to a supporting role, for example to drive a simulation or to enable model checking, but the tight integration of the cyber and physical in CPS may benefit from applying models across all development stages. For example, models can be used in development and debugging

Table 2: Summary of common assumptions/misconceptions about CPS development and debugging

Claim	Confirmed	Dismissed	Sections
A programming language like Java is not applicable in CPS.		**X**	§3.1, §4
Resource constraints (mainly in terms of CPU, memory and storage) are a major issue in developing and debugging CPS.		**X**	§4
Simulation alone is not sufficient to support verification and validation of CPS. [55]	**X**		§3.2, §3.3, §4
Existing model checking practices are insufficient to meet CPS application's needs. [13, 16, 43]	**X**		§3.2, §3.3, §4
The state of the art in debugging CPS involves trial and error. [57, 64]	**X**		§3.2, §4
Existing models are not sufficient to meet demands in CPS. There is a significant gap between models of computing and communications, and models of physics. [2, 44, 71]	**X**		§3.2, §3.3, §4
In CPS, model driven development (e.g., to generate code automatically from model) is still in its infancy stage. [24]	**X**		§4

to aid in constructing expressive appropriate assertions. We can also train physics models in a debugging environment and use these trained models in a deployment environment to generate accurate sensing and feedback loops at runtime. From our review and study, such efforts appear to be unexplored but of potential impact to real CPS developers.

Simulation and model checking. Simulation and model checking are still "go-to" techniques for CPS developers, at least at a small scale or for prototyping. Our study highlighted that both approaches remain frustrating from a practical perspective. Simulation tools are limited largely by the inaccuracy of models. For example, simulation results cannot be trusted to provide a true reflection of the real world, both with respect to representing the results of sensing and the effects of actuation. From our study, we conjecture that CPS developers would benefit from a flexible framework for moving between full simulation and a full testing environment, allowing aspects of the simulation to be incrementally replaced by physical devices and other characteristics of the real deployment environment. This framework requires an environment in which models and physical devices can "talk" the same language, making the transition from one to the other transparent to the CPS developer and his debugging task. Model checking suffers less from a lack of accuracy of the models and more from a scalability of applying the highly accurate models. To address these concerns and other characteristic aspects of CPS, model checking must allow for more complexity and include stochastic expressions for randomness and differential equations for physics. These must be incorporated with an eye to improving scalability, especially of the integrated system instead of just its kernel components as is currently available.

As such, future research in model checking shall focus on improving scalability and allows integration of stochastic expressions for randomness and differential equations for physics.

6. THREATS TO VALIDITY

Internal validity. We made some assumptions in some of the findings in Sections 3 and 4. For instance, from the reported remarkably high ratio of in-house simulation, we draw a conclusion that general purpose simulation tools are not sufficient. There might be other confounding variables that result in the high ratio of in-house simulation, for instance participants might have no access to the general purpose simulation tools due to license issues. The on-line survey's lack of interaction restricted us from ruling out those confounding variables. To mitigate these types of issues, we used literature survey and follow-up interview to corroborate our findings.

Construct validity. The questions in our on-line survey and follow-up interviews may neglect some additional important questions, which may consequently cause us to overlook key issues in verification and validation of CPS. To mitigate this issue, we carried out an extensive and comprehensive literature survey, which altogether covers around 160 research papers across highly relevant domains and publication venues. This coverage mitigates the concern that we missed a significant question for our survey or interview. Another construct validity issue lies in the number of participants in the on-line survey (25) and follow-up interviews (5). We did successfully reach a wide cross-section of disciplines and cultures, including both researchers and practitioners.

External validity. The participants in the interview are (necessarily) from a limited set of domains. These interviews do not include CPS developers from many interesting CPS fields like structural health monitoring, smart energy grid, and smart city. The conclusions drawn from the interviews may not be applicable to these domains. To mitigate this issue, our on-line survey, on the other hand did cover participants from a wide range of subfields.

7. CONCLUSIONS

We presented a three-part approach to obtaining a complete picture of the state of the art and the state of the practice of verification and validation in cyber-physical systems though a comprehensive literature survey, followed by an on-line survey of CPS developers, and finished with detailed follow-up interviews of a subset of the survey respondents. Our studies find that there are significant research gaps in addressing verification and validation of CPS; these gaps potentially stand in the way of the construction of robust, reliable, and resilient mission-critical CPS. Our study is the first of its kind to corroborate a colloquial belief: that *CPS debugging is largely a trial-and-error* process. Further, our study provides some evidence of the reason, namely a lack of trust that developers have in the veracity of simulation results and a frustration with the practicality of more robust model-based methods. Based on these studies, we highlighted some potential research directions that directly address challenges that real CPS developers cited in the experiences in developing and debugging real-world CPS.

8. ACKNOWLEDGMENTS

Thanks to all participants in the on-line survey and follow-up interviews. This work was supported in part by the NSF under grant CNS-1239498. Any findings, conclusions, or recommendations are those of the authors and do not necessarily reflect the views of the sponsoring parties.

9. REFERENCES

[1] E. Abrahám-Mumm, U. Hannemann, and M. Steffen. Verification of hybrid systems: Formalization and proof rules in PVS. In *Proc. of ICECCS*, 2001.

[2] R. Akella and B. M. McMillin. Model-checking BNDC properties in cyber-physical systems. In *Proc. of COMPSAC*, 2009.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicolin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theo. Comp. Sci.*, 138(1):3–34, 1995.

[4] R. Alur and D. Dill. The theory of timed automata. In *Real-Time: Theory in Practice*, pages 45–73, 1992.

[5] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in CHARON. In *Proc. of HSCC*. 2000.

[6] R. Alur, A. Kanade, S. Ramesh, and K. Shashidhar. Symbolic analysis for improving simulation coverage of Simulink/Stateflow models. In *Proc. of Emsoft*, 2008.

[7] R. Baheti and H. Gill. Cyber-physical systems. *The Impact of Control Technology*, pages 161–166, 2011.

[8] T. Ball, V. Levin, and S. K. Rajamani. A decade of software model checking with SLAM. *Communications of the ACM*, 54(7):68–76, 2011.

[9] A. Banerjee and S. Gupta. Spatio-temporal hybrid automata for safe cyber-physical systems: A medical case study. In *Proc. of ICCPS*, 2013.

[10] N. Bjoner, Z. Manna, H. Sipma, and T. Uribe. Deductive verification of real-time systems using STeP. *Theo. Comp. Sci.*, 253(1):27–60, 2001.

[11] C. D. Bodemann and F. De Rose. The successful development process with matlab simulink in the framework of ESA's ATV project. In *Proc. of IAC*, 2004.

[12] B. Boehm. Verifying and validating software requirements and design specifications. In *IEEE Software*, 1984.

[13] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li. Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior. *ACM SIGBED Review*, 8(2):7–10, 2011.

[14] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: a framework for simulating and prototyping heterogeneous systems. In *Readings in hardware/software co-design*, pages 527–543, 2001.

[15] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model verifier. In *Proc. of CAV*, 1999.

[16] E. M. Clarke, B. Krogh, A. Platzer, and R. Rajkumar. Analysis and verication challenges for cyber-physical transportation systems. In *Nat'l. Wkshp. for Research on High-Confidence Transportation CPS*, 2008.

[17] E. M. Clarke and P. Zuliani. Statistical model checking for cyber-physical systems. In *Proc. of ATVA*. 2011.

[18] R. Colgren. Efficient model reduction for the control of large-scale systems. In *Efficient Modeling and Control of Large-Scale Systems*, pages 59–72. 2010.

[19] R. David and H. Alla. On hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1-2):9–40, 2001.

[20] J. M. Davoren. *On hybrid systems and the modal μ-calculus*. 1999.

[21] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III*. 1996.

[22] I. Demongodin and N. T. Koussoulas. Differential petri nets: Representing continuous systems in a discrete-event world. *IEEE Trans. on Automatic Control*, 43(4):573–579, 1998.

[23] P. Derler, E. A. Lee, and A. L. Sangiovanni-Vincentelli. Addressing modeling challenges in cyber-physical systems. Technical Report UCB/EECS-2011-17, UC Berkeley EECS, 2011.

[24] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *Proc. of the IEEE*, 100(1):13–28, 2012.

[25] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, 2010.

[26] A. Elci and B. Rahnama. Considerations on a new software architecture for distributed environments using autonomous semantic agents. In *Proc. of COMPSAC*, 2005.

[27] H. Elmqvist, S. E. Mattsson, and M. Otter. Modelica-a language for physical system modeling, visualization and interaction. In *Proc. of CACSD*, 1999.

[28] J. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef. A formal approach to collaborative modelling and co-simulation for embedded systems. *Mathematical Structures in Computer Science*, 23(04):726–750, 2013.

[29] Z. Han and B. Krogh. Reachability analysis of hybrid control systems using reduced-order models. In *Proc. of American Control Conference*, 2004.

[30] D. Harel, J. Tiuryn, and D. Kozen. *Dynamic logic*. 2000.

[31] T. Henderson, M. Lacage, and G. Riley. Network simulations with the ns-3 simulator. In *Proc. of SIGCOMM*, August 2008.

[32] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. In *CAV*, 1997.

[33] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proc. of STOC*, 1995.

[34] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS*. 2006.

[35] G. J. Holzmann. Basic spin manual, 1980.

[36] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 2001.

[37] J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In *Proc. of HSCC*. 2000.

[38] Z. Jiang, M. Pajic, and R. Mangharam. Cyber–physical modeling of implantable cardiac medical devices. *Proc. of IEEE*, 100(1):122–137, 2012.

[39] H. Jifeng. From CSP to hybrid systems. In *A classical mind*, pages 171–189, 1994.

[40] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han. Dustminer: troubleshooting interactive complexity bugs in sensor networks. In *Proc. of SenSys*, 2008.

[41] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int'l J. on STTT*, 1(1):134–152, 1997.

[42] J. Le Bail, H. Alla, and R. David. Hybrid petri nets. In *European Control Conference*, 1991.

[43] E. A. Lee. Cyber-physical systems-are computing foundations adequate. In *NSF Wkshp. On CPS*, 2006.

[44] E. A. Lee. Cyber physical systems: Design challenges. In *Proc. of ISORC*, 2008.

[45] E. A. Lee and H. Zheng. Hyvisual: A hybrid system modeling framework based on Ptolemy II. In *Proc. of ADHS*, 2006.

[46] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[47] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. of SenSys*, 2003.

[48] J. Lin, S. Sedigh, and A. Miller. Towards integrated simulation of cyber-physical systems: a case study on intelligent water distribution. In *Proc. of DASC*, 2009.

[49] J. Lin, S. Sedigh, and A. Miller. Modeling cyber-physical systems with semantic agents. In *Proc. of COMPSACW*, 2010.

[50] G. Lipovszki and P. Aradi. Simulating complex systems and processes in LabVIEW. *J. of Mathematical Sciences*, 132(5):629–636, 2006.

[51] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. Technical report, U Virginia, CS, 2006.

[52] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. S. Sastry. Dynamical properties of hybrid automata. *IEEE Trans. on Automatic Control*, 48(1):2–17, 2003.

[53] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Info. & Computation*, 185(1):105–157, 2003.

[54] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. Technical Report TM-373, Laboratory for Computer Science, MIT, 1988.

[55] S. Mitra, T. Wongpiromsarn, and R. M. Murray. Verifying cyber-physical interactions in safety-critical systems. *IEEE Security & Privacy*, 11(4):28–37, 2013.

[56] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, 1989.

[57] A. Murugesan, S. Rayadurgam, and M. Heimdahl. Using models to address challenges in specifying requirements for medical cyber-physical systems. In *Proc. of ICCPS Workshops*, 2013.

[58] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. of CADE*. 1992.

[59] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. Safety-critical medical device development using the UPP2SF model. *ACM Trans. on Embedded Computing Systems*, 2014. (to appear).

[60] D. L. Parnas. Really rethinking formal methods. *Computer*, 43(1):28–34, 2010.

[61] T. S. Perry. In search of the future of air traffic control. *IEEE Spectrum*, 34(8):18–35, 1997.

[62] A. Platzer. Differential dynamic logic for hybrid systems. *J. of Auto. Reasoning*, 41(2):143–189, 2008.

[63] A. Platzer. Quantified differential dynamic logic for distributed hybrid systems. In *Proc. of CSL*, 2010.

[64] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proc. of DAC*, pages 731–736, 2010.

[65] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proc. of SenSys*, 2005.

[66] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, MIT, 1974.

[67] K. Romer and J. Ma. Pda: Passive distributed assertions for sensor networks. In *Proc. of IPSN*, 2009.

[68] R. Sasnauskas, O. Landsiedel, M. H. Alizai, C. Weise, S. Kowalewski, and K. Wehrle. KleeNet: discovering insidious interaction bugs in wireless sensor networks before deployment. In *Proc. of IPSN*, 2010.

[69] C.-J. Seger. *An introduction to formal hardware verification*. 1992.

[70] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.

[71] J. Shi, J. Wan, H. Yan, and H. Suo. A survey of cyber-physical systems. In *Proc. of WCSP*, 2011.

[72] S. Sierla, B. M. O'Halloran, T. Karhela, N. Papakonstantinou, and I. Y. Tumer. Common cause failure analysis of cyber-physical systems situated in constructed environments. *Research in Engineering Design*, 2013.

[73] J. Sifakis. Use of petri nets for performance evaluation. *Acta Cybern.*, 4:185–202, 1980.

[74] H. Siy and L. Votta. Does the modern code inspection have value? In *Proc. of ICSM*, 2001.

[75] J. M. Solis and R. G. Longoria. Modeling track–terrain interaction for transient robotic vehicle maneuvers. *J. of Terramechanics*, 45(3):65–78, 2008.

[76] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrodebugging: Global views of distributed program execution. In *SenSys*, 2009.

[77] R. A. Thacker, K. R. Jones, C. J. Myers, and H. Zheng. Automatic abstraction for verification of cyber-physical systems. In *Proc. of ICCPS*, 2010.

[78] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In *Proc. of HSCC*. 2002.

[79] S. Tripakis and C. Courcoubetis. Extending promela and spin for real time. In *Proc. of TACAS*. 1996.

[80] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proc. of Simutools*, 2008.

[81] K. Wan, K. Man, and D. Hughes. Specification, analyzing challenges and approaches for cyber-physical systems (cps). *Engineering Letters*, 18(3), 2010.

[82] Y. Yalei and Z. Xingshe. Cyber-physical systems modeling based on extended hybrid automata. In *Proc. of ICCIS*, 2013.

[83] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In *Proc. of SenSys*, 2007.

[84] X. Zheng. Physically informed assertions for cyber physical systems development and debugging. In *Proc. of PerCom*, 2014. (to appear).