Verification and Validation in Cyber Physical Systems: Research Challenges and a Way Forward

Xi Zheng and Christine Julien*
*The University of Texas at Austin
Email: jameszhengxi,c.julien@utexas.edu

Abstract—**It is widely held that debugging cyber-physical systems (CPS) is challenging; to date, empirical studies investigating research challenges in CPS verification and validation have not been done. As a result, the exact challenges facing CPS developers in the real world remain at best unquantified and at worst unknown, and the research directions the community should undertake are not clearly identified. In this paper, we review our recent empirical study of real-world CPS developers. This position paper then uses the findings from this study to highlight the discovered key challenges and to present a research trajectory to address these challenges.**

I. Introduction

Cyber-Physical Systems (CPS) are an integration of computation and physical processes. CPS have gained popularity both in industry and the research community and are represented by many varied mission critical applications [17]. Debugging CPS is important, but the intertwining of the cyber and physical worlds makes it very difficult. There are several common conceptions and misconceptions related to CPS verification and validation, but very little existing research attempts to systematically clarify these mysteries. As a result, research on CPS verification and validation is largely working in the dark, tackling issues that may or may not be of importance to the CPS development community.

For instance, simulation-based testing is often used in mission-critical CPS applications, and many CPS research projects build support for such simulation-based testing. But is simulation-based testing appropriate? What features are inefficient or incomplete? 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 limited the steering rate to low speeds [13]. When the path planner produced a sharp turn at higher speeds, the vehicle physically could not follow. The analysis also concluded that, although simulation-centric tools are indispensable for rapid prototyping, design, and debugging, they are limited in providing correctness guarantees. However, there is no study to systematically analyze why simulation-based testing fails and show how CPS practitioners think about using simulation-based testings for CPS verification and validation.

Through an extensive literature review, we found many such examples. For instance, we found that, in some mission-critical industries (e.g., medical device development), correctness is currently satisfied by documentation associated with code inspections, static analysis, module-level testing, and integration testing. These tests do not consider the context of the patient [7]. We could not find any literature to show how a real CPS practitioner could convince himself that something like the Therac-25 disaster [12] will not reoccur.

Given the current state of research on software engineering for CPS, many additional questions derived naturally. For instance, who are these real CPS practitioners and what are their backgrounds? Are they trained in traditional software engineering methodologies and tools? Given research and development trends associated with programming languages that target the resource constrained computing platforms that characterize CPS (e.g., nesC [6]), what programming languages are actually used by CPS developers and how do these languages and their programmers deal with challenges caused by resource constraints? More to the point of understanding the end-to-end software engineering process for CPS, it is largely believed that a trial-and-error approach is the state of art for developing and verifying CPS systems [17]. Even faced with this reality, many researchers in more traditional software engineering verification and validation domains regard this claim as difficult to fathom due to the well demonstrably poor test coverage of such an approach. But how do real CPS developers view this approach to verifying their systems?

To seek the answers to these questions, we conducted a qualitative and quantitative empirical study of the state of the art in CPS verification and validation [20]. Based on the overview of the study results and research gaps it identifies, this position paper proposes an incremental and realistic approach to address the research challenges in CPS development and verification. We first review our empirical study and follow with a research road-map towards supporting feasible and usable CPS runtime verification to address a subset of the research challenges identified in the study.

II. The Empirical Study

To address the dearth of empirical information available about CPS development, specifically in debugging and testing, we conducted an empirical study [20] with three parts: a broad literature review, a quantitative on-line survey, and qualitative interviews. In the literature review, we took a very broad look at techniques that have been or could be applied to CPS verification and validation. Based on the findings, we created a set of multiple choice questions that attempted to resolve the veracity of a set of commonly held beliefs surrounding CPS development and debugging. We received survey responses from 25 researchers and developers who are active in the area of cyber-physical systems. In our more in-depth interviews, we used a set of open-ended questions around the trends we saw in the survey results to further explore the opinions of CPS practitioners related to CPS verification and validation. The nine interviewees were CPS experts from around the world.

Our literature review covered exemplars in a wide variety of applicable areas, including formal methods, model- and simulation-based testing, runtime verification, practical tools, and social and cultural factors that have a non-trivial impact on the adoption of techniques for verification and validation of CPS. One of the more interesting and relevant findings was a piece of work that used Isabelle/HOL [16] to formally verify the kernel of a secure embedded real-time operating system that is the foundation for a highly secure military CPS application [9]. This demonstrated that, with careful design, a critical component of a complex CPS can actually be formally verified by a state-of-the-art theorem prover. However, we also found that requirements and costs of human involvement (i.e., the total effort for the proof was about 20 person-years [8]) makes this specific approach inadequate for verifying an entire general-purpose CPS application. Also particularly promising in our literature review was the state-of-the-art in runtime verification, especially in the form of aspect-oriented monitoring [4], which can potentially provide a great supplement for formal methods and traditional testing. However, as we discuss in more detail in Section III, runtime verification applied to CPS has its own distinct research challenges.

The most illuminating aspects of our empirical study [20] came from the surveys of and interviews with real CPS developers and researchers. We next briefly highlight some of the widely held beliefs we targeted and summarize the findings of our studies. For each belief we review, we briefly state the belief in bold text and then discuss our findings.

CPS developers are generally unfamiliar with traditional software verification and validation tools and methodologies. We found this belief to be largely, and disappointingly, true. Many of the CPS developers and researchers responding to our survey came from other engineering fields (e.g., civil and mechanical engineering), and it is therefore not unreasonable that they have not been introduced to mainstream software engineering tools. These results also motivate the fact that new tools and specification languages targeted to CPS verification and validation must have intuitiveness as a primary goal.

High-level programming languages (e.g., Java) are not applicable to CPS. Contrary to this popularly held belief, we found that high level programming language, along with functional languages, are actually quite commonly used in developing CPS applications. **Resource constraints (e.g., CPU, memory, and storage) are a major issue in developing and debugging CPS.** This claim often goes in hand with the previous one; outsiders assume CPS developers eschew high-level programming languages because they tie developers' hands in dealing with low-level concerns like resource constraints. We found that, although resource constraints are indeed often considered at the lower layers of CPS system (e.g., sensor drivers and wireless networks), the upper layers (i.e., the core application logic) are not bounded by resources. One particular interviewee, who is in charge of a large scale bridge monitoring system, stated that the team designed the resource layer (e.g., sensors and networking unit) to fit the computation requirement for the software. These findings might have a rippling effect on research directions for CPS since much current research focuses on building tools and techniques specific to resource-constrained platforms and languages and are therefore potentially limited in practical applicability.

Existing model checking and other formal techniques are insufficient to meet CPS applications' needs. This belief rang true with our study participants. Our respondents indicated that the two primary reasons that these existing techniques are insufficient are their steep learning curves and the resulting inefficiencies in checking a system of any real size. **There is a significant gap in language between formal models of computing and communications and models of physics that makes applying them jointly in CPS challenging.** In our studies we found that, while cyber physical systems inherently intertwine the computational and the physical, the tools and techniques available to developers have largely focused only on the computational aspects, leaving physics and physical models underrepresented in the verification and validation stages. CPS researchers and practitioners therefore discount the value of simulation tools not only because of the inefficiencies associated with applying them but also because of the inaccuracy of the (physical) models.

This brings us to the final contested belief. **An ad hoc, trial-and-error approach to development is the state of the art for CPS systems.** We found this belief to be overwhelmingly true among CPS developers "in the trenches." CPS practitioners are not enamored with this approach, and they are fully aware of the shortcomings. Nonetheless, the demand for CPS applications and the insufficiencies of alternative approaches lead trial-and-error to be the "state-of-the-art." The lack of rigor in employed testing approaches in general should be of grave concern to the software engineering community and leads us to propose the research roadmap described in the next section, based on three specific challenges:

- The trial-and-error testing currently employed in CPS do not provide sufficient rigor in error detection.
- Formal methods provide a desired level of expressiveness and would improve testing coverage, but existing formal methods are not intuitive and efficient (scalable) enough to be adopted in real CPS scenarios.
- Existing simulation tools are limited; their capabilities to jointly model physical and cyber components are lacking, which poses a significant threat to the accuracy of CPS.

III. A Research Roadmap

Based on the results of our literature review and empirical study, we posit that runtime verification, tailored to the special needs of cyber-physical systems offers a reasonable and feasible complement to existing trial and error testing approaches in CPS, where developers are already tuned to detecting and potentially responding to faults in real-time. Enabling CPS developers in employing such techniques requires providing:

- an easy and intuitive way for CPS developers to specify desired correctness properties of a CPS application;
- mechanisms to incorporate properties of the physical world into the runtime verification process; and
- a usable tool suite that is both integrated with the development process and approachable to CPS developers who may not be expert software engineers.

Figure 1 shows the *Brace* architecture, which fits these components together. CPS developers use an *assertion* language to annotate a CPS program with desired correctness guarantees. This annotated program is passed through the *Brace* compiler to the runtime framework, which uses a set of tools that model and connect to the physical aspects of a CPS to generate a deployable CPS program. *BraceForce* refers to a tool that provides access to physical information from sensors and actuators deployed in the environment [21]; *BraceForce* is similar in nature to other tools that connect applications to sensing capabilities in the wild [3], [18]. *BraceBind* refers to a tool that connects the developer's correctness specifications either to outputs of these sensors and actuators or to models (e.g., simulations or numerical models) of physical properties.

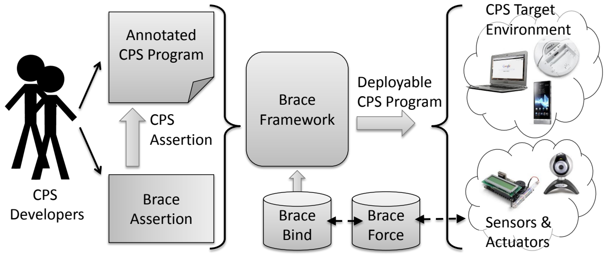

Fig. 1. The Brace Framework Architecture

We use this strawman architecture to elucidate research challenges in bringing practical runtime verification to CPS development. Our first step is to create a specification language for intuitively and expressively stating CPS assertions; this is *BraceAssertion* in Figure 1. Secondly, the state of art in runtime verification lacks a balance between expressiveness and efficiency [1]. Moreover, CPS has distinct requirements for runtime verification (e.g., the ability to represent both functional and timing behaviors). *Brace* is an online monitoring framework that address these issues. Thirdly, as found in our empirical study, physical properties cannot be simply ignored, and a robust approach to accounting for the intertwined physical and cyber components of a CPS must be addressed; this is *BraceBind* in Figure 1. Lastly, cyber physical systems exhibit stochastic attributes: sensors can be faulty, delays of actuation are random, and impacts of the deployment environment are often undetermined. Addressing this randomness is the last piece of our research roadmap.

Intuitive and Expressive Assertions. We envision an accessible approach to providing correctness specifications through *BraceAssertion*. A *BraceAssertion* allows a CPS developer to provide natural language statements of correctness properties that are automatically converted to a temporal logic with qualitative (e.g., eventually, always, eventually always, never) and quantitative (e.g., within, more than, after exactly k time units) expressiveness. The main idea is to turn this (signal clock) temporal logic into a (determinizable) monitor automaton for run-time verification. The formal specification must be accessible to CPS practitioners who are not logicians and may not understand formalisms such as temporal logics. To that end, our natural language specifications will be based on the paradigm of *Behavior Driven Development*[1]. Though the idea of using temporal logics to generate a monitor is not fundamentally new, we see an essential challenge in making the paradigm more accessible and therefore practically viable.

An Online Monitoring Framework. Fully supporting the complex properties necessary for CPS, e.g., capturing global properties that can only be checked by combining events and properties from a set of distributed nodes in the CPS [14], a major component of the Brace framework will be to develop time synchronization algorithms to deal with challenges associated with unreliable networks, to guarantee ordering of events from distributed nodes, to synchronize between (distributed) events and objects used for checking global properties. To meet desired requirements of CPS (e.g., minimum impact to the observed system and predictable behaviors of monitors irrespective of surges of events), the online monitoring framework will require optimization algorithms to dynamically adjust the behavior of runtime monitors. The ultimate goal is to create a practical runtime verification framework to replace the trial-and-error method that is currently adopted by CPS practitioners as the *de facto* standard.

Connecting to Real-time Simulation. Connecting the CPS application to the physical world through *BraceBind* is an essential piece of our strawman architecture. Since testing *in situ* is expensive and may not be reasonable for all CPS applications, connecting the CPS runtime verification framework to high-quality real-time simulation of physical processes will be a key enabler of a verification and validation tool suite for CPS. The integration of real-time simulation with the *Brace* online monitoring framework would enable models of physical components (e.g., sensors, actuators, battery, and environment) to be checked at runtime against the cyber part of a CPS in a simple and repeatable way. The research challenges that remain relate to quantifying the accuracy of the models and creating a programming interface that can act as a facade between the cyber components and physical data sources, whether they are sensors or real-time simulations. We envision a *cyber-model interface* that can be automatically synthesized from a simple specification of sensors and actuators. We also envision a *model-environment interface* that glues transducer models (e.g., of sensors, actuators, and resources like CPUs and batteries) with physical models (e.g., kinematic models) to enable runtime verification in varying deployment environments. Modelica [15], [10] provides both a cyber-model interface and a model-environment interface,

[1] http://dannorth.net/introducing-bdd

enabling implementing physics simulations with equations and introducing the execution of algorithms over these simulations. However, Modelica is not applicable to heterogenous models, which are required for CPS in general. Functional Mock-up Interface (FMI) [5] integrates simulation components across platforms but requires simulation tools to strictly support FMI function calls. The two fundamental challenges in integrated simulation (time synchronization and data integration) are left for developers to handle, which is complex and error prone. We intend to create a middleware to handle these challenges using a simple and approachable specification language. More specifically, one could imagine a tight integration of simulation models and tools (especially an off-the-shelf product like Simulink) with a CPS runtime monitoring framework to detect bugs that originate from the interaction between the cyber and the physical components of the system. The ultimate result would be a practical framework/tool for industrial CPS developers to do software-in-the-loop testing.

Addressing Uncertainties. Finally, a CPS runtime monitoring framework must support reasoning under uncertainty. Fuzzy logic enables altering the guarantees associated with models of physical components. For instance, instead of using a single observation to generate an event, we can perform temporal aggregation over a period of observations. In [11], the period of sampling, the weight of each observation, and the acceptance threshold were all set empirically. In contrast, we envision determining these parameters using machine learning and creating efficient monitor synthesis and verification algorithms for the fuzzy models. We should also evaluate alternatives to deal with stochastic nature of CPS systems. For example, using robust control [2], the control system (e.g., a car's braking system) should be robust to many situations (e.g., the conditions of the asphalt). Robust control identifies the uncertain parameters and their variations. Control strategies can be designed to be robust with respect to a variation in the parameters within the prescribed range. Applying this for CPS verification requires automating the identification of the parameters and their ranges. Another viable tool to deal with testing uncertainty of CPS is the scenario theory/approach, which has been used to test wind farm systems [19]. The idea is to define all the potential sources of uncertainty and perform a certain (very high) number of experiments on the real system. The number of experiments to be performed with random conditions for the uncertainty sources is given by a chance constrained problem. The result of the application of this approach is the formal guarantee that the system will do what is required with probability $1 - \epsilon$, where ϵ is very small. Our research in this direction is to automate a process to identify and solve a chance constrained problem (namely to find the number of experiments required for a given ϵ).

IV. Conclusions

In this position paper, we used an empirical study on debugging teting CPS to motivate a set of research challenges and presented a roadmap that highlights a set of challenges addressed by an integrated runtime monitoring framework designed with the goal of supporting CPS developers *in situ* as they attempt to verify their complex cyber physical systems. Key aspects of this roadmap include integrating simulation tools and fuzzy models into a tool for validating CPS.

Acknowledgments

Thanks to all participants in the empirical study and all collaborators in our ongoing research. 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 sponsor.

References

[1] H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In *Proc. of FM*. 2012.

[2] S. Bhattacharyya, H. Chapellat, and L. Keel. Robust control: the parametric approach. *Upper Saddle River*, 1995.

[3] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. V. Orden, and G. Borriello. Open data kit sensors: A sensor integration framework for android at the application-level. In *Proc. of Mobisys*, 2012.

[4] F. Chen and G. Roşu. Java-MOP: A monitoring oriented programming environment for Java. In *Tools and Algorithms for the Construction and Analysis of Systems*. 2005.

[5] Functional mock-up interface. http://www.fmi-standard.org.

[6] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Acm Sigplan Notices*, 2003.

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

[8] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive formal verification of an os microkernel. *ACM Transactions on Computer Systems (TOCS)*, 2014.

[9] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Deffin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. sel4: Formal verification of an OS kernel. In *Proc. of SOSP*, 2009.

[10] C. Knobel, G. Janin, A. Woodruff, and A. Modelon. Development and verification of a series car modelica/dymola multibody model to investigate vehicle dynamics systems. In *Proc. of Int'l. Modelica Conf.*, 2006.

[11] K. B. Lamine and F. Kabanza. Using fuzzy temporal logic for monitoring behavior-based mobile robots. In *Proc. of IASTED*, 2000.

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

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

[14] T. A. Moehlman, V. R. Lesser, and B. L. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group decision and Negotiation*, 1(2):161–191, 1992.

[15] L. Morawietz, S. Risse, T. Christ, H. Zellbeck, and H. C. Reuss. Modeling an automotive power train and electrical power supply for hil applications using modelica. In *Proc. of Int'l. Modelica Conf.*, 2005.

[16] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. 2002.

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

[18] A. Sani, K. Boos, M. Yun, and L. Zhong. Rio: A system solution for sharing I/O between mobile systems. In *Proc. of Mobisys*, 2014.

[19] M. Vrakopoulou, K. Margellos, J. Lygeros, and G. Andersson. A probabilistic framework for reserve scheduling and security assessment of systems with high wind power penetration. *IEEE Transactions on Power Systems*, 28(4):3885–3896, 2013.

[20] X. Zheng, C. Julien, M. Kim, and S. Khurshid. On the state of the art in verification and validation of cyber physical systems. Technical report, The University of Texas at Austin, 2014. http://goo.gl/VJg0lB.

[21] X. Zheng, D. Perry, and C. Julien. Braceforce: A middleware to enable sensing integration in mobile applications for novice programmers. In *Proc. of MOBILESoft*, 2014.