Physically Informed Assertions for Cyber Physical Systems Development and Debugging

Xi Zheng
The University of Texas at Austin
Email: jameszhengxi@utexas.edu

***Abstract*—Cyber Physical Systems (CPS), widely used in pervasive computing, integrate computation in the cyber world with control of physical processes. Developing CPS is challenging because interactions between physical and cyber components are complex and often unpredictable. Traditional debugging techniques can detect bugs in the cyber world, but bugs introduced from physical components and induced by limitations in the software interface to physical hardware are still difficult to detect. My research will capture the state of art and the state of the practice in verification and validation of CPS. Based on this, I will design middleware that combines models of the physical world with programming-language based assertions to help developers to design, develop, and debug robust CPS applications with ease.**

I. INTRODUCTION

Cyber Physical Systems (CPS) have gained popularity both in industry and the research community and are represented by many varied mission critical applications. Debugging CPS is important, but the intertwining of the cyber and physical worlds makes it very difficult. First, the connections between the debugging tasks and sensor platforms are tenuous. It is difficult (and at times impossible) to incorporate physical sensor readings into debugging rules and tools. This issue is further complicated by the heterogeneous nature of sensor platforms; different platforms require their own languages and operating systems making it challenging to add sensor information to the debugging task in a general purpose way. Second, sensors are often unreliable, and retrieval of sensor data usually incurs latency. Traditional debugging tools and techniques do not account for nondeterminism or the fact that the impact of a program action may not be instantaneously assessable in the physical environment. Third, the impact of a CPS application's actuation is often unexpected from the developers' perspective and influenced by the environment. Existing development tools do not make it straightforward for a developer to explicitly specify the program's assumptions about the operating environment. Lastly, the cyber world is discrete, while the physical world is continuous; this gap makes it nearly impossible to observe both cyber and physical correctness collectively.

CPS developers rely heavily on simulation to safeguard the correctness of CPS. As an example of the limitations of simulation, a vehicle in the 2007 DARPA Urban Challenge dangerously deviated from its computer-generated path and stuttered in the middle of a busy intersection. The bug was ultimately determined to be related to limitations of the steering rate at low speeds, which was undetected by the (sophisticated) simulation model used [1]. While simulation models may provide very good representations of the real world, they often fail to accurately represent the environment; playback of recorded traces in simulation environments suffers from a similar limitation in that it reduces the inherently continuous environment to a discrete one.

My work (i) quantitatively and qualitatively captures the state of the art and state of the practice in debugging CPS; (ii) creates a modeling language to capture continuous/discrete dynamics and asynchrony in CPS; (iii) lays the groundwork for *in situ* CPS debugging by enabling integrating sensing into CPS applications across different sensor platforms; (iv) creates a middleware that bridges the gap between the cyber and physical worlds using physics models that relate the developer's expected conditions (captured in logical variables) to real physical constraints; (v) creates a physically informed assertion language that allows developers to capture both cyber and physical correctness; and (vi) builds a debugging framework that checks CPS assertions at run time.

II. PHYSICALLY INFORMED ASSERTIONS

As an example to explain the key contributions, consider an autonomous vehicle tasked to move in a square [2].

I am conducting a qualitative and quantitative review of the state of the practice in verifying and validating CPS applications using a survey and interviews of CPS experts. Early results confirm our hypothesis that methods used today are *ad hoc* and lack rigor and repeatability. Further, CPS debugging processes usually entail deploying and measuring the system in each different target deployment environment, a manual, tedious, expensive, and sometimes dangerous process.

To address these debugging challenges, my work will develop *Brace*, a multi-component toolkit to support debugging CPS. Fig. 1 shows how Brace will support the specification, development, and debugging of CPS applications.

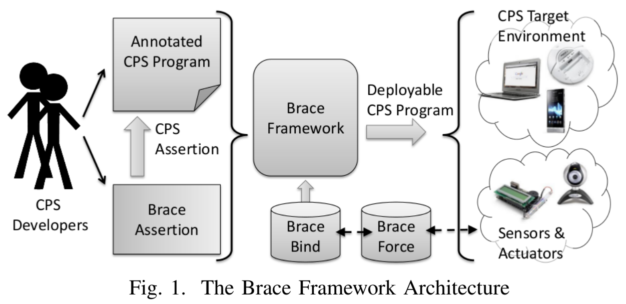

Fig. 1. The Brace Framework Architecture

CPSP. To capture the complex nature of continuous and discrete dynamics in CPS, we are creating a modeling language, Cyber Physical Systems Processes (CPSP) which is both simple to use and expressive enough to provide accurate modeling support for both development and debugging, including checking invariants at runtime. The intrinsic heterogeneity and complexity of CPS stresses existing modeling languages and frameworks [3]. For instance, a model of a CPS usually contains heterogenous models of various physical dynamic processes as well as models of computations, networks and software. Modeling those systems requires knowledge in and inclusion of various domains including but not limited to control theory, software engineering, networking, mechanic engineering, and electrical engineering. To address these challenges, CPSP will incorporate aspects of modeling techniques drawn from the domain of hybrid systems, in which continuous dynamics and discrete events coexist and interact [4].

Alur et al. [5] introduced Hybrid Automata, in which a hybrid system is modeled as a finite automaton with discrete and continuous variables. At a given location, the values of the continuous variables change over time according to the model's associated law. Each location is associated also with an invariant control condition. The transitions of the automaton are labeled with guarded commands. The original Hybrid Automata model has undergone many refinements, and is the most widely used model in the domain of hybrid systems [6]. Hybrid CSP [7] relies on continuous statements that were added to CSP to describe interactions among processes exclusively as communication. The syntax of Hybrid CSP, however, is not straightforward, which we believe will hinder its adoption. SHIFT [8], which is used to describe and simulate dynamic networks of hybrid automata, is, on the other hand, much more accessible. SHIFT is not as expressive as HCSP in terms of describing asynchrononous processes, which is crucial for complex CPS.

We will merge two languages into CPSP to enable the simplicity of being compatible with the object-oriented programming paradigm and the expressiveness for specifying continuous/discrete dynamics and asynchrony of CPS. We will provide a tool to map Java classes of existing CPS programs to *t*ypes; the resulting CPSP map file (based on XML) will be used by BraceBind and BraceAssertion (described below). As an extension to CPSP, we can apply the Model Driven Development (MDD) methdology [9], where an efficient Java application prototype can be automatically generated from CPSP models. The prototype will have pre-defined CPS assertions and can interact with the BraceBind library to access data from mathematical models and physical sensors. The main challenge of this research direction is how to generate efficient code in turns of performance, memory utilitzation, compilation time and system size [9].

BraceForce. CPS demand a tight yet flexible connection between a debugging environment's available sensing capabilities and the cyber world of the CPS application. To bring sensing into CPS applications, I created BraceForce [10], which integrates different sensor platforms with very little programming effort. In our example, BraceForce allows our autonomous movement application to connect to precise localization capabilities that may be available in the debugging environment but not in the deployment environment (e.g., overhead cameras). BraceForce not only supports the CPS debugging challenge that motivates my work but is also useful for general purpose CPS application development as these applications also often incorporate sensing into core application functionality.

BraceBind. In debugging CPS, developers must consider unreliable sensor data, latency in data retrieval, and unexpected or delayed results from actuation. I am creating BraceBind, a middleware that allows domain experts to connect CPSP models to CPS code.

BraceBind provides tool to take in a CPSP model and a related CPSP map file, and convert the model into a Java library, which converts the continuous dynamics in the model into Java functions/classes and maps discrete variables in the model either to static Java variables with pre-assigned values or Java variables linked to sensors, the values of which are accessed through BraceForce. The integration of CPSP models into CPS applications enables two channels of data to inform the debugging task: one from the raw sensor data and another computed from the models. While the raw data gives direct measurement of the debugging environment, the models bring the benefits that come from simulation: the ability to express expectations of the values. The former enables debugging in a target environment; the latter enables simulated debugging. Together, the two provide a complete, unified debugging suite. In our example, we use BraceBind to connect our program to the expected acceleration and steering capabilities of our particular robotic platform (i.e., the Roomba).

BraceAssertion. A key challenge in debugging is specifying expected behavior, for which assertions are commonly used. My research will define a rich assertion language, BraceAssertion to enable CPS architects to capture underlying assumptions and characteristics of the environment. For example the following code shows how a developer might use a BraceAssertion to specify that, after instantiating the `RobotMover`, the robot's location changed by 1 meter:

```
CPSAssert(distance(Sensors.before(Sensors.LOC),
                   Sensors.after(Sensors.LOC))
              == 1) {
  new RobotMover(speed, angle, duration);
}
```

We can define more expressive assertions, for example allowing *tolerance* in the assertion to account for both noise in sensing and error in actuation. I will explore various semantics of BraceAssertions based on my interviews with developers.

Besides inline annotation, we are also looking at the possibility of specifying BraceAssertion for state and transition invariants, or as pre/post conditions for specific function blocks in the CPSP models, where we can also introduce quantification (e.g. universal quantifier and existential quantifier). BraceAssertions provides tool to automatically place the assertions in the right place of the existing program by the help of CPSP map file.

Brace. The BraceAssertion will become the fundamental component of Brace [2], a run-time CPS debugging middleware, which validates CPS assertions at runtime. Evaluating the assertions can be based on input from sensors in the debugging environment (connected via BraceForce), input from the physical models (connected via BraceBind), or both. The assertion in the code snippet above shows one that takes input from BraceForce, indicated by the use of the `Sensors` package. Alternatively, using BraceBind would change `Sensors` to `Models` but leave everything else about the CPS code unchanged. In this sense, Brace allows the developer to validate the same system in multiple ways. Brace will also provide static analysis, for example, using physics models to invalidate assertions that are physically impossible.

III. Evaluation

To evaluate Brace and its constituents, I will use CPS applications from the autonomous vehicle domain and from a heart pump controller. For the purposes of this (brief) explanation, we focus on the autonomous vehicle. Evaluation of Brace will include both "unit testing" of the constituent components and "integration testing" of the entire Brace family, including user studies involving real CPS developers.

Evaluating BraceForce will assess the ease with which developers can access general-purpose sensor data and the overhead of performing this access through the BraceForce middleware. Evaluations of BraceBind will assess the impact of different degrees of expressiveness on both the quality of the results and the computational complexity of computing the models at run time. Finally, evaluations of BraceAssertions will be based on expressive power and developer's assessments of usability. When the entire Brace suite is assembled, I will conduct a user study that will ask CPS developers to write a variety of CPS assertions for the evaluation applications, with the aim of to answering these research questions:

1) How easy it is to use BraceAssertions to specify desired properties in the CPS applications? And at what cost?
2) How expressive is BraceAssertion by a grade scale (1-5, 1 as lowest) from professional programmers in the user studies?
3) Can Brace find errors quicker or find more errors compared with state of the art simulation tools (LabView and Simulink/Stateflow) and a model checking tool based also on hybrid automata (HyTech [11])?

I will also use benchmarks for the following metrics:

1) Runtime overhead imposed by Brace
2) Scalability of Brace (e.g., can Brace successfully check assertions distributed across multiple distributed cyber and physical components)
3) Fault localization (i.e., accuracy of bug detection)
4) Precision (i.e., false positive bug detections)
5) Recall (i.e., bugs undetected by Brace)

IV. Conclusion

I presented a middleware system that brings physically informed assertions to CPS development and debugging. Application architects can capture underlying assumptions and characteristics of the environment through assertions and will be able to develop and debug robust pervasive applications with ease across different deployment environments.

Acknowledgement

I thank my advisor, Christine Julien, for her guidance and support. This work was supported by the NSF CNS-1239498.

References

[1] S. Mitra, T. Wongpiromsarn, and R. M. Murray, "Verifying cyber-physical interactions in safety-critical systems," *IEEE Security and Privacy*, vol. 11, no. 4, pp. 28–37, 2013.
[2] X. Zheng, C.-L. Fok, C. Julien, S. Khurshid, and M. Kim, "Brace: Assertion-driven development of cyber-physical systems applications," *Tech. Report TR-ARiSE-2013-001, University of Texas at Austin*, 2013.
[3] P. Derler, E. A. Lee, and A. L. Sangiovanni-Vincentelli, "Addressing modeling challenges in cyber-physical systems," DTIC Document, Tech. Rep., 2011.
[4] D. Liberzon, *Switching in systems and control*. Springer, 2003.
[5] R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," in *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*. Springer, 1994, pp. 329–351.
[6] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.
[7] Z. Chaochen, W. Ji, and A. P. Ravn, "A formal description of hybrid systems," in *Hybrid Systems III*. Springer, 1996, pp. 511–530.
[8] A. Deshpande, A. Gollu, and L. Semenzato, "The shift programming language for dynamic networks of hybrid automata," *Automatic Control, IEEE Transactions on*, vol. 43, no. 4, pp. 584–587, 1998.
[9] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, no. 5, pp. 19–25, 2003.
[10] X. Zheng, D. E Perry, and C. Julien, "Braceforce: A middleware enabling novice programmers to integrate sensing in cps applications," *Tech. Report TR-ARiSE-2013-003, University of Texas at Austin*, 2013.
[11] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in *Computer aided verification*. Springer, 1997, pp. 460–463.