Teaching CPS Foundations With Contracts

André Platzer
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
aplatzer@cs.cmu.edu

***Abstract*—We describe the experience with courses that teach the *Foundations of Cyber-physical Systems* (CPS) and methods for ensuring the correctness of CPS designs. CPSs combine cyber effects (computation & communication) with physical effects (motion or other physical processes). CPS represent a paradigm shift that transcends the separation of computer science, which traditionally focuses on computation & communication isolated from the physical world, versus engineering and physics, which traditionally focus more on physical effects than on software intensive solutions. CPS are a unique challenge and unique opportunity for education. They challenge, because of their mathematical demand and interdisciplinary nature. CPS are an opportunity, because students learn important insights about the interface with other areas and develop a deeper understanding about the principles that make cyber and physical aspects work together. The course addresses the challenges of designing CPS that people can bet their lives on by emphasizing CPS *contracts*.**

I. Introduction

Cyber-physical systems (CPS) are systems combining cyber effects (computation and/or communication) with physical effects (motion or other physical processes). Cars [1], aircraft [2], and robots [3] lead to prototypical examples, because they move physically in space and how they move is determined by computerized control algorithms that may even exchange information with their environment. CPS are a general solution concept, however, which extends to many other areas, including power plants [4], the whole power grid [5], or medical CPS [6]. Since CPS perform critical tasks [7], the most important question about them has been aptly phrased by Jeannette Wing:

> How can we provide people with cyber-physical systems they can *bet their lives on*?

To realize the importance of this question, the reader is advised to imagine him or herself taking a ride in a self-driving car and wondering whether it would be safe to read a book or take a nap instead of watching out for how the autonomous car deals with the oncoming traffic.

In this paper, we describe our view on how CPS education can help reach this goal. This view has crystallized based on our experience with two different versions of a course that teaches CPS with an emphasis on foundations and methods for ensuring the correctness of CPS designs. Both have been offered in the Computer Science Department and the Electrical and Computer Engineering Department at Carnegie Mellon University, with about equal participation from both backgrounds. Our view is also based on the experience with mentoring around two dozen undergraduate, master, Ph.D. students, postdocs, or domain experts in the field. This view is, furthermore, based on the ongoing design of a new CPS course exclusively for undergraduates,[1], about which it is too early to report on results, however.

This course will likely become easier to teach in succession with programming courses that focus on contracts as the one developed at CMU [8]. But since that course has only recently been introduced into the CMU curriculum, we have not yet had the opportunity of observing the result of such a succession.

II. CPS Foundations Education

With CPS principles and technologies becoming increasingly important in a growing number of areas, the only hope for a successful and enduring CPS education that is true to the CPS ideals of an overarching science is proper attention to the CPS foundations. Solid foundations are not just the part of CPS with the most enduring impact in an ever-changing world of CPS applications during the student's careers. Foundations are also required to master the formidable challenges that CPSs provide. The education in most science and engineering subjects starts with foundations. CPS are not an exception. It is easier to grasp the specifics of a particular new CPS application based on a solid understanding of the foundations of CPS.

At the same time, a significant fraction of the CPS education depends on developing the right intuition for the relevant behaviors of systems, for their critical properties, and for an adequate assessment of the difficulties involved in ensuring that models of the system will perform as expected. Developing this intuition would be difficult, if not impossible, without concrete CPS applications. One way of ensuring a CPS view across application domains would be to study multiple application domains in depth, e.g., during subsequent lectures. In our experience, however, a deeper appreciation for the commonalities across CPS can be obtained if part of the learning objective in homework and lab assignments or projects is this study of application domains that first look different on the outset, only to uncover their closely related mathematical core as part of the assignment or project. Since many CPS applications have a component of equations of motion from Newtonian mechanics, commonalities are inevitable to be found during the study.

The most challenging aspects of CPS education are:

1) The high demand for mathematical sophistication cutting across numerous areas of mathematics, which are traditionally less seamlessly integrated;
2) The diversity of backgrounds from people interested in CPS, which directly reflects the large number of disciplines from which CPS draws ideas and relates to;

[1]Course material available at http://symbolaris.com/course/fcps13.html

3) The intellectual challenges of understanding and identifying the complex interactions that different aspects of the CPS design have on its functioning;
4) The difficulties with making CPS designs safe, which is still a grand challenge in research, but whose current solutions have to transition into widespread practice regardless, because today's system designs need to be made safe today and not just patched after a problem has already occurred in practice, which might harm people.

We argue that challenge 1 is best addressed by a very gradual introduction of the most important principles and intuitions of the required mathematical background alongside a growing set of practicing challenges with pointers for self-study on demand. Challenge 2 is a challenge for the instructor, but one that turns into a welcome opportunity, because it is part of the special learning experience of CPS to have a cross-disciplinary audience in which members from each discipline bring different solutions and different ways of thinking to lectures, recitations, and labs. This phenomenon, plus possible collaboration on projects across disciplines, gives students an appreciation for the values and styles of other disciplines and enables them to communicate more effectively across disciplines, which is one of the challenges in CPS practice. Challenge 2 is alleviated somewhat by focusing on principles and intuition for the mathematical background material and referring to on-demand background reading material, e.g., [9]. According to informal feedback from the students, differences based on prior background tend to deteriorate after a few lectures, where reasonably balanced challenges arise. Challenge 3 is addressed by giving an overview of each of the most characteristic features of CPS and how they can be modelled and analyzed over the course of the semester. We reserve an in-depth study for a few aspects, mostly the interaction of discrete and continuous phenomena [7]. But it is crucial that CPS students are exposed to all characteristic fundamental features in order to be prepared and sensitive to the specific issues of each phenomenon, once they encounter them in practice. Challenge 3 is simplified substantially by the multi-dynamical systems view underlying our approach [10]–[12]. Challenge 4 is significant, but central to the success of CPS. The appreciation for safety technologies and correctness methods for CPS is easy to convey with the perspective of having to trust our lives to a CPS and in light of a number of infamous historical examples. The understanding how to approach correctness of CPS designs can, however, be taught as well. Towards the end of our courses, for example, most students have been able to formally verify interesting CPS. A large number of the students have come up with impressive verification results in final projects, most of which were entirely self-defined, e.g., about distributed elevator controls, (simplified) TCAS-type aircraft collision avoidance maneuvers, or CICAS-type car controllers for intersections, or statistical sampling based verification procedures.

While there is unbounded potential for getting any other part of a CPS implementation wrong, the most devastating consequences often come from oversights in the high-level system or control design. This, along with the fact that the most comprehensive overview of CPS phenomena can be conveyed starting from those levels, leads us to focus CPS education efforts primarily on CPS control designs in this course.

III. Learning Objectives

There are many interrelated learning objectives for this 15-week course. In analogy to programming language education [8], we organize the objectives along the dimensions: *modeling and control*, *computational thinking* [13], and *CPS skills*.

A. Modeling and Control

In the area of *modeling and control*, successful students will

- **understand the core principles behind CPS.** They are important for effectively recognizing opportunities how the integration of cyber and physical aspects can solve problems that no part could solve alone.
- **develop models and controls.** In order to understand, design, and analyze CPS, it is important to be able to develop models for the various relevant aspects of a CPS design and to design controllers for the intended functionalities based on appropriate specifications.
- **identify the relevant dynamical aspects.** It is important to be able to identify which types of phenomena of a CPS have a relevant influence for the purpose of understanding a particular property of a particular system. These allow us to judge, for example, where it is important to manage stochastic effects, or where a nondeterministic model is more adequate.

B. Computational Thinking

In the area of *computational thinking*, successful students should be able to

- **identify specifications and critical properties.** In order to develop correct CPS designs, it is important to identify what "correctness" means, how a design may fail to be correct, and how to make it correct.
- **understand abstraction and system architectures.** They are essential for the modular organization of CPS, and for the ability to reason about separate parts of a system independently. Because of the overwhelming practical challenges, abstraction is more critical than in software design. A formal treatment of CPS architectures [14] is beyond the scope of this course.
- **express pre- and post-conditions and invariants for CPS models.** Pre- and post-conditions allow us to capture under which circumstance it is safe to run a CPS or a part of a CPS design, and what safety entails. They allow us to achieve what abstraction and hierarchies achieve at the system level: decompose correctness of a full CPS into correctness of smaller pieces. Invariants achieve a similar decomposition by establishing which relations of variables remain true no matter how long and how often the CPS runs.
- **use design-by-invariant.** In order to develop correct CPS designs, invariants are an important structuring principle guiding what the control has to maintain in order to preserve the invariant. This guidance simplifies the design process, because it applies locally at the level of individual localized control decisions that preserve invariants without explicitly having to take system-level closed-loop properties into account.
- **reason rigorously about CPS models.** Reasoning is required to ensure correctness and find flaws in a CPS

design. Both informal reasoning and formal reasoning in a logic are important objectives for being able to establish correctness.

- **verify CPS models of appropriate scale.** Formal verification helps finding and fixing bugs and proving correctness, which is helpful in all stages of the CPS design. Verification is not only critical but, given the right abstractions, surprisingly feasible in high level CPS control designs.

C. CPS Skills

In the area of *CPS skills*, successful students will be able to

- **understand the semantics of a CPS model.** What may be easy in a classical isolated program becomes very demanding when that program interfaces with effects in the physical world. A full treatment of, e.g., the semantics of stochastic CPS effects is better placed in a specialized course. But understanding the meaning of a CPS model with fewer dynamical aspects and know how it will execute is fundamental to reasoning.
- **develop an intuition for operational effects.** Intuition for the joint operational effect of a CPS is crucial, e.g., about what the effect of a particular discrete computer control algorithm on a continuous plant will be.
- **use higher-level model-predictive control.** The design of many CPS can be guided systematically by adopting a higher-level model-predictive control style in which verification and design go hand in hand [12]. This successively replaces constraints on the future evolution of the system by conditions on the current state that ensure them.

IV. CPS Program Models

We are convinced that CPS design is inseparably linked to computational thinking. Whenever the CPS design needs to be correct, it is hard to achieve that goal without a sufficient understanding of the relevant safety conditions and careful attention to the invariants maintained during the system run.

A. Hybrid Programs

The cornerstone of our course design are *hybrid programs* (HPs) [9]–[12], which capture relevant dynamical aspects of CPS in a simple programming language with a simple semantics. One important aspect of HPs is that they directly allow the programmer to refer to real-valued variables representing real quantities and specify their dynamics as part of the HP. The control structure of HPs supports simple regular expression style operators for nondeterministic control as in Kleene algebras [15]. HPs support sequential composition $(\alpha;\beta)$ of HPs and nondeterministic branching by nondeterministic choices $(\alpha \cup \beta)$, which will nondeterministically run either α or β. HPs support nondeterministic repetition (α^*), which runs HP α repeatedly any number of times. Because CPS deal with the uncertainties of the real world, nondeterminism is more important for modeling accuracy than in isolated computer programs. Conditionals (**if**$(H)\,\alpha$ **else** β) and loops (**while**$(H)\,\alpha$ etc.) are supported as abbreviations based on test statements $(?H)$, which test the condition expressed by logical formula H in the current state similar to an assert or assume. The most important aspect of HPs, however, is that, besides assignments $(x := \theta)$ of the value of expression θ to variable x, HPs support *differential equations* $(x' = \theta \,\&\, H)$. The latter represents a continuous evolution that, at the time of execution, makes the system follow this differential equation $x' = \theta$ any arbitrary nondeterministic amount of time, provided the system stays in the *evolution domain* characterized by logical formula H during this entire continuous evolution.

The extension to the domain of real-valued variables and the addition of differential equations as primitive operations in the middle of the program are the most fundamental extensions making HPs a proper programming language for CPS. But nondeterminisms of HPs are another important feature required for the adequacy of CPS models. Over the course of the semester, further extensions of the programming language of HPs are revealed in subsequent layers for other dynamical aspects of CPS [9], [12], including disturbance, nondeterministic inputs, continuous-time inputs, aspects of distributed hybrid systems, and a brief glimpse at stochastic effects. For lack of space, those will not be discussed here.

B. CPS Contracts

The design of the CPS contracts language for HPs loosely follows the contract languages for conventional programming languages, especially of the C0 programming language [8], JML [16], and Spec# [17]. Preconditions for HPs are expressed as **@requires**(H) where H is a (simple) logical formula that has to evaluate to true before the HP runs. It is an error to start an HP in an initial state where H evaluates to false. Preconditions capture the requirements on the state in which it is safe to run a HP. Postconditions for HPs are expressed as **@ensures**(H) where the HP is considered to be unsafe if it can lead to a state in which H evaluates to false. Postconditions capture the safety properties that a HP enforces.

The following is a simple example of a HP with a contract. It is a simple model of a vehicle at position x moving with velocity v and acceleration a along a one-dimensional line.

```
@requires(v^2 ≤ 2*b*(m–x))
@requires(v ≥ 0 ∧ A ≥ 0 ∧ b>0)
@ensures(x ≤ m)
  {
    if (v^2 ≤ 2*b*(m–x) – (A+b)*(A+2*v)) {
        a := A
    } else {
        a := –b
    }
    t := 0;
    {x'=v, v'=a, t'=1, v ≥ 0 ∧ t ≤ 1}
  }* @invariant(v^2 ≤ 2*b*(m–x))
```

This HP model repeats (as indicated by the repetition * in the last line) a control loop whose controller first sets the acceleration a (according to the **if** statement), then a clock t is reset to zero, and then the system follows the differential equation $x' = v,\ v' = a,\ t' = 1$ for an arbitrary amount of time, yet at most as long as indicated by the evolution domain $v \geq 0 \wedge t \leq 1$. In particular, the vehicle will only move continuously for at most 1 time unit and at most as long as $v \geq 0$, before the loop repeats and the control executes again.

The controller checks the safety-critical **if** condition $v^2 \leq 2b(m-x)-(A+b)(A+2v)$ and chooses positive acceleration ($a := A$) if it evaluates to true, and braking ($a := -b$) otherwise. This condition checks whether it is safe to accelerate and is one of the most important decisions in the CPS design. Systematic CPS design constructs it by high-level model-predictive control from the invariant [12]. The most critical part of a CPS system design are the invariants. Invariants for HPs are expressed as **@invariant**(H) attached to a loop, where H is a logical formula that evaluates to true every time that loop is run. Invariants can be attached by **@invariant**(H) to a differential equation (not shown), where H evaluates to true always when following the differential equation. Invariants and their corresponding safety conditions capture the most critical elements of a CPS design and convey the most informative insights about what we can rely on no matter how long and in which way the system will evolve.

Reasoning why H is an invariant, either of a loop in a CPS or of a differential equations in a CPS, is part of the learning objectives. The course first uses informal arguments and later shifts to formal proofs. Techniques for generating invariants automatically [18] have been implemented in the verification tool KeYmaera. But it is bad style not to include the most crucial design choices in a CPS design: invariants. Furthermore, we are convinced that designing any CPS on any level needs to work with at least an intuitive understanding of preserved properties of the system behavior, which should, consequently, be formally captured as invariants that are part of the CPS design for subsequent use in the overall design and implementation process rather than having to be rediscovered.

Contracts are ultimately rendered in *differential dynamic logic* ($\mathsf{d}\mathcal{L}$) [9]–[12] for formal specification and verification purposes and used for finding a proof. Since correctness and proofs are an important part of CPS design, and $\mathsf{d}\mathcal{L}$ is crucial for high-level model-predictive control [12], $\mathsf{d}\mathcal{L}$ and its proof calculus are discussed in the course, but only after an initial emphasis on CPS models, specifications, and contracts.

V. Conclusions and Future Work

We have given an overview of a new course on the *Foundations of CPS*, of which we have taught different versions to a mix of students from quite different areas. The course has a focus on CPS foundations, resting on an integration of principles from computational thinking, modeling and control, and CPS skills. The most exciting features of the course are that it shows a smooth and accessible way of understanding the most important essence of CPS in successive layers of a CPS programming language, while, simultaneously putting an emphasis on correctness, contracts, and CPS reasoning. We believe that it is the explicit focus on contracts and the simplicity and elegance of their formal basis in logic [11] that allows this course to be successful.

Our experience with the course has been very positive. We believe that a clear focus on the true essentials of CPS without the complexity and distractions, e.g., of controller implementations in a low-level system programming language like C, have helped students reach the learning goals of this course. Results from the online course evaluations conducted by the university have been encouraging, e.g., with an average total of 4.77 on a scale of 1 to 5, with 5 being the best.

Future work includes the development of a richer course toolset, e.g., for CPS simulation and verified code synthesis.

Acknowledgment

Many have contributed to the course. We would especially like to thank Jan-David Quesel for his help with developing the KeYmaera verification tool, Stefan Mitsch, who has contributed to the wider tool infrastructure, and thank the students in previous instances of the course, whose feedback and interactions have helped shape this course. This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246 and NSF EXPEDITION CNS-0926181.

References

[1] A. Deshpande, A. Göllü, and P. Varaiya, "SHIFT: A formalism and a programming language for dynamic networks of hybrid automata," in *Hybrid Systems*, ser. LNCS, P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds., vol. 1273. Springer, 1996, pp. 113–133.

[2] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: a study in multi-agent hybrid systems," *IEEE T. Automat. Contr.*, vol. 43, no. 4, pp. 509–521, 1998.

[3] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: from verification to falsification by combining motion planning and discrete search," *Form. Methods Syst. Des.*, vol. 34, no. 2, pp. 157–182, 2009.

[4] G. K. Fourlas, K. J. Kyriakopoulos, and C. D. Vournas, "Hybrid systems modeling for power systems," *Circuits and Systems Magazine, IEEE*, vol. 4, no. 3, pp. 16 – 23, quarter 2004.

[5] S. Sridhar, A. Hahn, and M. Govindarasu, "Cyber-physical system security for the electric power grid," *Proc. IEEE*, vol. 100, no. 1, pp. 210 –224, 2012.

[6] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. L. King, M. Mullen-Fortino, S. Park, A. Roederer, and K. K. Venkatasubramanian, "Challenges and research directions in medical cyber-physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 75–90, 2012.

[7] R. Alur, "Formal verification of hybrid systems," in *EMSOFT*, S. Chakraborty, A. Jerraya, S. K. Baruah, and S. Fischmeister, Eds. ACM, 2011, pp. 273–278.

[8] F. Pfenning, T. J. Cortina, and W. Lovas, "Teaching imperative programming with contracts at the freshmen level," 2011.

[9] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer, 2010.

[10] ——, "Differential dynamic logic for hybrid systems." *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[11] ——, "Logics of dynamical systems," in *LICS*. IEEE, 2012, pp. 13–24.

[12] ——, "Dynamic logics of dynamical systems," *CoRR*, vol. abs/1205.4788, 2012.

[13] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[14] A. Rajhans, A. Bhave, S. M. Loos, B. H. Krogh, A. Platzer, and D. Garlan, "Using parameters in architectural views to support heterogeneous design and verification," in *CDC*, 2011, pp. 2705–2710.

[15] D. Kozen, "Kleene algebra with tests," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 427–443, 1997.

[16] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll, "Beyond assertions: Advanced specification and verification with JML and ESC/Java2," in *FMCO*, ser. LNCS, F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, Eds., vol. 4111. Springer, 2005, pp. 342–363.

[17] M. Barnett, M. Fähndrich, K. R. M. Leino, P. Müller, W. Schulte, and H. Venter, "Specification and verification: the Spec# experience," *Commun. ACM*, vol. 54, no. 6, pp. 81–91, 2011.

[18] A. Platzer and E. M. Clarke, "Computing differential invariants of hybrid systems as fixedpoints," in *CAV*, ser. LNCS, A. Gupta and S. Malik, Eds., vol. 5123. Springer, 2008, pp. 176–189.