Preprint: CPS-Ed 2013, April 8, 2013

Lessons Learned in Teaching a Complex Distributed Embedded System Project Course

Philip Koopman
Carnegie Mellon University, ECE Dept.
Pittsburgh, PA, USA
Koopman@cmu.edu

***Abstract*— Teaching Cyber-Physical System (CPS) design requires covering significant breadth while ensuring students experience how all the pieces fit together. This paper describes a distributed elevator control system design project that addresses many of the areas required in a CPS project experience. Mapping project aspects to ABET accreditation areas frames a discussion of the course's treatment of CPS issues. The most important lesson learned is that students benefit from being immersed in and reflecting upon a carefully curated experience with a CPS engineering process rather than being turned loose to invent their own ad hoc approach to building complex systems.**

Keywords*— *Cyber-Physical System (CPS) education; distributed embedded system course project.

I. INTRODUCTION

Teaching Cyber-Physical System (CPS) design requires covering a broad range of topics in the areas of computer hardware design, software design, mechanical system design, and control design, as well as potentially covering related disciplines such as system safety, user interface design, and dependability. While it is unrealistic to expect a single college course to teach everything needed, it is likely that a significant unified project experience can be instrumental in helping students integrate the aspects of CPS design. An important question is how to do this in a scalable way while ensuring that students learn the right lessons.

This paper describes a semester-long project in a distributed embedded system course that has been taught at Carnegie Mellon University since 1999 [1] with class sizes of up to 70 students. While it is not intended to be a perfectly balanced CPS experience, it addresses many of the required points, especially in the area of methodical engineering processes. This paper describes the project in terms of how it addresses ABET accreditation goal areas [2] in the context of CPS education. A "lessons learned" section reflects upon problems that are likely to face significant CPS project courses.

II. PROJECT SUMMARY

The project is the design of a distributed elevator control system that runs entirely in simulation on a discrete event simulator. The elevator is designed as a fine-grain distributed system, with a separate (simulated) CPU allocated to every instance of every button, light, door, and so on. While real elevators are distributed systems, this project elevator intentionally exaggerates the number of processors to expose students to the issues that arise in complex distributed systems.

The project is not intended to be an open-ended capstone design exercise in which students determine their own project goals and concentrate on achieving an aggressive technical result. Rather, it is designed to be a carefully curated design experience in which students more or less follow the same process, demonstrate a certain set of technical skills, and experience a core set of design process challenges.

A. The Elevator Project

The elevator is based on the author's experience as part of an industry elevator architecture team, and includes sufficient detail to be a design experience that captures (often in simplified form) a wide range of industry-relevant CPS design considerations. A key part of the project is that simulated passengers ride the elevator, providing a self-contained way to exercise and evaluate the system's behavior. Examples of the details involved include door reversal triggering that varies based on passenger size, sounding a buzzer to get some passengers to exit an overweight elevator, randomized time constants on passengers behaviors such as button presses, and passengers who only enter and exit when car lanterns and floor indicators display correct information.

Elevator kinematics models include door motion and a multi-speed main motor with a specified acceleration profile. The elevator must serve floors in response to hall call and car call buttons, re-level as necessary due to weight-induced cable stretch, and attempt to maximize passenger satisfaction as a function of waiting time and other factors. A simulated electromechanical safety system asserts an emergency stop if the elevator moves out of the leveling zone with doors open.

Some parts are already done to ensure the project presents a reasonable student workload, such as pre-made passengers, a main drive kinematics model, test scaffolding, a graphical user interface, and starting points for some complex functions. A significant simplification is avoiding the implementation of elevator modal behaviors (e.g., firefighter service). But, overall, the simulation is complex enough to force students to think things through – and to discover behaviors of real elevators they'd never noticed before. Most importantly, the behaviors are complex enough that it is virtually impossible for a non-elevator expert to implement the nuances required for successful operation without following a methodical design process rather than just diving in to write code.

The goal of the course is a 12-hour weekly total student workload including a pair of two-hour lectures per week. Lectures cover material most students have not encountered before this course, primarily in the areas of system engineering process, embedded networking, and critical system design.

The project is organized as a set of weekly deliverables of increasing complexity that follow a defined process flow. Students work in teams of 3 or 4, and must have a very simple elevator operational by the middle of the course which just stops at every floor. After students understand basic elevator operation and the simulator framework, they spend the second half of the course designing a relatively sophisticated dispatching and control system to optimize passenger satisfaction without violating stated high level design requirements. Some of these high level requirements are obvious (e.g., don't cause an emergency stop). But some requirements are passenger preferences that are somewhat arbitrary from a purely technical point of view (e.g., once a car lantern has announced an elevator is "going up" it is not allowed to travel down without first going up to a higher floor).

Students must demonstrate that they meet the requirements by building in run-time monitors to ensure none of ten high level requirements is violated at any time during simulated workloads involving dozens or hundreds of randomly generated passengers. Final projects grades are based largely on the quality of the design package per grading rubrics, emphasizing completeness and end-to-end traceability from requirements through acceptance tests (and every step in between), with performance weighted less than design quality.

III. Mapping to Outcomes

The call for papers for this workshop proposed an adaptation of the ABET accreditation standards [2] to more specifically address CPS education, discussed below.

A. Applying mathematical models

We teach an evolutionary improvement of common embedded industry practices. To do this, we have pieced together relatively well known techniques from embedded system design and software engineering rather than attempting a unified CPS mathematical modeling framework.

Our most important mathematical model is deadline monotonic scheduling theory applied to network performance. We additionally require students to use physics knowledge to predict the "commit point" at which a moving elevator has to decide whether to initiate deceleration to stop at a floor vs. bypass that floor. There are interactions between the jittery periodic message delivery of the real time communication system and the possibility of overshooting a floor due to communication latency or randomized message corruption that expose students to the need to build in design margins to account for timing variations.

We mandate a purely time triggered design with some jitter due to the use of deadline monotonic scheduling. The addition of a combined event-based and time triggered model might expose additional design issues to students. We have found that most students naturally think in terms of event-triggered systems, so we intentionally push them to a purely periodically run system to teach that way of thinking about time.

Beyond purely mathematical models, students use a lightweight Unified Modeling Language approach to model and refine elevator requirements, architecture, and design. We use a Use Case Diagram, Sequence Diagrams, and Statecharts. We use a modified class diagram notation that includes hardware/software allocation to document the architecture.

It is surprisingly difficult to know whether a CPS is completely working rather than almost working. This is especially true when some requirements involve *how* overall goals are accomplished rather than *whether* they are accomplished. We use a runtime monitoring approach in which students add state-machine based monitors to their elevator to ensure that no elevator requirements are violated during simulations. Our experience is that without such monitors almost every elevator violates requirements without students realizing it. Left to their own devices, students tend look at whether passengers are delivered rather than whether specified functional requirements are violated in an otherwise "working" elevator. Having a rigorous model of correct system behavior that can be used for acceptance testing is just as important as having a model from which to build the system itself.

B. Design and conduct simulations

A customized discrete event simulator is used to implement the elevator. Real hardware is not used in part because of the complexity and expense of providing dozens of CPUs to students, and in part to expose students to simulation as a technique for understanding system behavior. CPU speed and memory are not simulated, but network bandwidth and latency are. In this particular design, as in many embedded systems, almost all modules have very simple software that could easily be handled by a 16-bit processor running at a few MHz. The complexity of this system is in the interaction of the distributed components, so that is where the simulation fidelity is highest.

All interactions between computing elements occur via a CAN [3] network simulation. Sensors, actuators, and people also communicate via the same messaging facilities, except they bypass the CAN performance model. The simulator has a test framework that supports injecting messages to objects, and a way to record output messages to monitor test results. Students are required to run unit tests and integration tests with these test facilities. Acceptance tests are performed via delivering simulated passengers. Carefully designing their own tests at all levels is an essential part of students getting their elevators to pass the staff-created acceptance tests.

C. Good engineering practices and quality attributes

A core goal of the course is to have students experience using a very well defined engineering process rather than winging it as they might in a typical student project demo. Thus, the emphasis is on solid engineering rather than novelty, and the project experience is carefully curated by using a well-debugged, standardized project assignment. Some ambiguities and gaps in the materials are intentional and strategically included as part of a realistic experience. Keeping the project substantially the same every year minimizes gratuitous bugs.

Other aspects of creating a realistic system include: meeting explicit safety goals, dealing with user interaction issues (the passenger time constants are more representative of a retirement home and than a student dormitory), and fault tolerance to dropped network messages.

Lectures cover other important engineering aspects such as economics, ethics, and security, because they are not explicitly included in the project.

D. Multi-disciplinary teams

The student body is fairly homogeneous, consisting primarily of students with computer science and computer engineering backgrounds. Team members are encouraged to do at least some of every aspect of the project, although some specialization does occur within teams (e.g., test scripts, system builds, hand-in audits). Moving to a CPS hardware-based lab instead of a simulator-based lab would increase the opportunity for multi-disciplinary interaction, but would present scalability issues and add substantial hardware debug workload.

Student demographics make it challenging to create cross-disciplinary teams of 3 or 4 for the large class sizes we encounter (the most recent class had 68 students). Larger and more flexible projects would simplify attaining a good skills mix even if only a few students from a particular discipline enrolled in the class, but would require significant modification to the project and might not scale as well to large classes.

E. Formulate and solve engineering problems

The first half of the course presents a very restrictive project assignment and a rigorously enforced design process flow, including a defined architecture, high level requirements, requirements allocation to subsystems, detailed requirements, design, implementation, design reviews, unit test, integration test, acceptance test, and process quality monitoring. The main point of this part of the project is to get the students up to speed on process, technology, and domain knowledge. Therefore, the required elevator behavior is extremely simplistic.

The second half of the semester is still highly structured, but is much less constrained. Students are free to use any technical approach to meet ten specified behavioral requirements, with bonus points given both for passenger delivery performance and having a high quality design package. A change of maximum main drive speed is made about three quarters of the way through the semester, which requires students to use a more sophisticated dispatcher to determine the elevator speed ramp-down point multiple floors ahead of the desired stop. This lets students experience three generations of changing requirements.

F. Life- and safety-critical systems

The elevator has a simulated electro-mechanical safety brake typical of one found in real elevators. Elevators must function in complex and heavy passenger workloads without tripping the safety shutdown monitor. Significant lecture content covers the topics of dependability, safety, and security, including safety-critical design standards. Including a stronger safety critical design experience would be appealing, but is beyond the time available in the current course.

G. Effective communication

Each student group presents a mid-semester and end of semester status report, highlighting design choices and lessons learned. These tend to focus mostly on software engineering concerns and simple metrics such as productivity and review effectiveness. Students often touch upon cyber-physical interactions in the form of physical performance optimization in the final project. If physical project aspects were strengthened, more time could be spent on cyber-meets-physical aspects. Every project team must also provide a comprehensive design package with complete traceability from requirements through design, implementation, and test.

H. Cyber vs. physical design decisions

The tradeoffs in this dimension are twofold. Students adjust the real time system schedule to ensure that control loops are tuned to the time constants of the physical portion of the system (primarily main motor speed, door speed, and user interface speed). A second potential tradeoff would be having safety performed by a physical system rather than a cyber system. Creating a credible software safety system has been explored on an individual basis by several students who use the elevator as a pilot system for PhD thesis work.

Opening up the main motor model to permit optimization of acceleration/deceleration profiles and energy use (including energy recovery from passenger/counterweight imbalances) would be a good CPS-oriented extension for this project.

I. Life-long learning

Gaps in textbook coverage, such as embedded networks, are covered by using practitioner-oriented articles rather than academic articles to acclimate students to using such publications to keep their skills up to date after graduation.

J. Contemporary issues

The course text [4] (and, indirectly, the course organization itself) is based on extensive field experience with design reviews. Additionally, “war stories” told in each class based on instructor experience serve to illustrate and motivate the material and issues that students need to be aware of. A more formal way of capturing and communicating such issues from a variety of sources would aid scalability in this area to other courses, but would be a significant undertaking.

K. Cyber-physical techniques, skills, and tools

The course uses an end-to-end design process that is representative of (or, in many cases, more rigorous than), a typical small-system industrial control project. Tools are quite lightweight, including drawing UML diagrams with whatever tools the students are comfortable, and using spreadsheets for traceability. Therefore, the emphasis is more on understanding how pieces fit together and learning basic representations of ideas rather than on tool proficiency. Students are encouraged to use auxiliary tools such as a version control tool and customized scripts to manage project portfolio assembly and hand-in. In large part this philosophy is driven by an observation that end-to-end tool chains are generally not in use in smaller industry projects for a variety of reasons.

IV. Lessons Learned and Practicalities

While the current project is remarkably similar to the first time the course was taught at a high level, just about everything in it has been overhauled over the years in response to an evolving understanding of what works, what doesn't work, and what benefits students the most given limited time and resources for both faculty and students.

A. Tools

We have our own simple discrete event simulator written in Java and a custom GUI that are reasonably platform independent, and require no proprietary tools. While this has cost us time to write and rewrite things, it also minimizes tool learning curves, license hassles, tool/project mismatch, version incompatibilities, and tool obsolescence problems that would have otherwise occurred. Adoption of a newer simulation tool is a possibility, but there is some virtue in having the simplest simulation framework necessary even if it is limited.

Making students perform all the steps more or less by hand helps them to see what goes on "under the hood." With a more multi-disciplinary student body one might have every student use an integrated CPS toolset while getting more in-depth exposure to what goes on behind the curtain in selected specialty areas. If we were to adopt an integrated CPS-specific design tool, it would have to be adapted to support the distributed system nature of our project, including test frameworks, precompiled opaque solution/scaffolding modules, detailed network simulation, support for complex non-CPS objects such as human passengers, and so on.

B. Design process vs. technology

Over the years we have found via student and employer feedback that our most valuable contribution is giving students an experience in which they come to realize, on their own, that sound methodical engineering practices are worth doing regardless of how smart the student may be. We've achieved this by intentionally creating a project that is too complex and subtle for most students to get right without using a methodical design process. Students who try to brute force things by writing code first and documenting later usually have a catastrophic experience (we help them recover before the end of the project). Many students list "skipping design steps bites you later" as one of the things they learned in the end-of-course retrospective presentations, often with compelling stories from their course experience to illustrate the point.

We also have increasingly had success by requiring the students to monitor and comment upon a few simple process metrics. For example, using a simple spreadsheet to standardize peer review reports and requiring a tally of defects found in each peer review has had dramatic effect. Student effort has shifted to earlier weeks (to fix bugs earlier) and the ratio of bugs found in peer reviews increased from 10% to about 50%. Most groups now have a light hourly load the last week of the project because the big bugs are being found early. After that change, students started listing one of the big things they learned in the course as being "peer reviews really work."

One of the open challenges with this course is individual assessment. We have tried many approaches to measuring individual performance within teams, and have found that most students game any grade-related metrics heavily. We now just concentrate on detecting dysfunctional teams via meetings and a weekly effort report. An essential ingredient in team success is a weekly 30-minute project management meeting for each group with a Teaching Assistant (TA). We have found that it is essential for each TA to have taken the course previously to appreciate the project issues that arise.

V. Conclusions

After more than a dozen years teaching distributed embedded systems, we've created a project that scales to large class sizes and provides a robust design experience without being a "killer project" course. It is strong in engineering process, but is somewhat software-centric, with other non-software aspects present as second-class citizens. The complete set of project materials, including the simulation framework, are publicly available on the course web site [1].

Based on our experiences, it is essential to teach not just "good CPS design" but rather "good engineering design" as applied to a CPS project that is complex enough that students must to follow a good process to succeed.

We have opted for a do-it-yourself tool set that has served us well, but as off-the-shelf CPS tool chains mature and are adopted by industry this approach will need to be reconsidered. We think it is essential that students understand the underlying principle of each step in the design process so as not to treat the tool chain (or the steps in using that tool chain) as a magic box that just makes CPS designs happen without careful engineering consideration. Some students will need to understand the details so they can become future tool builders.

Other approaches to a CPS project experience have been and no doubt will be devised with different strengths and weaknesses (for example as summarized in [5]), especially as CPS skills infuse the undergraduate curriculum so that more diverse students are ready to undertake projects of this depth and scope. Hopefully the description of this course and some of the lessons learned will help others down that path.

Acknowledgments

Developing this course has involved the support and efforts of dozens of teaching assistants, many of whom have made significant technical and organization contributions. They all have my deep and continuing appreciation for their support. Many of the contributors to this course have additionally been supported in their research by the General Motors Collaborative Research Lab at Carnegie Mellon University.

References

[1] 18-649 Distributed Embedded Systems, Carnegie Mellon University, http://www.ece.cmu.edu/~ece649

[2] ABET, Criteria for Accrediting Engineering Technology Programs (2012-2013), Oct 29, 2011, pp. 2-3.

[3] Bosch, CAN Specification version 2.0, 1991.

[4] Koopman, P., *Better Embedded System Software*, Drumnadrochit, 2010.

[5] Koopman, P., "Risk Areas In Embedded Software Industry Projects", Workshop on Embedded System Education, October, 2010.