From the Browser to the Remote Physical Lab: Programming Cyber-physical Systems

Steffen Peter, Farshad Momtaz and Tony Givargis
Center for Embedded and Cyber-physical Systems, University of California, Irvine, USA
Email: {st.peter, fdmomtaz, givargis}@uci.edu

Abstract—**Cyber Physical Systems (CPSs) integrate networked embedded computation systems with real-world physical installations. Programming of CPSs is not trivial, since CPSs combine traditional programming challenges and real-world timing, concurrency, and communication. This paper shows how a programming framework that allows students to implement and test CPS control programs in their Internet browsers, can improve both the students' learning experience and learning results. Students model and program a CPS application on a high abstraction level in a web page. This web page, provided by the instructor, invokes the student's code either together with the CPS as functional specification models in a virtual timing environment, or as component in a real-world system that interacts with a real remote physical implementation. Using the provided abstraction, students can incrementally design a CPS and experience challenges such as channel delays, model uncertainties, and real-time behavior, but without the need for complex low level programming or tools. For a CPS example system, we applied the framework in an embedded system design class. Our results show, the ability of a JavaScript-based programming and execution environment to design, program, and run CPSs on different levels of abstraction. Our results also indicate an increased approval from the students and a significantly improved understanding of modeling and programming in the class.**

I. Introduction

Cyber Physical Systems are systems that integrate a networked embedded computation subsystem (the cyber system) with physical subsystems from the mechanic, hydraulic, or electric domain. Examples are as simple as a garage door opener to complex systems such as medical robots, self-driving cars, or the electric grid. Therefore, building cyber-physical systems requires broad interdisciplinary knowledge, including appropriate physical models, precise timing, but also correct programming and implementation skills.

Due to their focus on specific domains, classic domain-specific curriculums do not support the required overview and abstraction. As one example, specific technical implementation details such as memory addresses and port numbers still dominate the embedded systems education, but are alienating for non-computer engineers.

In recent years, a range of new colloquiums and courses were proposed to address the development of cyber-physical systems [1], [2]. Most of the approaches rely on abstract simulations in environments like Ptolemy, Matlab, or in virtual labs that focus on the physical attributes of the system. While these approaches help to educate the concepts of modeling and physical integration, the abstraction is very high. Important concepts like timing, communication, programming, and event-handling are abstracted away so that an actual real-life implementation is not supported. The problem is not new. Already Vahid [3] described the trade-off between abstraction and implementation detail in embedded system education, and proposed a virtual computer system and a time-oriented programming model, which however only focuses on the embedded system properties.

In this paper we extend Vahid's idea to the programming of CPSs. We propose a framework that allows students to program the control system of a CPS in an abstracted but real-time oriented programming language. The programming and model abstraction is based on Process State Machines (PSMs), which model concurrent processes, communication and timing on a high abstraction level without the need to express low-level implementation details. The student programs are developed in the JavaScript programming language, which is available on most state-of-the-art internet devices. Therefore, students can practically control a virtual or real physical remote system from their laptop, tablet or even mobile phone. While the students' program is implemented and runs in the web browser, the physical system is executed on a remote location – either as virtual plant or on a real physical plant. The architecture facilitates, for instance, to run the program for a simulated virtual plant in a homework assignment, and execute the same program on a real physical system in class.

The benefits of our approach are:

1) the availability of a simplified in-browser programming language, which improves the accessibility of CPS programming,
2) the support of a experiment-driven systematic transition from PSM models, over abstract simulations and transaction-level models to an actual physical experiment,
3) the support for smaller CPS examples that can be addressed in student's homework but also in demonstrations and discussions in class.

In this paper we describe the methodology of our framework and outline the programming model (Section III). In section IV, we discuss the approach in detail for the educational cyber-physical system 'the Falling Ball' [4]. The results originating from a graduate system design class indicate the suitability of our approach to support the education of CPS design, both technically as well as pedagogically.

978-1-4799-8454-1/15/$31.00 ©2015 IEEE

II. Current Curriculums and Related Work

The education of CPS has drawn increasing attention in recent years and has been addressed in a range of dedicated workshops at major conferences. Examples are CPS-Ed - Workshop on Cyber-Physical Systems Education along the CPSWeek, and WESE - Workshop on Embedded and Cyber-Physical Systems Education along the ESWeek. The workshops produced a range of compelling approaches and example systems for educational purposes. For instance [5] proposed a focused programming language for dynamic systems. The application scenario is the control of a boat in a sea with currents. The example was discussed and evaluated in a simulation environment. While relatively simple and easily understandable, the example is not trivial and reveals a range of important CPS algorithm challenges, such as the impact of discretization of continuous systems and computation delays to the correctness of the system.

One educational system that can be built in practice is the coupled tank system as discussed by [6] and further applied by [7]. The idea of the system is to balance the levels in coupled tanks with a variable inflow and outflow. [6] proposed a web-based virtual laboratory for the system, but the system can be modeled and build in practice as well, with design kits such as [8].

Another popular teaching instrument for control CPSs – both in practice as well as in simulations – are inverted pendulum systems as for instance discussed by [9]. In those systems a pendulum has to be stabilized with appropriate control mechanisms calculated on a computation platform. The pendulum systems are well-suited for the education of cyber-physical control systems. However, the emphasis on control limits its application as a simple general CPS example.

Other CPS examples include capabilities of mobile phones into the CPS research. For instance [10] applied Android platforms to automatically classify human activities based on models developed in model-based simulation tools. An extended example that facilitates teaching by sensor-driven mobile applications was discussed by [11]. However, such mobile platform is already a complex system for which the integration in a course is not trivial. Other complex use cases include the robotic systems discussed in [12], the amphibious vehicles proposed by [13], or a search and rescue robot outlined in [14]. Such complex examples are exciting and motivating for students and researcher to investigate many interesting details in designing and building CPSs. However, due to their complexity such project usually forbid a complex design analysis, and in-depth discussion of design challenges. In addition, the integration of complex use cases in course frameworks is questionable in most cases.

Recent proposed CPS courses such as the flipped classroom course by [1] or the CPS undergraduate in-class courses introduced in [2] rely on a set of smaller design examples to demonstrate the variety of CPS design challenges. Due to its accessibility is our programming framework and the falling ball example, which is discussed in this paper, a promising contribution for easy integration in such on-class or flipped classroom setups.

From the technological perspective, online submission and assessment tools for programs written by students exist and have been reviewed in [15]. These tools, however, do not provide an interactive programming and experimentation environment and do not replace the installation of a dedicated development environment on the student's computer. Our approach does not aim at online submission but can be considered as a CPS run-time environment that can be programmed in JavaScript.

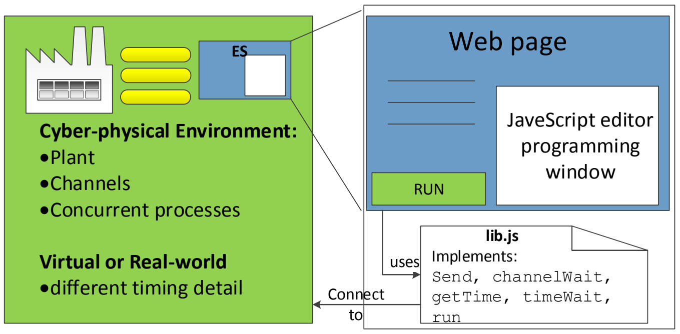


Fig. 1. Architecture of the programming framework: the instructor provides a web page with instructions and a programming window. Students write the application code in this programming window using abstract functions for communication and timing (in lb.js). The run button includes the students program in a cyber-physical run-time environment

JavaScript has already been proposed as language to educate basic programming concepts [16]. The proposed works focus on development for web pages or online games. Our work does not aim for web page programming. Instead we apply the capabilities of modern browsers to directly execute JavaScript code that is typed into the editor window of a web page, which is provided by the instructor of a class.

III. JavaScript-based programming of CPSs

In this section we discuss the technical background of the applied programming framework. We further outline the student's user experience as well as the required steps for the instructor to set up the educational programming environment.

As introduced, the primary goals of the framework are to provide

- a well-defined and easily understandable programming model,
- a clear abstraction from technical details such as timers or communication channel,
- and the flexibility for instructor to set up the CPS run-time environments ranging from timeless functional abstraction models to real-world experiments.

We realize these goals with our educational programming framework, whose architecture is illustrated in Fig. 1. When students load the instructions web page in their browser, they see a page with instructions, an editor window, and a run button. The editor window allows students to enter their JavaScript program. In this program students are encouraged to use a set of abstract platform-independent functions to access channels and the timing. These abstract high-level functions are provided by the instructor in a library file (`lib.js`), together with the code to access the run-time environment. Pressing the RUN button, the student's code will be included

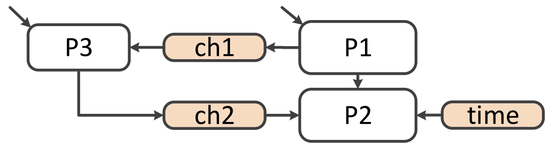


Fig. 2. Concurrent Process State Machine as basic model, containing process states (P1, P2, P3), state transitions (arrows between processes), and communication channels (ch1, ch2).

in the run-time environment and the system will be executed. Results of the run will be shown in a text box or graphically.

In the following subsections we discuss the suitability of JavaScript as a lightweight programming language, process state machines as the selected programming abstraction for the students, and the different kinds of cyber-physical environments that can be set up by the instructor.

A. Browser execution with JavaScript

As the underlying programming language we selected JavaScript for a range of reasons: JavaScript is a cross-platform scripting language and is supported by most operating systems and web browsers. Therefore, the execution of JavaScript does not require additional programs, tool chains or plug-ins on most platforms. While JavaScript code is usually embedded into the web page, with the `eval` and `globalEval` functions any use-provided code can be executed just in time. Even though JavaScript is a scripting language with an uncomplicated syntax structure, it is still very powerful, so that classes are available to build online connections or to draw images.

Another advantage of JavaScript is its popularity. According to latest statistics, JavaScript is the most popular programming language in the web [17]. From teaching perspective, the popularity has important benefits, as it results in existing knowledge and interest in the language. The broad availability of material and online support on online platforms like stackoverflow [17] is valuable for instructors and students as well.

The application of JavaScript requires to consider potential drawbacks, too. One disadvantage of the browser programming is the lack of debugging and syntax checking. The interpreter does not evaluate a line of code until this line actually is supposed to be executed, so that errors may stay unnoticed or result in unpredictable behavior. To cope with the syntax issues, we used the ACE editor [18] which provides syntax highlighting and live syntax checking. Another potential disadvantage is the low and non-deterministic performance of today's JavaScript interpreters. Since the intended size of the projects is rather small, we are convinced that the benefits such as good usability and easy understanding outweigh the listed drawbacks.

B. Programming Model and Abstraction

While the browser-based execution environment technically supports a wide range of structured and unstructured programming schemes, in our work we focus on a programming model that is based on the concept of process state machines (PSMs). The construction of a PSM model is widely considered one important first step in the design of embedded

```
if (state == 1) {
  t1 = getTime();
  send("ch1");
  state = 2;
}

if ((state == 2) && waitChannel("ch2")) {
  print("Round trip time="+(getTime()-t1));
}
```

Listing 1. Example JavaScript code for channel timing measurement (for PSM in Fig. 2).

systems [19]. PSMs allow a designer to model concurrent, communicating processes with a very concise semantic, consisting of processes (states), state transitions, and communication channels. As we will discuss later, PSMs are also suitable tools to model the behavior of the physical part and the interfaces of CPSs. As example, the PSM shown in Fig. 2 contains three processes: P1 and P2 are executed in sequential order, and P3 is a concurrent process. Between concurrent processes we allow communication and synchronization with abstract directed channels. In Fig. 2, P1 may send data to P2 via channel ch1, while P3 may send data to P2 via ch2. Due to the importance of timing for CPSs, we explicitly model timing as separate channel. Details such as the underlying platform or timing details of the channel are not considered. However, in the PSM abstraction, the implementation details of timing and channels are not important. As a result, students can use a small set of abstract function hubs to implement the PSM model. Important basic functions are:

- `waitChannel(ch)`: reads from channel ch,
- `send(ch, msg)`: writes optional message msg to channel ch. send without msg works as synchronization mechanism,
- `getTime()`: returns the time in ms,
- `waitTime(t)`: waits until time t.

These basic functions are part of the library (lib.js), provided by the instructor. Using these functions, students can implement the PSM in JavaScript independent from the underlying system.

As an example Alg. 1 shows the JavaScript code for the PSM in Fig. 2, as it can be entered in the editor box of the web page. Alg. 1 implements a round-trip time measurement between process P1-P2 and the concurrent process P3. Visible are the two states, as well as the interaction with the channel (`waitChannel` and `send`) and the time (`getTime`). Alg. 1 may be executed in a virtual environment or in the real world to measure the channel latency to a remote process. The environment is invariant to the students, but set up by the instructor within the instructions web page.

C. Setting up the Environment

One important idea of our architecture is that the application program (programmed by the students) is separated from the cyber-physical environment. Therefore students can use the same application program and apply it to cyber-physical environments on different levels of abstraction.

We consider two general run-time environments: the Virtual timing environment and the Real timing environment.

1) Virtual timing environments: A virtual timing environment controls the timing and progress of all system entities within the execution environment. Communication channels and the physical system are simulated. In our experiments, this virtual environment will be set up and executed as embedded JavaScript code in the browser. Both, the physical and the cyber environment are executed in the browser with the same virtual time.

In the virtual timing environment, several abstraction levels are supported. The first is the entirely functional specification model without any timing for computation and communication. More detailed transaction-level models separate communication and computation and facilitate the annotation of timings for each operation.

Benefits of the virtual timing environment are the easy and fast execution, the low organizational overhead, and the reproducible results. Since the timing is virtual, the experiments result in the very same result in every run on every device.

2) Real timing environments: The second run-time environment is the real-world, real-time environment. This environment does not simulate the physical part in the browser, but connects the student's program to an external physical system via real communication channels, such as the Internet. The remote physical system may be implemented either as remote virtual lab executed on a server computer, or as a real physical installation. The remote physical lab is an emulation of the real-world system executed on a powerful server. The remote lab can be used when repeated runs of an experiment by many users is not feasible for a real physical installation. The students program works as real-time control program in both cases.

For the Falling Ball example, which we discuss next, we realized both, the virtual plant (Section IV-A5) as well as the real system (IV-A6).

IV. In-class Experiment and Results

We applied the proposed framework in a graduate level course in the school of Electrical Engineering and Computer Science (EECS) at the University of California, Irvine. The basis for the course is the textbook Embedded System Design by Daniel Gajski et.al. [19], with extensions towards the design of CPS. The course has been taught in a regular classroom setup and included lectures and homeworks.

One of the CPSs that was modeled, programmed, and analyzed in the class, is the Falling Ball example (FBE) [4], for which we first outline the setup and the implemented models, and then discuss the results and the implications of the experiments.

A. The Falling Ball Experiments

1) Use Case: The falling ball example: The architecture of the Falling Ball example is illustrated in Fig. 3 (A). The goal of the system is to take a picture when a falling ball passes a camera mounted on a pole. To determine timings, the system has two motion sensors. The ball is dropped from a height initially unknown to the system, while the height of the sensors and the camera are known. This, in fact simplistic, example has some interesting properties of CPSs that refer to the main challenges of CPSs.

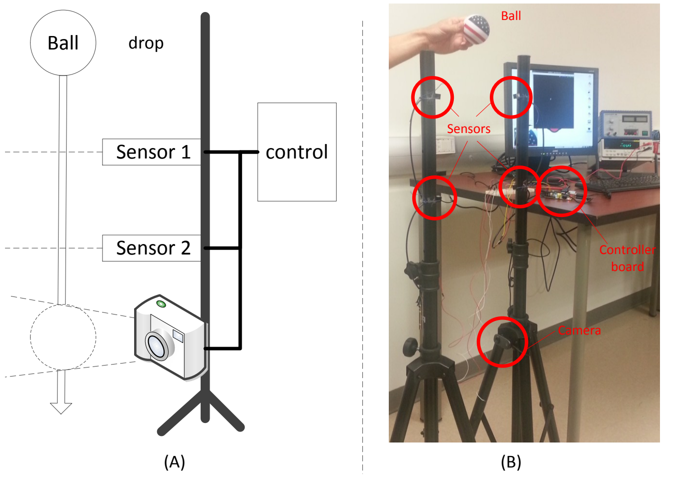


Fig. 3. Setup of the Falling Ball example: (A) as schematics, (B) in practice.

The FBE needs exact timing. The timing relates to the physical process, which runs completely independent of any cyber processing. Even though the example can be described in a physical process, which is well understood by the developer and can be expressed in well-known mathematical equations, it is obvious that we will not achieve perfect precision – a fact that interferes with the needed accuracy requirements. The system combines various sensors and the actuator. So their cooperation affects the expected outcome of the system and needs to be regarded as an important factor.

One beneficial attribute of the Falling Ball example is the possibility to implement the system with little manual effort. A practical implementation of the FBE is shown in Fig. 3(B). The system was realized with on a Raspberry Pi Model B Revision 2.0 with Debian Linux. We further used two Honeywell Through Beam Infrared sensors (HOA6299 Series), which work by detecting an interrupt of the line of light from the emitter to the detector. The actuator is the Raspberry Pi Module camera. Two Pyle-Pro Tripod Speaker Stand poles mount sensors and camera.

We used this setting for local tests (written in ANSI C), but also as part of the course to connect the student's JavaScript programs to the real implementation (see Subsection IV-A6). Before the real-world test concluded the series of experiments, we asked the students to model and test the system on different levels of system detail.

2) PSM modeling: As first development step, before the control program can be expressed the system has to be modeled as Process State Machine (PSM). In the class, this modeling step was given as homework assignment and was later discussed in class. Fig. 4 shows one correct PSM for the FBE with concurrent processes in the physical and cyber part as well as the interfaces. It can be noticed that the system consists of five concurrent processes: the physical system, the two sensors, the actuator and the embedded system process. In the PSM we can already see the boundary of the embedded system, which eventually should be implemented by the students, and

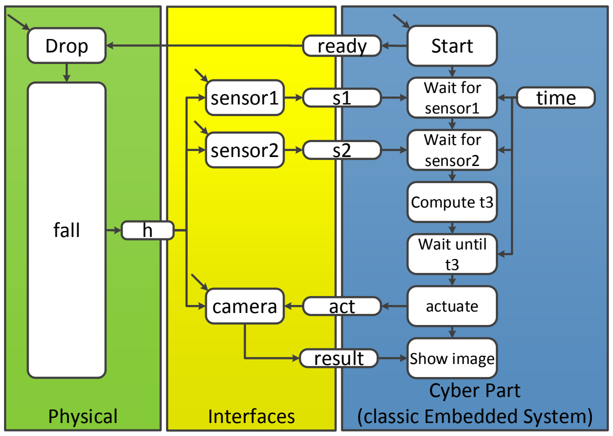


Fig. 4. Process state machine for the falling ball example: the embedded cyber system state machine (blue) interacts with the physical system via sensors and the camera actuator. Communication between the processes is modeled by directed channels.

the environment, which is provided on the web page by the instructor. The embedded system in this case consists of six states, which are implemented next.

3) Functional Model in the virtual timing environment: The functional specification model allows the students to implement and test the FBE with ideal (i.e. no) delays and timing uncertainties in computation and communication. The functional models of channel and physical model are executed in a virtual timing environment as invariant JavaScript code in the student's browser. The students had to write the control program, which is an implementation of the 'Cyber Part' state machine of Fig. 4.

In the class, students implemented this program together with an instructor in the editor window of the loaded web page. Alg. 2 shows one complete functional implementation of the control part of the PSM. It can be clearly seen how the program reflects the states, state transition, as well as timing and channel operations of the PSM. The only complex operation is the computation of the expected time. The equation was provided by the instructor. Running the code results in a range of print outputs and a generated graphic of the ball in front of the virtual camera (see Fig. 5).

The result of this first executable simulation is that, first, we successfully translated a PSM into actual code as part of a simulated CPS, and second, that after few milliseconds the ball is always caught slightly below the center of the camera. These observations trigger recommended discussions about the functional and timely determinism of the result (why is it always the same on all devices), about the cause why the ball is caught slightly off center (continuous vs. sampling time), and the difference between real time and simulation time.

4) Transaction-Level Model in the virtual timing: The transaction-level model (TLM) extends the functional model with a decoupled channel model. Thus, timing properties of computation and communication can be expressed separately, and can be modeled in more detail. Therefore, in the instruction web page students have the opportunity to parametrize the channels. The channel model uses the same syntax and semantic as the computation model, so that, as example, the

```
if ((state==0) && waitChannel("S1")) {
  t1 = getTime();
  state = 1;
}

if ((state==1) && waitChannel("S2")) {
  t2 = getTime();
  state = 2;
}

if (state==2) {
  t_diff = 0.001*(t2-t1);
  velocity = (h2-h1)/(t_diff) + (g*(t_diff)/2);
  exp = (-V+Math.sqrt((V*V)+(2*g*(h3-h2))))/g*1000;
  state = 3;
}

if ((state==3) && (wait_until(s2+exp))) {
  send("act");
  state = 4;
}

if ((state==4) && (result=waitChannel("result"))) {
  print("result ="+ result);
}
```

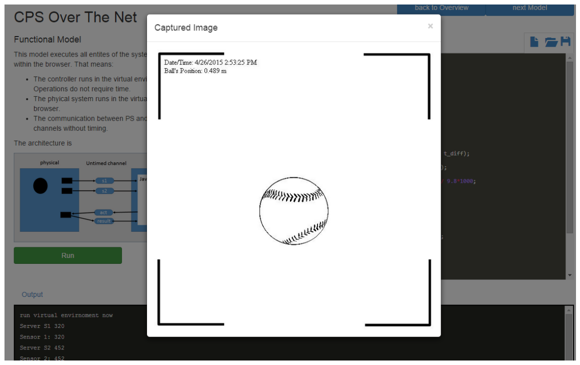
Listing 2. JavaScript implementation of the FBE (see PSM Fig. 4).

S2 channel with a 2ms delay can be expressed as

```
if (waitChannel("S2.in")) {
    waitTime(2);
    send("S2.out");
}
```

Each computation and communication operation can be annotated with a time this operation needs, by adding the `waitTime` statement. This allows students to express a timed model of the system and experience the effect to the outcome of the simulation and need adaptations of the control program. Students have to understand the source and quantity of delays and anticipate the delays in the control program. Advanced design options include the dynamic measurement of the round trip time of the channel (see Alg. 1).

5) Remote Virtual Plant Real-world model: The virtual plant model is a model of the falling ball executed on a remote server. While the actual plant model is still idealistic (i.e. it does not consider air resistance or material), we see two major differences to the previous TLM example: First, real timing, and second, the use a real Internet communication channel.



This model executes all entites of the syste
within the browser. That means:

- The controller runs in the virtual envi
  Operations do not require time.
- The phyical system runs in the virtua
  browser.
- The communication between PS and
  channels without timing.

The architecture is

Date/Time: 4/26/2015 2:53:25 PM
Ball's Position: 0.489 m

```
run virtual environment now
Server S1 320
Sensor 1: 320
Server S2 452
Sensor 2: 452
```

Fig. 5. Result after a successful run in the functional model: The simulated ball is caught by the camera.

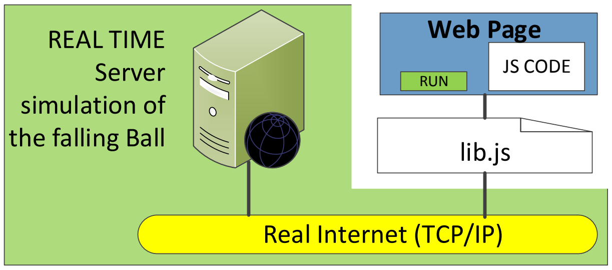
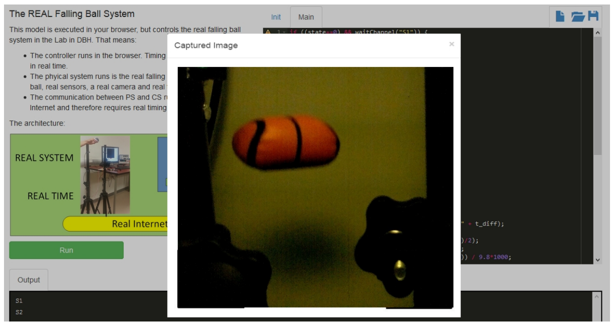


Fig. 6. Architecture of the FBE experiment with a virtual plant: the lib.js connects the student's code to a server computer, that emulates the falling ball in real time, via the Internet.



The REAL Falling Ball System

This model is executed in your browser, but controls the real falling ball system in the Lab in DBH. That means:

- The controller runs in the browser. Timing in real time.
- The phyical system runs is the real falling ball, real sensors, a real camera and real
- The communication between PS and CS r Internet and therefore requires real timing

The architecture:

```
if ((state==0) && waitChannel("S1")) {
" + t_diff);
)/2);
;
)) / 9.8*1000;
```

```
S1
S2
```

Fig. 7. Resulting picture of the falling ball. The ball appears crushed due to the slow vertical sampling (bottom to top) in the camera sensor.

The server was implemented on an Intel Xenon machine running ASP.NET. After the server receives the signal from the 'drop' channel (compare Fig. 4), the simulator waits for a random time before simulating a drop from a random height. We included the randomness to avoid deterministic behavior that could be exploited. In the physical simulation, the server program updates the velocity and position of the ball as fast as possible (tens of microseconds resolution) in real time. Sensor signal s1 and s2 are sent to the controller, which has to process the signals and send the actuation signal at the right time under consideration of the channel delays. After receiving the 'actuate' signal, the server generates a picture of the ball at the simulated position.

While the complexity of the simulation is substantially increased, the student's control program remains unchanged. All platform-dependent code was added in the `lib.js`, and remains invariant to the students. The actual control program still expresses the small PSM, only with added timing anticipation.

We conducted this experiment in class to control the virtual plant, which runs on a server in another building on campus. Since we use an actual Internet connection, the experiment outcome is less deterministic. In more than half of the cases, the ball could not be caught by the camera. The result is not surprising and motivated discussion on the suitability and the need for detailed modeling of communication channels for reliable CPSs.

6) Real-world system: In the final experiment of this series, we replaced the virtual plant with the Raspberry Pi that accesses the sensors and the camera. In the classroom students were able to execute their JavaScript code, expressed in the web page, to control the physical system in the lab, few buildings away from the class room. This experiment required assistance in the lab. We needed to actively drop the ball. The practical part was recorded on video and projected in the class. Due to the nature of the experiment only one student at the time could run the experiment. The outcome was very similar to the virtual plant setup, since in many runs the ball could not be caught. In a successful case, however, the student received a photo of the actual ball, as shown in Fig. 7.

B. Results and Discussions of the Experiments

Admittedly, the organizational and building efforts for the final experiment are considerable. This observation increases the importance of the virtual plant setup, which already includes all the timing and programming properties the students are supposed to experience. The virtual remote physical setup contains real timing of control and physical system, communication over the network, and timing uncertainties in computation and communication – all at student's hand in the Internet browser, without additional tools.

However, the practical setup adds additional benefits: First, the real experiment is a special event in usually less eventful programming and design class. In addition, the experiment gave practical closure to the design flow that started with a PSM model and which then developed to a real-world cyber-physical system that can be controlled in real time by a mobile device in the classroom. In this development process, each design step is incremental and seems very little, and therefore, is be easy to understand. We studied this hypothesis in a short sentiment study. In the survey 82% of the students agreed that they understand all design steps, while the biggest challenge was the development of the PSM. Even though the real-timing experiments failed in many cases due to the unreliable communication properties, in this survey 86% of the students agreed that this particular misbehavior helped them understand the importance of good modeling and the purpose of the design flow. On a broader question, 96% of students agreed that the falling ball example helped to understand modeling or programming of embedded systems in general.

In addition to the student sentiment, we also analyzed the learning results. We compared the results of weekly graded quizzes to the results of earlier classes on the same subject. Compared to earlier classes without the new programming framework, the amount of correct answers related to model abstraction and the CPS programming improved from 80.7% to 95.3%. One particular result is an improved understanding of abstraction and the impact of channel uncertainties. As comparison, the average percentage of correct answers between the two groups for all other questions increased from 83.0% to 87.5%. The results indicate that the modeling and timing abstraction, which was a relative weakness in earlier years, now is a relative strength of the students.

One additional observation from this class was the increased number of practical student's projects that investigated the effect of communication timing and uncertainties. For instance students applied functional modeling and TLM-based time sensitivity analysis for an electric circuit breaker in the smart grid, the control of a quadcopter drone, and the control of the security and safety system in a smart building.

V. Conclusions

This paper discussed a browser-integrated development approach for the education of modeling and programming of Cyber-Physical Systems. Following our experiments in an Embedded System Design class class, we can draw three conclusions:

First, we got confirmation that teaching techniques that include active programming and experiment-driven analysis are well received by students. The idea to control a real physical system from a mobile phone, a tablet or the laptop in the classroom is appealing to students and instructor. As a result, not only the student's approval improved, but also their results in quizzes on modeling-related questions improved from 80 to 95%.

The second conclusion addresses JavaScript as a suitable programming language for the instructor as well as for the students. JavaScript is a programming language with low organizational and programming overhead, as it does not need the setup of error-prone and platform-dependent tool chains. Applying JavaScript, the proposed framework runs the program, which student enter in a web page, as part of a virtual or real CPS.

Finally we can conclude that the education of CPS programming and design is feasible with smaller examples and just within an Internet browser. On different levels of abstraction and detail, students could run their program, first, as part of a CPS simulation with entirely virtual timing, then, in real-time using a virtual remote physical lab, as part of a real physical setup. The technical interfaces to the environments in all cases are provided by the instructor and hidden from the students, who can concentrate on the modeling, programming and running the experiments.

Based on these results we consider the proposed approach as promising contribution to programming exercises for embedded and cyber-physical systems in traditional class setups, but also to online classes, for which we consider an implementation in near future.

Acknowledgment

This work was supported in part by the National Science Foundation under NSF grant number 1136146.

References

[1] P. Marwedel and M. Engel, "Flipped classroom teaching for a cyber-physical system course-an adequate presence-based learning approach in the internet age," in *Microelectronics Education (EWME), 10th European Workshop on*, 2014.

[2] A. M. Cheng, "An undergraduate cyber-physical systems course," in *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, 2014.

[3] F. Vahid and T. Givargis, "Timing is everything–embedded systems demand early teaching of structured time-oriented programming," *Proceedings of WESE*, pp. 1–9, 2008.

[4] S. Peter, F. Vahid, D. D. Gajski, and T. Givargis, "A ball goes to school - our experiences from a cps design experiment," in *First NSF Workshop on CPS Education (at CPSWeek 2013)*, 2013.

[5] K. Bauer and K. Schneider, "Teaching cyber-physical systems: A programming approach," in *Workshop on Embedded and Cyber- Physical Systems Education (WESE)*, 2012.

[6] C. C. Ko, B. M. Chen, J. Chen, Y. Zhuang, and K. Chen Tan, "Development of a web-based laboratory for control experiments on a coupled tank apparatus," *Education, IEEE Transactions on*, vol. 44, no. 1, pp. 76–86, 2001.

[7] P. Derler, E. Lee, and A. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[8] QUANSER, *Coupled Tanks*, 2014, http://www.quanser.com/products/coupled_tanks.

[9] M. Demirtas, Y. Altun, and A. Istanbullu, "Virtual laboratory for sliding mode and pid control of rotary inverted pendulum," *Computer Applications in Engineering Education*, vol. 21, no. 3, pp. 400–409, 2013.

[10] K. Damevski, B. Altayeb, H. Chen, and D. Walter, "Teaching cyber-physical systems to computer scientists via modeling and verification," in *Proceeding of the 44th ACM technical symposium on Computer science education*, 2013, pp. 567–572.

[11] H. Chen and K. Damevski, "A teaching model for development of sensor-driven mobile applications," in *Proceedings of the 2014 conference on Innovation & technology in computer science education*. ACM, 2014, pp. 147–152.

[12] K. Huang, H. Shah, K. Savant, D. Chen, G. Chen, S. Klose, and A. Knoll, "A lego/fpga-based platform for the education of cyber-physical/embedded systems," in *Workshop on Embedded and Cyber-Physical Systems Education (WESE)*, 2013.

[13] T. Withrow, M. R. Myers, T. Bapty, and S. Neema, "Cyber-physical vehicle modeling, design, and development," in *ASME 2013 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2013.

[14] O. Lawlor, M. Moss, S. Kibler, C. Carson, S. Bond, and S. Bogosyan, "Search-and rescue robots for integrated research and education in cyber-physical systems," in *e-Learning in Industrial Electronics (ICELIE), 2013 7th IEEE International Conference on*, 2013.

[15] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. ACM, 2010, pp. 86–93.

[16] Q. H. Mahmoud, W. Dobosiewicz, and D. Swayne, "Redesigning introductory computer programming with html, javascript, and java," in *ACM SIGCSE Bulletin*, vol. 36, no. 1, 2004, pp. 120–124.

[17] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[18] Ace, *Ace - The High Performance Code Editor for the Web*, 2015, http://ace.c9.io/.

[19] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*. Springer Science & Business Media, 2009.