A Ball Goes to School - Our Experiences from a CPS Design Experiment

Steffen Peter
Ctr. for Embedded Computer Systems
University of California, Irvine
Email: st.peter@uci.edu

Frank Vahid
Dept. of Computer Science and Eng.
University of California, Riverside
Email: vahid@cs.ucr.edu

Daniel D. Gajski, Tony Givargis
Ctr. for Embedded Computer Systems
University of California, Irvine
Email: {gajski, givargis}@uci.edu

***Abstract*—Teaching the methodologies of Cyber Physical System (CPS) design requires good examples that are easy to understand and tools that are commonly used in practice. This paper presents our experiences during the practical execution of model-driven design processes applying a range of state-of-the-art design tools for a novel and simple example from the CPS domain. The Falling Ball example has several properties that support teaching basic design principles of CPSs. On one end, students use a number of modeling tools to design and simulate the Falling Ball example. On the other end, students actually build the Falling Ball example using a variety of approaches. Our methodology teaches not only the tools and how to use them to design a CPS system but imparted general concepts such as the need for modeling and the presence of certain technical problems and challenges. This paper presents the example, the applied tools and experiences gained during the first test run of this example in our research group. We plan to use the material presented here in an introduction to CPS course to be offered at UCI later in 2013. In this work we share our experiences with the larger educational community.**

I. Introduction

Most generally, in a Cyber Physical System (CPS), an integrated computation subsystem (cyber system) interacts with a physical subsystem. The design of such a CPS usually requires good understanding of both subsystems and their interactions. For instance, properties of the cyber system, such as timing, may influence the outcome of the physical process. conversely, the correctness of the cyber part cannot be established unless an adequate model of the physical subsystem is well understood and included in the design process.

The CPS groups at UCI and UCR have been working on a project in the area of CPS since 2012 and are funded by the NSF. A main objective in these groups is the development of an educational program to teach design of CPS to students. Typically students of computer science have obtained good knowledge on implementing computation systems, they, however, likely fail to catch the interactions with the physical subsystem. Therefore, as part of this educational program it is imperative that students execute actual examples including a close interaction between the cyber and the physical subsystem. While it may be tempting to teach one single large example using the most-popular development environment we are of the opinion that such an approach does not reflect the variety of CPS applications, tools and methodologies. Instead we propose teaching material that:

- cover a range of simpler examples that are easy to understand, design and evaluate using a variety of existing development tools and methodologies,
- allow students to understand the importance of models and learn their limitations (e.g. precision),
- allow students to gain hands-on experience with state-of-the-art simulation and modeling tools,
- allow students to experience typical design challenges and sharpen their attention in crafting solutions.

Several educational example applications exist ranging from inverted pendulums, connected tanks of fluids, and engine or boat control [1], to name a few. These examples typically belong to the domain of control systems, for which the control algorithms may be complex, are not easily understood and often distract from the actual design of the system. Instead, we propose as one rather simple example for a CPS –the Falling Ball– to be designed by the students. The Falling Ball is about taking a picture at exactly the moment a falling ball passes a camera, while the time is predicted based on information from motion sensors mounted above the camera.

Our groups, consisting of graduate and undergraduate students with different educational backgrounds and skill sets, experimented with the Falling Ball in the Fall quarter 2012. We implemented the example in practice and in a range of model-based design and simulation tools recommended in related work. This paper illustrates the example, the work that was done, the tools that were used and the experiences we gained. The paper is structured as follows: After this introduction we explain the Falling Ball and its design challenges. Then we outline the tools we used for simulation and briefly describe experiences gained during this design process. A short summary and outlook concludes the paper.

II. The Falling Ball example

In this section we introduce the Falling Ball example and illustrate a typical design process for this kind of CPS. We first introduce the use case, then describe the practical setup and then outline a model-driven design process.

A. Setup

The example is illustrated in Figure 1(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 will be 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: First it needs exact timing.

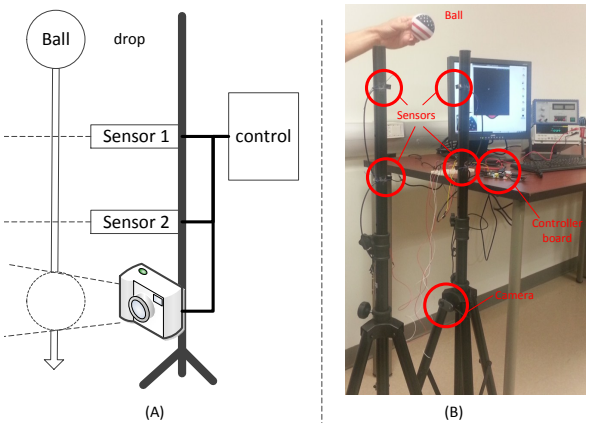

Fig. 1. Setup of the Falling Ball example: (A) as schematics, (B) in practice.

This is one example where faster is not necessarily better. Second, the example can be described in a physical process that is well understood by the developer, for which the control system needs a mathematical understanding, and for which it is obvious that we will not achieve perfect precision. Additionally the example can be implemented with little manual effort.

B. Practical demonstration setup

We implemented this system in practice, first to demonstrate its practicability and second to experience problems. The practical setup is shown in Figure 1(B). The system was realized with the following components:

- Raspberry Pi (RP) Model B Revision 2.0 with Debian Linux and standard C development environment,
- Two of Honeywell Through Beam Infrared sensors (HOA6299 Series), which work by detecting an interrupt of the line of light from the emitter to the detector,
- Logitech HD Webcam C310 attached via USB,
- Two Pyle-Pro Tripod Speaker Stand poles to mount sensors and camera.

The sensors were mounted on two poles and connected to the GPIO port of the RP board. The camera was mounted below the sensors and connected to the USB port of the RP. The control program was written and compiled in standard C on the Debian Linux environment. The setup and programming was done by one undergrad student.

The initial version of the control program did not account for the delays in sensors, actuators, and computation. Through a trial and error process the control program was revised with more and more timing details until it worked. This method worked for this rather simple example. However, obviously, such a trial-and-error implementation strategy is unlikely to work efficiently for complex scenarios. A suggested way to address this issue is model-based design, which is briefly discussed in the following sections.

C. Model-based Design Flow

Model-based design (MBD) flows, as for instance described in [2], commonly apply an abstract model of the system

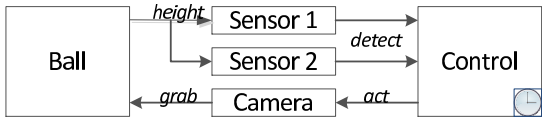

Fig. 2. System entities and high level data flow.

consisting of the physical subsystem (PS), the Cyber Subsystem (CS), and the interfaces, in order to simulate the interaction between the subsystems. This executable simulation of the system is the starting point for the actual implementation which is a successive refinement of the model into an implementable system description. The presence of the simulation allows the developer to test the refined models in context of the full system at any time. We illustrate the four steps to establish the executable simulation for the Falling Ball example next.

1) Define the architecture of the system: The first step is the description of the interfaces and the data flow between the physical components. The block diagram for the Falling Ball example is shown as Figure 2. It consists of the Ball entity, which contains the PS, the Control entity, which corresponds to the CS, and the two sensors and the camera component. The sensors receive information of the height of the ball, and indicate to the controller when the Ball is in sight. The controller receives the events, records the times of these events and triggers the camera after the computed timing. The camera receives the trigger command from the controller and forwards the activity to the physical system where it is decided if the ball is caught.

2) Description of the PS: The second step includes the description of the physical systems conveniently expressed as differential equations. In case of the Falling Ball, the height of the ball is determined by the differential equation of velocity over time, while velocity is determined by the differential equation of acceleration over time. This small system of equations is initialized with a constant acceleration (gravity=$9.81 m/s^2$) and a selected initial height, i.e. the drop height of the ball.

3) Design of the control program: As the control program usually cannot be expressed with differential equations it is necessary to express the physical system as part of the control with regular expressions. In case of the falling ball this step is relatively easy and we can apply the basic free fall equations. Using the known height of both sensors (h_1 and h_2) and the time the sensors triggered (t_1 and t_2), we can compute the speed of the ball at sensor 2:

$$v_2 = \frac{h_1 - h_2}{t_2 - t_1} + a\frac{t_2 - t_1}{2}.$$

We can use this v_2 to compute the expected time (t_3) at the height (h_3) with the following equation:

$$t_3 = \frac{at_2 - v_2 + \sqrt{{v_2}^2 + 2\,ah_2 - 2\,ah_3}}{a}.$$

Computing this equation is the most complex step in the small control program:

1) wait for sensor 1: after receiving signal from sensor 1, record the time,

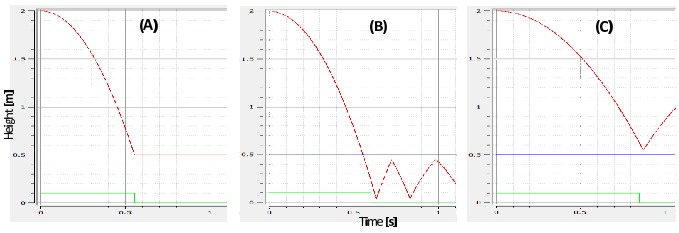

Fig. 3. Simulation results of the ball control model. Red is the simulated height of the ball, blue is the height of the grabber and green indicates the grabber signal. (A) is the ideal control, (B) shows the grabber being too late, and (C) too early.

2) wait for sensor 2: after receiving signal from sensor 2, record the time,
3) compute expected time for actuator,
4) wait until t_3, then activate camera while compensating for the expected delay time in the system.

4) Simulation, and test of the control algorithm: At this point of the development flow we have the global architecture and behavioral descriptions for each subsystem. With an appropriate Model of Computation (MoC) it is possible to simulate the behavior of the entire system. This allows the developer to test the algorithms in first place and further to test the behavior if parts of the systems deviate from the perfect model.

As an example Figure 3 shows three execution runs as recorded in the Modelica environment: Figure 3(A) shows the ideal result when the ball is captured exactly in the moment the ball passes the camera. Figure 3(B) assumes a small delay in the camera causing the ball to pass the camera Figure 3(C) assumes a smaller effective gravity constant in the PS - taking into account air resistance, which results in the camera triggering too early.

III. Model-driven design tools

In this section we describe four well-known tools (Modelica, Simulink, Ptolemy, SystemC) to design and simulate systems, and outline experiences we had in our group designing the ball example with these tools. We chose these tools as they are frequently recommended for model-based development of CPSs. The implementation work was executed by graduate students of computer science in their first to the third year of study. At the start of the project they had no specific skills in CPS design or modeling tools. The properties (timing, behavior) of the system components for this experiment are based on their specification and were given by us at the start of the project.

1) Modelica: Modelica [3] is a professional modeling language initially from the physical domain. It has been used to model hydraulic, mechanic, and electrical systems among others. Therefore, contrary to most design tools computer engineers typically apply, Modelica centers around the physical part of the CPS. However, for the cyber parts also algorithmic control programs can be added. Because Modellica resembles the well understood object-oriented paradigm, it was straightforward to design the four components (physical, control, sensor, actuator) and connect the ports of the entities according to the schematic seen in Figure 2. The implementation strategy for the simulation corresponds mostly to the ideal design

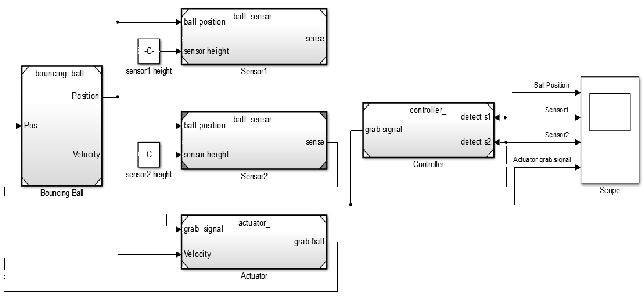

Fig. 4. CPS system in Simulink. On the left the model of the physical system (bouncing ball), in the middle the interfaces (sensors and actuator) and on the right the control program.

flow introduced in the previous section. The physical part can be described with differential equations, the cyber-part is a sequence program instruction. Figure 3 shows some results from the simulations.

Problems during this development were related to errors in the application of the programming language, which at least for our system often led to crashes instead of constructive error messages. A second problem that occurred during the simulation is related to the well known effect of Zeno-behavior. In one case the simulation stopped progressing and seemed to crash. This was caused by the simulation trying to calculate the bouncing of the ball with infinite precision. This phenomenon is an important problem and experiencing it helped the students understand and develop solutions. We could either enforce a higher time granularity in the simulation or avoid the Zeno behavior in the model of the PS by adding a condition to stop bouncing if the altitude is too low.

2) Simulink: The Matlab/Simulink [4] package is often considered as the industrial standard for simulation of control systems. The program is not free, but it is available at most universities. Simlink provides the designer with a set of basic building blocks which can be parametrized, instantiated in the simulation environment and then connected via available ports of the blocks. Figure 4 shows the top level block diagram of the simulated system.

Using Simulink, the biggest problems we encountered were related to the design of the control part, as it seemed necessary to assemble building blocks. It needed a significant amount of creativity and time to understand and assemble available building blocks to achieve the intended goals. Another problem was that automatic settings in the simulation environment led to unpredictable and wrong simulation results, so that as solution it was necessary to set a constant time rate. In general, however, the results were similar to the outcome from Modelica.

3) Ptolemy: In contrast to the first two tools, Ptolemy [5] comes from an academic background. While the design experience, i.e. composing building blocks in a graphical environment, is similar to Simulink, Ptolemy emphasizes the model of computation in each block of the system. Beside this explicit addressing of the MoC, our design experience was very similar to Simulink. Within Ptolemy, the biggest challenge was to find the correct building blocks and find a way to assemble them into a system.

4) SystemC: SystemC is a set of C++ libraries and tools originating from the domain of electronic systems design to model and simulate hardware on a higher abstraction level. Today it is an industry-approved toolset for model-driven development of complete systems. Contrary to Simulink and Ptolemy, SystemC is particularity well suited for modeling the cyber part of the ball example. Due to its origins, SystemC makes it easy to model the CS including details with regard to system and processor architectures. In contrast the opportunities to describe the PS are rather limited. In fact, the physical model had to be explicitly described similar to traditional test benches with physical equations computing velocity, acceleration, and position for each time step of the simulation.[1] The resulting implementation consists of two classes: ball –simulating the physical subsystem– and controller, which describes the behavior of the cyber system. Both subsystems were simulated with a frequency of 1 kHz. The textual output of the simulation shows internal timing information and if the ball could be caught by the camera.

A. Results

Modeling the Falling Ball with each of the four tools described in the previous sections could be finished by the graduate students within 3 to 4 weeks, spending 10 hours per week for practical work on average. This does not include the time for studying related literature and research on discovered phenomena. Each student started without any prior knowledge of the development environment. The relatively short time needed for the project has been to some degree caused by the low complexity of the chosen example but also by the rich resources of tutorials and examples for each evaluated platform available in the Internet. We encouraged to look for a bouncing ball example as starting point which is available for most CPS-related platforms. As this ramp-up phase for tools was relatively low, one major challenge for the students was to realize a good structure and system architecture in the tools. Without adequate initial planning the implemented systems became messy resulting in an unpractical tight entanglement of physical and computational components. We had to provide the high level architecture as introduced in Figure 2 to separate PS and CS as well as the interfaces. Otherwise the outcome is hardly comparable and is not suitable as basis for other design challenges. Other errors were related to the development of the equations in the control part, mostly caused by wrong application of Newton laws equations. Some students used a wrong starting point, so that the simulation was set up in a way that the drop starts at a known position (usually 0) and the distances are relative to this drop position. Other challenges such as Zeno behavior, selection of the appropriate MoC and the selection of the correct simulation resolution were already mentioned in the sections above. To solve these questions and to discuss the progress we had weekly meetings. Discussing the problems triggered more general questions related to the design of CPS such as: If the equations already are complicated to solve correctly in this small example, how can we design more complex systems? If Modelica is good for physical simulation and SystemC is good for the computation, can we combine both of them? How and why does simulation frequency affect the simulation outcome, and how can we be sure that we still learn something from varying results?

From the practical side we learned about workings of the tools and their strengths and limitations. Modelica seems to be best suited to simulate physical systems, while SystemC seems to be optimal for the design of the computational system. Simulink is a powerful tool suite with capabilities we could not entirely comprehend in this small project. Ptolemy is valuable with regard to teaching MoCs which are important in CPS design with their continuous and discrete behavior.

As a final aspect, the availability of the practical set up of the Falling Ball has additionally contributed to the learning experience, as it first, delivered actual parameters from the real world to the simulations, and second, we can try and refine the practical program applying results from the simulations.

[1]Recent extensions [6] allow to simulate physical systems described with differential equations but we have not applied this package in our project

IV. Conclusions

The experiences we described in this paper underline the importance of practical experiments with actual tools in the domain of CPS design. We showed that a model-driven design process using state-of-the-art design tools is possible in a relatively short time span. The lessons we learned include knowledge how to use the tools, how to perform the architectural and planing phase, how to develop and design computations and how to execute the simulations and make use of the results. We could execute all these design steps for the Falling Ball example, which has several challenging properties to support teaching basic design principles of CPSs. The simple ball example has been valuable as it allowed the students to do each step on their own - including the development of the physical subsystem and the development of the control program. Playing with the system, we could experience how small changes, in particular in the delay of the sensors, have significant impact on the systems correctness. The lessons we learned in this experiment likely help instructors to teach CPS design to students and we plan to use the material presented as parts for a full course to be offered at UCI later in 2013.

Acknowledgment

We thank Ting-Shuo Chou, Volkan Gunes, Ina Liu, and Abhinav Parvathareddy for their contributions to this experiment. This work was supported in part by the National Science Foundation under NSF grant numbers 1016789 and 1136146.

References

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

[2] J. Jensen, D. Chang, and E. Lee, "A model-based design methodology for cyber-physical systems," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE, 2011, pp. 1666–1671.

[3] *OPENMODELICA - Homepage*, 2013, http://www.openmodelica.org.

[4] *Simulink - Simulation and Model-Based Design*, 2013, http://www.mathworks.com/products/simulink/.

[5] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity-the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[6] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich, "An introduction to modeling embedded analog/mixed-signal systems using systemc ams extensions," Tech. Rep., 2008.