Virtual Prototyping through Co-simulation of a Cartesian Plotter

M.A. Groothuis, A.S. Damstra, J.F. Broenink

Control Engineering - University of Twente - The Netherlands

P.O.Box 217, 7500 AE Enschede, The Netherlands

{m.a.groothuis, j.f.broenink} @utwente.nl, a.s.damstra@alumnus.utwente.nl

Abstract

This paper shows a model-based design trajectory for the development of real-time embedded control software using virtual prototyping. As a test case, a Cartesian plotter is designed. Functional correctness of the plotter software has been ensured by means of co-simulation using a virtual prototype before deploying it on target. Except for the interface implementation, the software that is used in the co-simulation is identical to the software that is compiled to run on the target computing platform. Virtual prototyping is especially important if the real target can damage itself if it is operated outside its safe operation zone or when prototypes are not yet available for testing. The co-simulation of the software against a virtual prototype resulted in a first-time-right deployment on the real target.

1. Introduction

The components for a mechatronic system are made by engineers from different disciplines, each having their own way of working and tooling. It is hard to test the complete mechatronics design during development; often only the individual parts or components can be tested separately. When the entire system is finally integrated and tested, unforeseen problems arise. It is often at the boundaries of the disciplines that mistakes are made: incorrect gear ratios, mistaken polarity of a motor or software connections mapped by hand. Besides this, time-to-market constraints require a more and more concurrent design flow. Therefore, a need exists for new design methodologies that address these challenges. The Cartesian plotter used in this paper as test case is built to demonstrate and evaluate a model-based multiple-view approach in mechatronic system design [6, 2]. Concurrent engineering of disciplines and cooperation across the discipline boundaries to prevent integration problems are important features here. Especially when a mechanical setup is part of the system, the influences of dynamics of the mechanical system on the software behavior cannot be neglected and should be taken into account during the software design and testing. Furthermore, care needs to be taken in safety design to prevent damage. Especially, when the move-

Figure 1. Photo of the Cartesian plotter

ments are limited, like at, for instance, XY positioning systems (i.e. plotters) or robots. The embedded control software should be functionally correct before testing on target. In this paper we use a virtual prototype in combination with co-simulation to test the software across the boundaries of its discipline.

The design method [6, 2] used for designing the embedded control software of this test case is as follows:

1. Partition the system into top-level components. Make a suitable split-up to allow a concurrent design flow as much as possible.
2. Physical System Modeling, i.e. model the plant parts of the embedded control system (dynamic behavior of the mechanical system).
3. Control Law(s) Design, using the models obtained in the previous step. Model reduction is often necessary to get a model of adequate detail. Use the models of step 2 to verify the control laws.
4. Embedded Control System Implementation: the control software is designed via refinement of the control laws.
5. Realization of the embedded software via an ongoing refinement process. Models of the components are replaced by the real system parts in a stepwise manner (both for hardware, plant, and software).

This paper, focuses on the embedded control software development, although the relation to the other steps and other involved disciplines are taken into account. Within each design step, the design work is done iteratively, whereby each intermediate result is verified via (co-) simulation. Hence, details are filled in in a stepwise refinement manner. The goal is to get the embedded control

software running first time right on the embedded target and to prevent unforeseen integration issues.

Since several simulations tools are used, co-simulation is a suitable way of instrumenting the need for early integration testing. Although the idea of co-simulation is certainly not new, most existing implementations use dedicated point-to-point co-simulators or custom developed co-simulation links, making co-simulation a time-consuming activity and an inflexible testing solution. Our goal is to make it more user-friendly, less time-consuming and show the benefits, especially for testing the embedded control software. Therefore the CosiMate [3] backplane co-simulation tool is used to get a more flexible co-simulation solution. CosiMate allows us to interconnect many simulators and languages, such as ModelSim (VHDL), C, SystemC, Simulink, StateMate and Saber. At our Lab two tools are used which are not supported: 20-Sim and gCSP. 20-Sim [4] is a tool for modeling and simulating the behavior of dynamic systems, such as electrical, mechanical and hydraulic systems. gCSP [8] is a graphical tool for creating and editing Communication Sequential Processes (CSP) diagrams and is used in our Lab to generate the software framework for embedded control systems. Generic support for these tools is implemented by writing new interfaces to CosiMate.

Section 2 gives a brief overview of the related work on co-simulation. The rest of this paper focuses on the analysis (section 3) and design and realization (section 4) of the embedded control software for the plotter using virtual prototyping. Section 5 presents the conclusions and recommendations for future work.

2. Related work

In mechatronic system design often a collection of unlinked modeling & simulation tools are used. When the system is implemented, informal techniques are used that involve human-language interactions between groups that do not necessarily have full understanding of each others domains. This uncertainty can result in errors that are difficult to identify and debug [1].

The area of embedded control system design shows efforts to reduce these problems found during integration. The design of an automotive system for instance, requires the co-design of hardware, software, and micro-mechanical components [10]. In traditional design approaches the different parts are designed by different groups and the integration of the overall system is made at the final stage. Here, interfacing problems and resource problems may introduce extra time and extra cost. For optimal use of computing resources however, the control algorithm and the control software design need to be considered at the same time [13]. This is often not practiced due to separation of their design trajectories.

A different approach is found in the Ptolemy II [11] tool, where instead of linking the unlinked tools, one tool offers a heterogeneous simulation framework that can

Figure 2. Functional overview plotter

be used to model many different domains in one model. Ptolemy is a major step forward in integrated model-based design but currently it uses graphical modeling symbols, which are too much deviated from commonly used symbols.

3. Analysis

The cartesian plotter used for this virtual prototyping test case is a three degrees-of-freedom manipulator with a flat A3 drawing surface and a H-shaped axis configuration. The X and Y axis have a belt transmission (figure 1) and are driven by DC motors equipped with encoders for position feedback. DC motors are used in this setup to make it more interesting from a control engineering point of view. The pen is connected to a third axis (Z-axis) mounted on top of the Y-axis, which is, on its turn, mounted on top of the X-axis. Six end-switches are mounted at the edges of the axes in such a way that they are activated and switch off the motor power (hardware safety) when the plotter head moves outside the safe working area. They can also be used as input for the software (software safety and for homing & calibration).

Figure 2 shows a functional overview of the plotter. To plot an image, we need an image file created by a CAD program. This image file needs to be translated into setpoints for controller. The controller uses the setpoints and position feedback from the plotter to steer the motors and draw the image.

Two phases are needed to plot an image: image creation and image to motion processing. HPGL [7] is chosen as image format for plotter files. Many CAD programs can export vector drawings into this format. An HPGL example that draws a square, a circle and an arc is shown in figure 3 (one unit is $25\mu m$).

Figure 3. HPGL commands & image

To draw an HPGL image, the HPGL commands should be translated into motion sequences. A setpoint generator should create a smooth and time-optimal pen manipulator trajectory from these motions. The result is a list of time-x-y-z setpoints used as input for the controller. In order to be implemented in real-time, the setpoint generator

Figure 4. The 20-Sim model

should have a low computational complexity and it should take into account the limited working area and limit the speed, acceleration and jerk (time derivative of the acceleration) to reduce wear and improve tracking accuracy and speed [12]. Safe limits for this setup are $j_{max} = 5.0m/s^3$, $a_{max} = 1.0m/s^2$, and $v_{max} = 0.5m/s$.

4. Design & Realization

For the design of the plotter, we have followed the procedure indicated in section 1. The plotter is designed by multiple people working in parallel. The mechanics, electronics, controllers and the rest of the software are designed in parallel. First a physical-system model containing the dynamic behavior of the plotter is created. This model is used for the mechanics design and for the loop controller design. During the realization of the mechanics, this model will be verified against the setup. For the design of the Embedded Control System (ECS) software, we use our graphical CSP tool, gCSP. At the start of the software design, the real setup is not yet available. The approach is to first develop a fully functional virtual prototype which is verified by means of co-simulation. Using this virtual prototype the software can be implemented and tested via co-simulation against the 20-Sim physical-system model to make sure it is functionally correct. When this test is successful, the system software will be deployed and tested on the real plotter.

4.1. Virtual Prototype

The physical-system model and the controller are both modeled in 20-sim (figure 4). The model of the plotter is modeled in such a way that the interfaces correspond to the actual I/O interfaces of the plotter. Bond graphs [9] and Ideal Physical Models (IPM) are used to model the dynamic behavior of the plotter setup (see figure 5 for the X-axis model). To test the controller, simple motions (e.g. sine waves for a circle) are applied as setpoints. Concurrent to the controller design, an HPGL-to-setpoint generator is designed in Matlab.

The ECS structure is modeled in gCSP (figure 6). gCSP is used to 1) automatic C++ code generation for the ECS software and 2) to automatically generate CSPm code. The CSPm code is used to formally check, using FDR2 [5], and correct the software structure for possible deadlocks and livelocks.

Figure 6. The gCSP model

Figure 7. Co-simulation connection diagram

Figure 8. X axis 20-Sim simulation result

Next, the HPGL setpoint generator is converted to a C++ class and code is generated from the 20-sim *Controller* submodel. These code pieces (part A in figure 4) are included in the gCSP model to complete the software design. This software is verified first via a co-simulation with the 20-Sim plotter model (part B in figure 4). The co-simulation connection diagram is shown in figure 7. The gCSP generated ECS software and the 20-sim plotter model are both connected to the Cosimate co-simulation bus. The ECS software runs at $1kHz$ (controller frequency) and the plant model in 20-Sim runs at $10kHz$ to obtain enough accuracy, since the time constants of the plant model demand this 10 kHz in combination with an Euler integration method. Cosimate takes care of the time synchronization, interpolation and data exchange. Using this structure it is possible to co-simulate the software model and the 20-Sim plotter model. The actual movement of the plotter is co-simulated in real-time using a normal plots (figure 8) and a 3D animation of the plotter.

4.2. Target Implementation & Results

The transition from the virtual prototype to the target implementation is done by changing one `#define` in the software to switch the interfaces (CSP channels) from co-simulation to target I/O. Our gCSP C++ CSP support library is designed such that we have a 100% identical API for Windows, Linux and RTAI Linux. The change from virtual prototype to target is not more than replacing the external CSP channels (the circles in figure 6) and re-compiling the sources for our RTAI linux system.

The co-simulation is used only for functional software

Figure 5. The plotter (plant) model, X-axis

testing. Because no simulation of target processing platform (e.g. CPU speed and memory) was included we cannot guarantee from the co-simulation alone that the software can reach its real-time requirements on target. For this purpose we have executed a processor-in-the-loop simulation to measure the performance on the real target computing platform where the I/O is still redirected to a development machine.

Finally the software is executed on the real plotter setup showing a correct working plotter setup. This is a first time right implementation. Several test plots (see also figure 9) have been made, ranging in duration from 30 seconds to over one hour. Safety layers were successfully tested and the velocity is within the specified bounds.

Figure 9. Actual plot results

5. Conclusions & Recommendations

Virtual prototyping proves to be useful for the development of the mechatronic system software. Knowing the software is correct by formally checking its structure and using co-simulation to verify its functional behavior is especially necessary if the real target can damage itself if it is operated outside its safe operation zone. In the case that a validated model (used for the mechanics and controller design) is already available, one can re-use this model for co-simulation testing at low extra costs. In a parallel design trajectory the virtual prototyping allows the software designer to design and test his embedded control software earlier and even before a real prototype is available. This resulted in our test case that the software was finished and deployed on target first time right when the real mechanic setup arrived.

Co-simulation proved to be a useful means for early integration testing. To make it more user-friendly and straightforward to use, we are investigating automation possibilities (interface generation & coupling) by using

a SysML system level model of the mechatronic system. Furthermore we will more explicitly incorporate virtual prototyping in our design method and optimize the used tools for this.

References

- [1] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *Computer*, 36(4):45–52, April 2003.
- [2] J. F. Broenink, M. A. Groothuis, P. M. Visser, and B. Orlic. A model-driven approach to embedded control system implementation. In *2007 Western Multiconference on Computer Simulation*. SCS, San Diego, CA, 2007.
- [3] Chiatek. Cosimate, 2008. www.chiatek.com.
- [4] CLP. 20-sim, 2008. <http://www.20sim.com>.
- [5] Formal Systems. FDR2, 2008. www.fsel.com.
- [6] M. A. Groothuis and J. F. Broenink. Multi-view methodology for the design of embedded mechatronic control systems. In *Proc. IEEE Int'l Symposium on Computer Aided Control Systems Conference, CACSD 2006*, pages 416–421. IEEE, Munich, 2006.
- [7] Hewlett-Packard. *The HP GL/2 and HP RTL reference guide: a handbook for program developers*. Addison-Wesley Professional, 3 edition, 1997.
- [8] D. S. Jovanovic, B. Orlic, G. K. Liet, and J. F. Broenink. gcsdp: A graphical tool for designing csp systems. In I. East, J. Martin, P. H. Welch, D. Duce, and M. Green, editors, *Communicating Process Architectures 2004*, pages 233–251. IOS press, Oxford, UK, 2004.
- [9] D. C. Karnopp, D. L. Margolis, and R. C. Rosenberg. *System Dynamics: Modeling and Simulation of Mechatronic Systems*. Wiley-Interscience, 3rd edition edition, 2000.
- [10] P. Le Marrec, C. Valderrama, F. Hessel, A. Jerraya, M. Attia, and O. Cayrol. Hardware, software and mechanical cosimulation for automotive applications. In *9th IEEE International Workshop on Rapid System Prototyping, 1998*, June 3-5, 1998.
- [11] J. Liu. *Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems*. PhD thesis, University of California, 2001.
- [12] S. Macfarlane and E. Croft. Design of jerk bounded trajectories for on-line industrial robot applications. In *Robotics and Automation*, May 21-26, 2001.
- [13] M. Torngren, D. Henriksson, K.-E. Arzen, A. Cervin, and Z. Hanzalek. Tool supporting the co-design of control systems and their real-time implementation: Current status and future directions. In *IEEE International Symposium on Computer-Aided Control Systems Design, 2006*, October 4-6, 2006.