On Model-driven Design of Robot Software using Co-simulation

Jan F. Broenink, Yunyun Ni, and Marcel A. Groothuis

Control Engineering, Faculty EEMCS, University of Twente, Enschede, The Netherlands,
{j.f.broenink, y.ni, m.a.groothuis}@utwente.nl

Abstract. In this paper we show that using co-simulation for robot software design will be more efficient than without co-simulation. We will show an example of the plotter how the co-simulation is helping with the design process. We believe that a collaborative methodology based on model-driven design will improve the chances of closing the design loop early, improving cross-discipline design dialog, and reduce errors, saving cost and time.

Keywords: co-simulation, model-driven design, robot software

1 Introduction

The development of robot software is a demanding discipline. Technical challenges arise from the need to develop complex, software-intensive products that take the constraints of the physical world into account. Commercial pressure includes the need to innovate rapidly in a highly competitive market and to offer products that are simultaneously resilient to faults and highly efficient. Traditional development approaches are mono-disciplinary in style, in that separate mechanical, electronic and software engineering groups handle distinct aspects of product development and often do so in sequence. Contemporary concurrent engineering strategies try to improve the time-to-market by performing these activities in parallel. However, system-level requirements (or so-called cross-cutting concerns) that cannot be assigned to a single discipline, such as performance and dependability, can cause great problems, because their mono-disciplinary impact is exposed late in the development process, usually during system integration. Robot (software) development, in which the viability of the product depends on the close coupling between the physics and computing disciplines, therefore, calls for a more multi-disciplinary approach.

So, besides models of the embedded control software, also models of the dynamic behavior of the robot mechanism are used for design and verification purposes. These are two different types of models, each with their own way of computation, namely Continuous Time (CT) for the robot mechanism dynamics and control algorithms and Discrete Event (DE) for the decision control and logic implemented in the embedded software. To compute (simulate) such a combined

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

model, co-simulation of both parts is used, where the value of the time variable on both sides is synchronized (Fig. 1). We use a layered structure for designing the embedded control software, supporting separation of design activities (Fig. 2). This gives focus to each design step, but also allows different design steps to be conducted simultaneously. A model-driven approach is effective here, as models can easier be adapted than the code generated from these models.

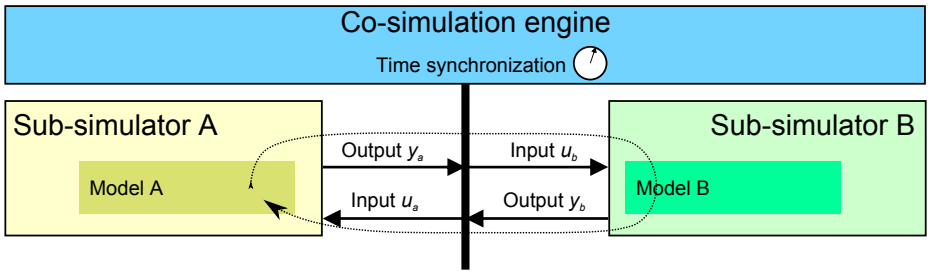

Fig. 1. Co-simulation scheme

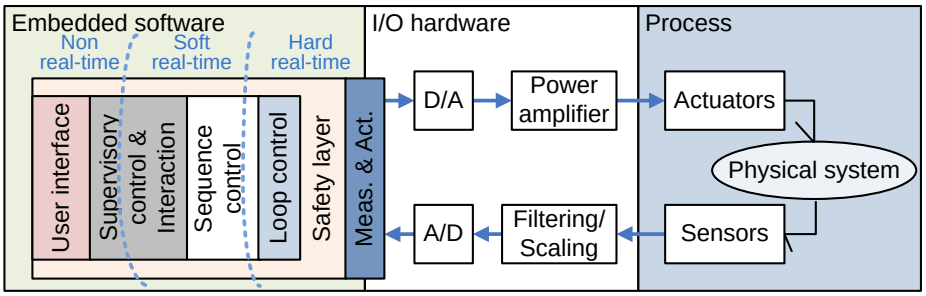

Fig. 2. Embedded control system and its software architecture

Section 2 presents some background knowledge on the formalisms, techniques, and tools used. In Sect. 3, we present the general approach that we use, and in Sect. 4, we present an example in which we use co-simulation to do early integration tests. Section 5 concludes the this paper and mentions the project that we currently are working on.

2 Background

A robot consists of a combination of a mechanical (physical) system, mixed-signal and power electronics, and an embedded (motion) control system (ECS),

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

as shown in Fig. 2. The combination of a mechanical setup and its ECS software requires a multi-disciplinary and synergistic approach for its design, because the dynamic behavior of the mechanics influences the behavior of the software and vice versa (also known as cyber physical systems). Therefore, we adhere a mechatronic or systems approach for the co-design of the physical system and its software, to find an optimal and dependable realization.

For describing the software structure and logic (i.e. DE part), we have our own graphical CSP tool, gCSP [12] which is based on the GML language (graphical notation for CSP) [11]. gCSP diagrams contain information about compositional relationships (SEQ, PAR, PRI-PAR, ALT and PRI-ALT) and communication relationships (waiting rendezvous channels). The tool supports animation/simulation [18] of these diagrams and code generation of CSPm code (for deadlock and livelock checking with FDR2 [6] or ProBE [7]), Occam code, C++ code (using the CTC++ library [16]) and Handel-C code [14].

For modeling the dynamic behavior of the robot mechanism (i.e. the CT part), we use bond graphs [17, 1, 13], which are a domain-independent graphical description of dynamic behaviour of physical systems. This means that systems from different domains (cf. electrical, mechanical, hydraulic, acoustical, thermodynamic, material) are described in the same way. The basis is that bond graphs are based on energy and energy exchange. Analogies between domains are more than just equations being analogous: the used physical concepts are analogous. Bond-graph modelling is a powerful tool for modelling engineering systems, especially when different physical domains are involved. Furthermore, bond-graph submodels can be re-used elegantly, because bond-graph models are non-causal. The submodels can be seen as objects; bond-graph modelling is a form of object-oriented physical systems modelling. See for further reading the text book of Karnopp and Rosenberg [13] and the short introduction by Broenink [2].

During the process of designing robot software, the interaction between controller models and models of the robot-mechanism dynamics is studied using simulation (more precise: co-simulation). Controller models are generally expressed in discrete event (DE) formalisms, while models of the robot mechanism are generally expressed in continuous time (CT) formalisms. It is best to allow these models to remain in their natural formalism. Interaction will be achieved by executing the models simultaneously and allowing information to be shared between them. This is so called co-simulation, which does simulation of heterogonous models.

3 Approach

3.1 Co-simulation

The approach that we use for designing the embedded control system (ECS) software for robotic systems is based on model-driven design with a close cooperation between the involved disciplines, using co-simulation.

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

The DE simulator and CT simulator are coupled together using a co-simulation engine which synchronizes the simulation time in both simulators, based on the work of Nicolescu et al. [15]. Figure 3 shows a schematic overview of the synchonization between the simulators following the numbered order. The CT simulator updates first the states for the next time instance, followed by the DE simulator. This prevents the need for back stepping at the DE size. Time events occur when the CT solver has reached some time instance t_k. These events can be seen as scheduled/expected CT events. Besides the normal time events, we can also get state events between two time instances (q_{se}). State events occur when the solution of a differential equation crosses some boundary value p, e.g. a zero-crossing event. In that case, the DE simulator gets an early update to reach s_{se}. The CT simulator then continue from q_{se} and reaches q^*_{k+1} as the next state instead of scheduled CT event q_{k+1}. The scheme in Fig. 3b is used in our case study (see the next section).

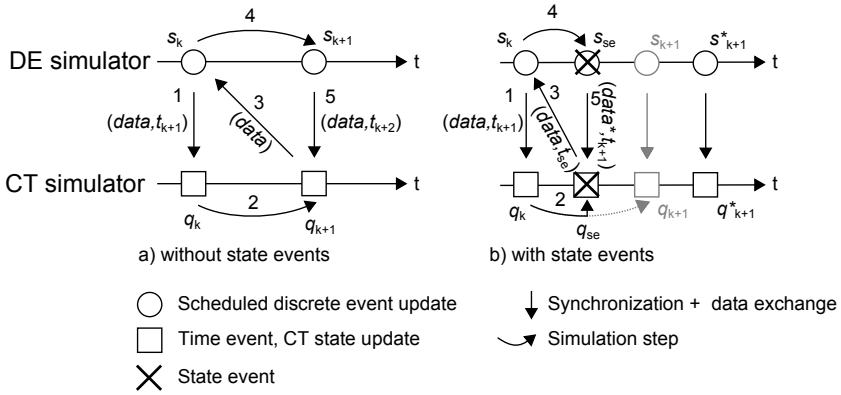

Fig. 3. Synchronization scheme for DE-CT co-simulation (inspired by [8])

3.2 Co-design

In general, when designing robotic software, we follow the design pyramid shown in Fig. 4. One starts from an abstract top-level model that is extended via stepwise refinement and design space exploration into a complete ECS software design.

Stepwise Refinement is the gradually adding more detail to the models (which are quite coarse in first instance) towards such a detail that the embedded control software can be generated from these detailed models. *Design Space Exploration* is trying out several alternative solutions, whereby all solutions together span the design space. In these co-design and stepwise refinement processes, co-simulation can help a lot, as with co-simulating the combined DE and CT model, some in-

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

sight on the appropriateness of the alternative being simulated, can be obtained, especially on cross-model type concerns.

During the route from idea to final realization, many design decisions need to be made which all have their own influence on the final result. Every decision restricts the design space and starts a new smaller design pyramid, as shown in Fig. 4. The reachable solutions (feasible design space), whether optimal or not, depend on all these decisions.

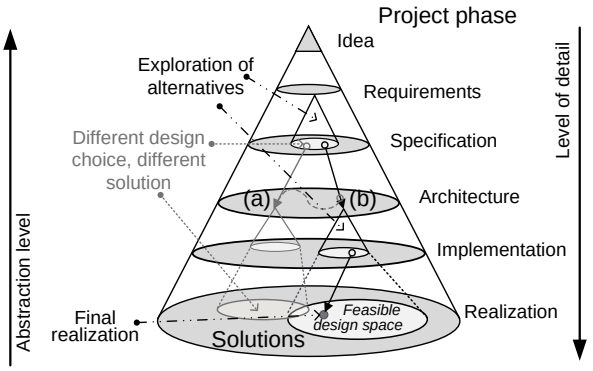

Fig. 4. Design Pyramid with different abstraction levels

4 Example: Design of a Cartesian Plotter

This section describes a case study which used co-simulation to do early integration testing. In this case study, a cartesian plotter setup was designed and built [4, 9]. The focus in this section is on the design of the embedded software for this mechatronic system. Although the chosen setup is not strictly a robotic system, it has many features in common with robotic systems like the need for safety layers to protect the mechanics and the environment, path planning (the drawing) and concurrent control of multiple joints.

The main design goal for this case study was to *concurrently* design and build the mechanical system, the electronics and the embedded software. The specific goal for the software was that it should be ready when the physical setup is also finished. This means that no possibilities existed to test the software on the real setup before the final integration of all components. The ultimate goal for the software was a first time right software realization in order to prevent a long integration phase due to late software modifications. In order to achieve this goal, we had to find and solve potential integration problems in earlier design phases.

Figure 5 shows a schematic overview of the applied concurrent design workflow. The design of the plotter software (5b) was started at the same time as

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

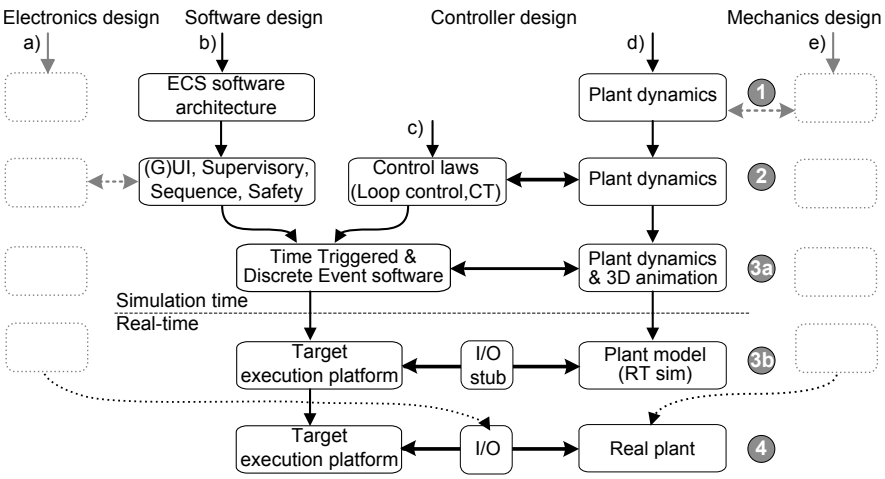

Fig. 5. Concurrent design workflow

the design of the electronics (5a) and the mechanics (5e) following an iterative development flow. The final software meets the the real plotter setup for the first time during the final integration (step 4 in Fig. 5).

The software of the plotter was designed using graphical block diagrams in 20-sim (Fig. 6a; the loop controller layer in Fig. 2) and gCSP (Fig. 6b; discrete event part; the left layers and the safety layer in Fig. 2). Besides unit testing, formal verification and co-simulation were used to ensure fault free software. Formal verification of the gCSP software framework using the FDR2 [6] tool was used to test the concurrent software against deadlocks. Co-simulation was used to do a more thorough test of the software across the boundaries of the software engineering discipline.

The mechanics part of the set up consists of a structure and a behavioral part: the CAD drawings and the dynamic behavior (plant part (B) of Fig. 6a) of the plotter. The dynamic system model of the plotter (Fig. 5d) is used initially for the design space exploration of the plotter mechanics (step 1, d and e in Fig. 5) and later on for the control algorithm design (step 2, c and d in Fig. 5).

This is where commonly the usage of the dynamic system model stops. For this test case, the dynamic system model was re-used to built a virtual prototype of the plotter. This virtual prototype was a co-simulation between the generated software from the model in Fig. 6b and the model of the plant dynamics (plant part (B) of 6a). Figure 7 shows the virtual prototype and the two modeling tools in a co-simulation experiment.

This virtual prototyping approach allowed running more extensive tests on the software than with regular unit tests. It allowed us to test the influence of the setup dynamics (time-dependent behavior) on the software and vice versa. Furthermore, we could safely test the safety measures without damage to real setup and see the result of our HPGL-drawing to setpoint generator as a drawing

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

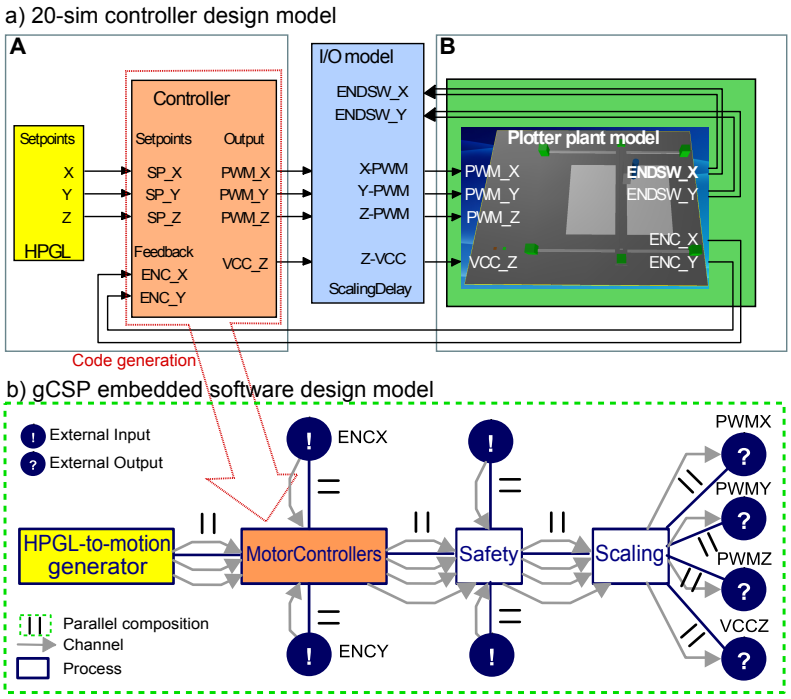

Fig. 6. a) 20-sim controller design model, b) gCSP ECS design model

instead of a log-file with setpoints. The co-simulation iterations (3a in Fig. 5) revealed several remaining errors in the software after unit testing and formal verification:

- Scaling: a gear ratio mismatch between the software and the Solidworks CAD drawing of the mechanics was found. The gear ratio is needed in the software to convert position values from rotation encoders into translation movement of the plotter pen;
- Sensors: sign mismatch between movement direction and sensor values;
- Safety layer: hitting an endstop means that the plotter reaches the end of the drawing area. The safety layer should disallow the movement towards the end but allow moving backward. The safety layer had the correct behavior but at the wrong side of the drawing area (the allowed direction was wrong);

These errors, of which several could cause severe damage to the mechanical setup, were all solved before the real on-target test. Although many errors could be found during this co-simulation step, we could not guarantee from the co-simulation alone that the software could reach its real-time requirements on target. Because no simulation of target processing platform (e.g. CPU speed and memory) was included in the co-simulation experiment, we have executed a processor-in-the-loop simulation (3b in Fig. 5) to measure the performance on the

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

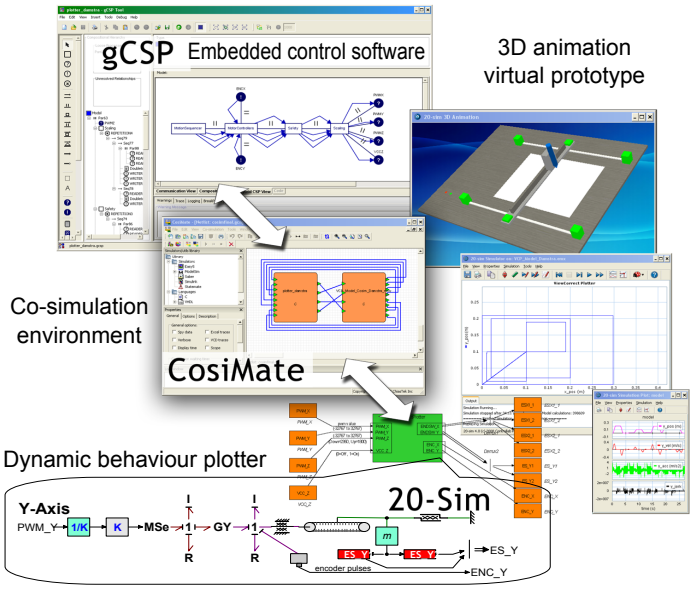

Fig. 7. Software co-simulation experiment against a virtual prototype

real target computing platform (an embedded PC/104 with RTAI Linux). The I/O was still redirected to a development machine running the plant simulation. The result for the plotter test case is that the software was finished and deployed on target first time right when the real mechanical setup arrived. No remaining errors were found during the final tests on the real setup. Figure 8 shows the final result.

Virtual prototyping proves to be useful for the development of robotics and mechantronics software. Knowing that the software is correct by formal checking its structure and using co-simulation to verify its functional behavior is especially necessary if the real target can damage itself when operating outside its safe operation zone. In case that a (preferably validated) plant 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. Of course, the plant model has to be updated when late structural changes are done at the mechanics side. However, this needs to be done anyhow to check the consequences of these changes on the control algorithm. In a concurrent design trajectory, the virtual prototyping allows the software designer to design and test the software earlier and more extensively even before a real prototype is available.

5 Conclusions & Ongoing work

The test case presented in the previous section supports our claim that a collaborative methodology based on model-driven co-design improves the cross-

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668

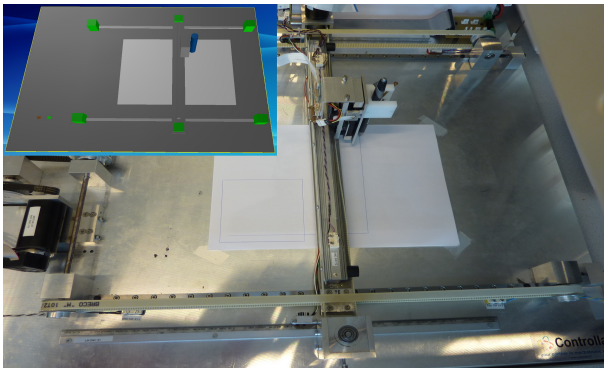

Fig. 8. Software controlling the real setup

disciplinary design dialogue, reduces integration errors, and thus saves costs and time. The claim that using models in the design of embedded control software, has already been shown in the BODERC project [10], which was a predecessor of the research projects involved in this work. The aim of our current European Community's Seventh Framework Programme DESTECS project [3, 5] is to research and develop methods and open tools that support the collaborative design of dependable real-time embedded control systems. In this project, we use fault-injection techniques to further test through co-simulation the system under study and thus enhance the quality of the resulting embedded cotnrol software. In DESTECS, we are coupling 20-Sim and Overture (a tool for VDM++(Vienna Development Method), for the DE part) in order to implement co-simulation.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248134.

This research has been carried out partly as part of the STW/PROGRESS ViewCorrect project, supported by the Dutch Ministry of Economic Affairs under the PROGRESS embedded system research program of the Dutch organization for Scientific Research, NWO, and the Technology Foundation STW.

References

1. Breedveld, P.: Multibond-graph elements in physical systems theory. Journal of the Franklin Institute 319(1/2), 1–36 (1985)
2. Broenink, J.F.: Introduction to physical systems modeling with bond graphs. Tech. rep., University of Twente (1999), http://www.ce.utwente.nl/bnk/papers/BondGraphsV2.pdf

3. Broenink, J., Larsen, P., Verhoef, M., kleijn, C., Jovanovic, D., Wouters, F., Pierce, K.: Design support and tooling for dependable embedded control systems. In: Proceedings of SERENE '10. pp. 77 – 82. ACM, ACM Sigsoft (Apr 2010)
4. Damstra, A.: Virtual prototyping through co-simulation in hardware/software and mechatronics co-design. Msc thesis, Control Laboratory, University of Twente (Apr 2008)
5. DESTECS: Design support and tooling for embedded control software. Website (2010), http://www.destecs.org
6. Formal Systems (Europe) Ltd: FDR2 Manual (Jun 2005), http://www.fsel.com/documentation/fdr2/fdr2manual.pdf
7. Formal Systems (Europe) Ltd: Formal systems europe website (2010), http://www.fsel.com
8. Gheorghe, L.: Continuous/Discrete Co-simulation interfaces from formalization to implementation. PhD thesis, University of Montreal, Canada (Aug 2009)
9. Groothuis, M., Damstra, A., Broenink, J.: Virtual prototyping through co-simulation of a cartesian plotter. In: Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on. pp. 697–700. No. 08HT8968C, IEEE Industrial Electronics Society (Sep 2008)
10. Heemels, M., Muller, G.: Boderc: Model-based design of high-tech systems. Embedded Systems Institute, Eindhoven, The Netherlands (2006)
11. Hilderink, G.: Managing complexity of control software through concurrency. PhD thesis, University of Twente, Enschede, The Netherlands (May 2005), http://doc.utwente.nl/50746/1/thesis_Hilderink.pdf
12. Jovanovic, D.: Designing dependable process-oriented software, a CSP approach. PhD thesis, University of Twente, Enschede, The Netherlands (2006)
13. Karnopp, D.C., Margolis, D.L., Rosenberg, R.C.: System Dynamics: modeling and simulation of mechatronic systems. Wiley (2006)
14. Mentor Graphics: Dk design suite & handel-c. Website (Oct 2009), http://www.mentor.com, mentor is since January 2009 owner of Handel-C and the DK Design Suite
15. Nicolescu, G., Boucheneb, H., Gheorghe, L., Bouchhima, F.: Methodology for efficient design of continuous / discrete-events co-simulation tools. In: Anderson, J., Huntsinger, R. (eds.) Proceedings of the 2007 Western Multiconference on Computer Simulation WMC 2007, San Diego. SCS, SCS, San Diego, San Diego (Jan 2007)
16. Orlic, B., Broenink, J.F.: Redesign of the C++ Communicating Threads library for embedded control systems. In: Karelse, F. (ed.) 5th PROGRESS Symposium on Embedded Systems, pp. 141–156. STW, Nieuwegein, NL (2004)
17. Paynter, H.: Analysis and design of engineering systems. MIT Press, Cambridge, MA (1961)
18. van der Steen, T.: Design of animation and debug facilities for gCSP. MSc thesis, Control Engineering, University of Twente (June 2008), http://purl.org/utwente/e58120

Proceedings of SIMPAR 2010 Workshops
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Darmstadt (Germany) November 15-16, 2010
ISBN 978-3-00-032863-3
pp. 659-668