Collaborative Modelling and Co-Simulation with DESTECS: A Pilot Study

Ken Pierce, Carl Gamble
School of Computing Science,
Newcastle University, Newcastle upon Tyne,
United Kingdom, NE1 7RU
Email: kenneth.pierce@ncl.ac.uk,
carl.gamble@ncl.ac.uk

Yunyun Ni, Jan F. Broenink
Control Engineering,
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente, Enschede, Netherlands
Email: y.ni@utwente.nl,
j.f.broenink@utwente.nl

***Abstract*—This paper describes a collaborative modelling exercise using the DESTECS framework. The DESTECS approach allows engineers and software designers to collaborate to produce system models that contain a discrete-event (DE) model of a controller and continuous-time (CT) model of a plant. We call these models *co-models* and call their execution *co-simulation*. The DESTECS tool couples existing DE and CT tools (Overture and 20-sim, respectively) allowing engineers to use paradigms and tools with which they are familiar, while collaborating to construct these shared system models. The work involved collaborative modelling of a line-following robot. We report on both the details of the models and experience in producing them.**

***Keywords*-Collaborative modelling, co-simulation, embedded control, fault tolerance, evolution, DESTECS, VDM, 20-sim.**

I. Introduction

The creation of dependable embedded control systems is challenging. Building these systems requires collaboration across disciplines, with engineers and software designers working together to produce systems with hardware and software elements. Increasing market pressures make successful collaboration more important, and the need to consider faults and fault tolerance increases system complexity.

One approach to dealing with this complexity is to build models in formally defined languages and run simulations in order to validate designs at an early stage, without resorting to expensive prototypes. Modelling of embedded control systems is restricted however by their cross-disciplinary nature: each discipline has a different culture, abstractions and formalisms. Often continuous-time (CT) formalisms are used by engineers in the design of physical systems, with which it can be difficult to describe supervisory and modal control. Conversely, discrete-event (DE) formalisms are used by software designers, where techniques such as object-orientation permit the modelling of complex supervisory control, but in which descriptions of plants must be greatly simplified.

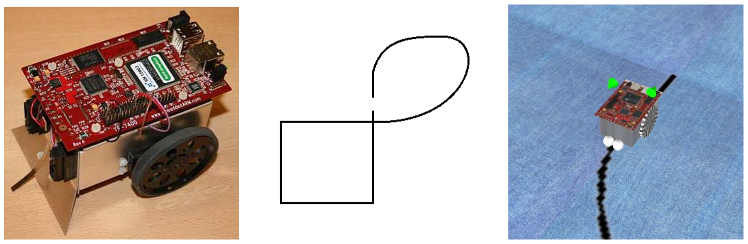
The DESTECS[1] project aims to bridge this gap by developing tools and methodologies that allow engineers and software designers to collaborate in building models of embedded control systems and to analyse them through simulation. Our approach is to provide a tool to allow DE and CT models to be executed simultaneously using existing tools, with data, events, and time shared between them. In our terminology, a *co-model* is a system model that includes a CT model and a DE model. Simulation of a co-model with the DESTECS tool is called a *co-simulation*. With co-models and co-simulation, engineers and software designers can collaborate to build and analyse models of embedded control systems using formalisms and abstractions with which they are familiar.

The primary benefit of the DESTECS approach is that co-models can describe systems that would be difficult to model within the CT or DE formalism alone. There are two primary challenges. First, in order perform co-simulations with predictable and valid results, the semantics of the two formalisms must be reconciled, and a tool built to manage co-simulation. Second, the cultural gap between the engineering and software disciplines must also be bridged. This includes describing the terminology and world view of each discipline to the other. To this end, DESTECS is also developing a set of *methodological guidelines* that will help engineers to collaboratively build and maintain co-models.

While the industrial case studies of DESTECS inform the content of the guidelines, they are by their nature relatively large co-models that evolve slowly and are worked on at individual sites, by one or two key personnel. A need was identified for a more agile, rapid feedback cycle to both generate and validate guidelines, which can then feed into the industrial case studies through a resulting “methods manual”. The pilot study presented in this paper was proposed to meet these needs, in which a collaborative, multi-site co-model development was undertaken in order to provide input to the guidelines and evaluate the emerging tools. A small line-following robot was selected as the basis for the study. This paper reports on the details of the co-model that was produced, as well as the experience in producing it.

Background and related work are given in Section II. The case study and co-model are described in Section III, with details of modelling faults and fault tolerance mechanisms given in Section IV. Experiences of collaborative modelling are given in Section V. Finally, conclusions are drawn in Section VI, with directions for future work identified.

[1]http://www.destecs.org/



(a) (b) (c)

Figure 1. A photo of the R2-G2P robot (a), an example path that the robot should follow (b), and a 3D representation of the robot generated by 20-sim (c).

II. Background and Related Work

The core concept of the DESTECS approach is that of a co-model. A co-model is a system model comprising a DE model of the control software and a CT model of the plant, with a *contract* that describes the communication between them. The contract is used to define shared variables (state shared between the two models); named events (raised in the CT model and handled by the DE model); and shared design parameters (values that affect the system design, but are fixed over the course of a co-simulation run). Shared variables can be either *monitored* variables (read by the DE model and written by the CT model) or *controlled* variables (written by the DE model and read by the CT model). The DESTECS tool allows such co-models to be defined and acts as a co-simulation engine, synchronising the time steps taken and data exchanged between the simultaneously executing tools during a co-simulation.

Within DESTECS, the Vienna Development Method (VDM) is used to model control software, supported by the Overture[2] tool [1]. VDM is a well-established formal method that includes features for object-orientation and concurrency [2], and features to describe real-time embedded systems [3]. The 20-sim[3] tool [4] is used to build, simulate and visualise CT models, it allows the dynamics of the plant to be modelled in a number of ways, including the powerful bond graph [5] notation: a domain-independent description for the dynamic behaviour of physical systems.

A methodology on co-simulation tools is described by Nicolescu et al. [6]. Ptolemy II [7] offers DE and CT simulation within a single tool, though lacks the mechanisms for structuring supervisory control of VDM and the component libraries offered by 20-sim. Work on how co-models can be combined, i.e. time synchronisation between the DE and CT elements, is described in *hybrid systems* literature, for instance, Cassandras et al. [8].

The terminology of Avižienis et al. [9] is adopted with respect to non-normative behaviours: a *failure* of a system is a deviation from its specified behaviour, caused by an *error* (some system state) propagating to the system boundary where the behaviour is observed. The (hypothesised) cause of this error is the *fault*. Within the methodology, the use of the terms *ideal*, *realistic* and *faulty* to describe a component's behaviour was found to be useful. The ideal behaviour is a model of a component's core functionality which ignores any deviations due to signal noise or manufacturing tolerances. A realistic model of such a component is more faithful to that of a real object and includes the previously ignored deviations. Faulty behaviours are those which cause a failure of the component. We do not give a prescriptive categorisation of behaviours into ideal, realistic and faulty, which are domain and experience specific.

[2]http://www.overturetool.org/

[3]http://www.20sim.com/

III. Pilot Study: A Line-Following Robot

The pilot study that is the focus of this paper involved production of a co-model of the small robot called R2-G2P (pronounced "*are-two gee-top*") shown in Fig. 1a. The pilot study focused on the task of line following, whereby the robot should follow a black line on the floor. The key features of the robot relevant to this task are: two wheels, connected to servo motors that provide movement and differential steering; and two infrared line-following sensors that sense the lightness of the floor based on infrared reflection.

An example line for the robot to follow is given in Fig. 1b. To follow the line, the controller uses the sensor data to always try to steer towards the line, whilst also moving forward. If it begins to lose the line, it alters the direction of its turn to try to stay on the line. In order to visualise the robot co-model, a 3D representation was created in 20-sim. A screenshot of this is shown in Fig. 1c. This view also shows the path on the floor in black (the floor texture is purely decorative and is seen as 'white' by the robot). Note that although the sensors on the real robot are placed underneath the body, those in the 3D view are placed on the front (represented as spheres) for the convenience of giving visual feedback about the state of the sensors. The colour of the sphere represents the lightness of the floor as interpreted by the sensor (i.e. if the sensor sees black, the sphere is black).

A. Co-model and Contract

The co-model of the robot consists of a DE controller model and a CT plant model. The CT model includes the main elements of robot (the body and wheels), models of the sensors, and data representing the line on the surface

	Name	Type	Notes
controlled	velocity_Left	real	range: [-1,1]
	velocity_Right	real	range: [-1,1]
monitored	lfLeft	int	range: [0,255]
	lfRight	int	range: [0,255]

Figure 2. Co-simulation contract for the R2-G2P co-model

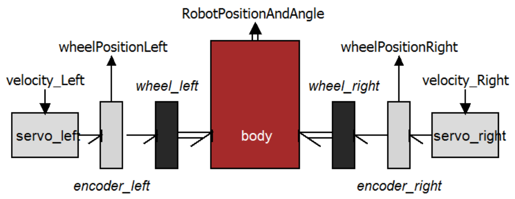


Figure 3. RobotPhysics submodel of the R2-G2P CT model

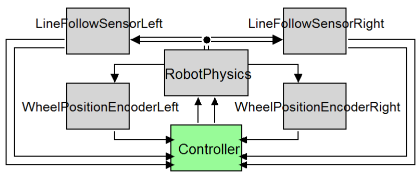


Figure 4. Top-level 20-sim model of the R2-G2P

under the robot. The DE model must read data from the two line-following sensors and control the speed of the two wheels. This suggests that four shared variables are required in the co-model: two monitored, one for each sensor; and two controlled, one for each wheel. These are realised in the contract in Fig. 2. In this simple task, only shared variables are used (no events and shared design parameters are required).

B. CT Model

The CT model can be split into two distinct parts, one in the signal domain and the other using bond graphs. Fig. 3 shows the physics model of the robot, described using a bond graph. This model includes continuous-rotation servo motors with speed control, wheels to translate the servo outputs to linear motion and a robot body that connects the two drive units and acts as a mass to be moved. Together these elements form the RobotPhysics submodel that can be seen in the top-level signal domain model, Fig. 4. This model also contains (left and right) LineFollowSensor and WheelPositionEncoder submodels, and a ‘controller’ block, which holds the CT model’s view of the shared variables.

C. DE Model

The DE model makes use of the object-orientation available in VDM, which offers structuring and encapsulation methods, and permits the use of Gamma et al. style design patterns [10]. The controller follows the style for VDM controller models described by Fitzgerald et al. [11]. A class diagram of the controller is given in Fig. 5. The core of the model is the `Controller` class, which defines a periodic thread that

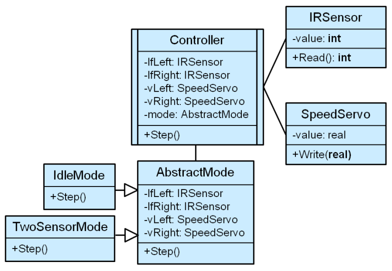


Figure 5. Modal R2-G2P VDM controller (class diagram)

calls a `Step` operation to perform the control loop. The line-following sensors and servos (actuators) are accessed through the `IRSensor` and `SpeedServo` classes respectively. Two objects of each type are used to access the four shared variables shown in Fig. 2. A special **`system`** class (not shown) is used to describe the architecture of the controller. In this case the controller object is deployed to a single CPU.

The controller is *modal*, broadly following the *state pattern* [10]. This means that the actual behaviour of the controller is delegated to a set of `Mode` classes, each of which implements the `AbstractMode` class. These classes encapsulate the behaviour of a particular mode and can easily be swapped at runtime. Initially, two modes were created: an `IdleMode`, which does nothing, and a `TwoSensorMode`, which follows the line using data from both line-following sensors. Further modes were introduced in order to add fault tolerance.

IV. Modelling Faults and Fault-Tolerance

The line-following sensors are modelled using the two-dimensional lookup table function provided by 20-sim. The data in the table is loaded from a bitmap, representing the surface under the robot. During simulation, the coordinates of the sensor are passed to the table, which yields the lightness of the surface under that point. The values output from the table are scaled to suit the expected outputs from the sensor before being quantised by an analogue to digital (A/D) block. This model represents the ideal behaviour of these sensors.

Faulty and realistic behaviours were then added to the line-following sensor model. A set of behaviours that could be included in the model was generated by considering the sensors in the light of the “SHARD guidewords” [12]. These guidewords help to guide the thinking of a person when considering what faulty and realistic behaviours a component might exhibit. The list of behaviours derived by applying these guidewords can be seen in Fig 6. From this list, three behaviours were modelled: stuck value (a failure), A/D noise and ambient light (both realistic behaviours).

A. Fault Modelling

The modelling of the two realistic behaviours necessitated modification of the ideal sensor model in different ways. Adding the effects of ambient light required the model to become more concrete than the previous version, by interpreting the values in the lookup table in a different way.

Guideword	Deviation
Subtle	Stuck value, A/D noise on lower bits, ambient light affecting readings
Coarse	A/D noise on higher bits
Early	None found
Late	Slow response to brightness change
Omission	Fail silent
Commission	None found

Figure 6. The fault/realistic behaviours generated using SHARD guidewords

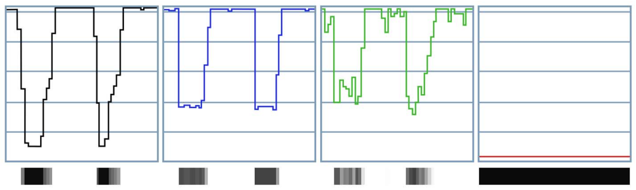


Figure 7. Example sensor output, with bars indicating the sensed data as the robot sweeps over a black line. Left to right: ideal sensor, ambient light added, A/D noise added, sensor value stuck.

Specifically, where the table values had previously represented the brightness of the floor directly, they were now interpreted to represent the reflectivity of the floor. This change allows the sensor to perform a calculation of the IR light incident on the floor and from this estimate the light that would be received by the sensor.

To add A/D noise, the decision was taken to modify the implementation of the standard A/D block provided by 20-sim. This new implementation adds a scaled amount of Gaussian noise to the input signal before it is quantised for consumption by the controller. The scaling factor is defined by the number of digital bits that may be subject to noise. This effectively defines a signal-to-noise ratio for the device. A second implementation for this behaviour was also considered, in which the Gaussian noise is generated by a standard 20-sim block and added to the signal before it enters the A/D block. This implementation has the advantage that the noise element, which would not be implemented in a real system, can be easily separated from the ideal behaviour that would be. The downside of the second implementation is that the magnitude of the noise is decoupled from the A/D parameters and so has to be updated manually if these are changed.

The final modification to the sensor model was to implement the stuck value behaviour. In DESTECS we advocate that such failure behaviour be separated from the desired behaviour and so the *fault injector pattern* was employed [13]. This pattern sees a single block effectively wrap the sensor model such that the inputs to and outputs from the sensor may both be intercepted and altered to implement a specific failure mode as required. A feature of 20-sim that allows multiple implementations for a single block was also used. In this case two implementations are provided: the "faulty" one, which exhibits the latent stuck fault that could be activated during co-simulation; and an alternative "no fault" implementation, which guarantees that no fault will not occur. This desired implementation can be selected before simulation commences.

Fig. 7 show samples of the outputs obtained from the sensor as it sweeps left and right over a black line. The leftmost sample shows the ideal behaviour of the sensor. The second reading is subject to ambient light, resulting in a reduced differential between the values for white and black. The third sample includes both ambient light and A/D noise. A combination of A/D noise and high ambient light levels may make it impossible to determine (from a single sample) if the sensor is over black or white. The final sample shows the output when the stuck value fault is activated.

B. Fault Tolerance Mechanisms

In order to tolerate the effects of ambient light, A/D noise and sensor failure, the DE controller incorporates a number of fault tolerance mechanisms implemented using design patterns [10]. A/D sensor noise is addressed by employing the *filter pattern* [13]. In this pattern a filter class, which provides the same interface as the sensor, is defined and instantiated in place of the original sensor object. The filter object holds a reference to the original sensor and all sensor readings pass through the filter to allow processing before the filtered value is passed on to the controller. In this case, the filter takes the mean of a series of readings to account for noise.

To handle the effects of ambient light, an additional mode, `CalibrationMode`, was added to the controller. This mode requires that initially one sensor is over a black area and the other over a white area. It then proceeds to take a series of readings from which the mean values for black and white are determined, which are then taken into account when making further readings. Once this is complete, the controller instantiates the original `TwoSensorMode` for line following. In the event of a sensor failing, the controller can attempt to follow the line using the remaining working sensor. To implement this, a `OneSensorMode` was also introduced. This is a degraded behaviour, requiring a slower sweeping motion. If the other sensor fails, the controller reverts to `IdleMode`, since it can no longer make progress.

V. Collaborative Modelling Experience

The purpose of the pilot study presented in previous sections was to gain experience in collaborative modelling with DESTECS, and to record problems that arose and communications between engineers needed to solve them. Two separate developments began, one between collaborators at different geographical sites and one on the same site. One aspect of interest was whether communications were different in the single-site development (where collaborator could quickly chat about problems in person) as opposed to the multi-site development (where communication requires emails and netmeetings). In the later stages of the study, the models were merged to form the single co-model described in the previous sections. The problems that arose are reported below, followed by a reflection on how they might have been avoided.

A. Initial Discussions

The single-site development began by writing a list of assumptions about the robot and environment (sensor ranges, size of the robot etc.) before defining an initial contract, in an attempt to minimise problems from implicit assumptions. The multi-site development defined an initial contract directly. Both developments encountered problems, despite the extra effort spent recording assumptions in the single-site development.

B. Integration Testing

Initial integration testing, in which the separate DE and CT models were combined into a co-model and tested, caused problems. Bugs were found in both CT models (one robot hardly moved and the other moved incredibly rapidly), which required feedback to the CT modellers and testing of new versions. This cycle was much quicker in the single-site development, but still took time. This led to the observation that truly parallel development, in which testing of the two models is performed within a co-model, is extremely difficult. In response to this, all further models were subject to greater testing before integration.

It is also worth noting that during initial integration testing, both developments were based on a small contract allowing control of motor power and feedback from wheel encoders. This meant that in theory both CT models could be tested with the same DE controller. However, although the types of the shared variables were the same, the meanings were different. In one contract, the encoder reported the actual distance travelled, while the other gave a rotation count that required further processing by the controller. This illustrates that knowledge of the type of shared variables alone may not be enough to use them correctly.

C. Model Evolution

Both developments followed the same stages of evolution: tracing a known path (e.g. a square), then adding line following with ideal sensors, and finally adding realistic and fault behaviours to the sensors. The initial addition of the sensors required extension of the co-simulation contracts with two new monitored variables per sensor. This did not cause problems since it simply represents an additional behaviour. The effect of adding faulty and realistic behaviours to the line-following sensors was significant on the DE side, because it represented a change in behaviour of an existing component (noisy readings and readings affected by ambient light). To handle these new behaviours, calibration and error detection operations were added to the DE sensor class using the *decorator pattern* [10], allowing the interface to be extended whilst maintaining backwards compatibility for regression testing.

D. Tool Problems

The addition of line-following sensors to one of the CT models caused a problem when it was used in a co-simulation, due to an incorrect filename and an error misreported by the co-simulation engine. Here the use of a third tool, the co-simulation engine, resulted in additional complexity in testing. To overcome this problem, a number of alternative sensor implementations were built in an attempt to find a workaround. This issue occurred in the single-site development, so new versions of the CT model could be quickly tested and discussed until a solution was discovered. The testing cycle of new CT models lasted for one day, but it is expected this would have taken longer in the multi-site development.

E. Model Merging

Towards the end of the study, elements from the two co-models were merged to form a single co-model. This involved merging the higher-fidelity physics model of one CT model with the sensor models from the other. This caused two problems. First, although the interface between the physics submodel was discussed —that the output should be the translation and rotation of the robot— it was discovered during testing that the angle of rotation assumed by each modeller was different. The physics submodel assumed that rotation is recorded counterclockwise from the x-axis, in line with standard practice, while the sensor models assumed clockwise rotation. This issue initially required the addition of numerous small 'fixes' to calculations and later a reworking of the model to follow standard practice. This could perhaps have been avoided if implicit assumptions made by modellers had been recorded, particularly the tacit knowledge of the engineer that "rotation goes counterclockwise". This is an example of a knowledge gap between two groups.

A second issue caused by this change was that the higher-fidelity physics submodel more closely resembled the real R2-G2P robot in that the wheels were attached to speed-controlled continuous-rotation servos, and not simple DC motors assumed by the chosen DE model. This evolution did not require a change in the contract or in the type of the controlled variables (the range of acceptable values in both cases is [-1,1]). It did however mean that the DE controller no longer needed loop (i.e. PID) controllers, since the interface to the servos is based on speed, not power: in essence, the speed-controlled servos have internal loop controllers. Again, the knowledge of the type of shared variables alone is shown to be insufficient: extra information needs to be communicated between modellers.

F. Reflection

While the collaborators successfully produced working co-models, effort was required in solving the problems described above. We can, in retrospect, suggest guidelines that could reduce the risk of other DESTECS users wasting effort in the same way. The first of these would begin to address problems due to interpretation of shared variables. It is apparent that while information regarding these variables was discussed, there were important details missing. Had extra information been added to the description of the variables then the difference in assumptions may have been avoided. We suggest that the following four properties of a shared numerical variable be made explicit as a minimum set of metadata: the SI unit or simple description of the value; the range of acceptable values;

the datum against which a value is measured and the direction of positive values or frame of reference.

The second guideline that we extract from the study so far recommends that models have an explicit set of acceptance tests performed before they are released for use by others within the project, in this case the other co-modeller. To support this we suggest that the CT and DE models are tested in a single domain initially (with a simple controller and plant, respectively), before integration into a co-model. These simple test models should only be good enough to gain confidence in the general performance of the model, and are not intended to replace the counterpart model in the co-model.

VI. Conclusions and Further Work

This paper presented a collaborative modelling exercise within the DESTECS framework, involving modelling of a line-following robot. Using the emerging DESTECS tools and methodologies, a CT modeller and DE modeller were able to collaborate to build a co-model of the robot that could be analysed through co-simulation. This co-model comprises a CT model of the robot, a DE model of the controller and a contract defining the variables shared between them (signals to the servos and from the line-following sensors). Both the CT and DE models required a detailed knowledge of the respective languages and tools, which would require a lot of time and effort for one person to learn.

The project has demonstrated two distinct benefits of the approach. The first is that the use of a contract between the DE and CT domains gave a clear boundary for each engineer to work in, allowing each to work on their individual parts with only a general overview of the other model. The second benefit was that each engineer was able to work in their own discipline, with the semantic differences between the modelling tools being reconciled transparently by the DESTECS tool.

As the co-model evolved, a number of problems arose that required communication between the modellers in order to be resolved: changing requirements; the introduction of realistic and fault behaviours; and integration of two CT models. A clear conclusion is that the main problems in "co-modelling" are similar to those of collaboration in general: there will always be assumptions that have not been made explicit, but this can be mitigated by documenting as many assumptions as possible. One novel problem caused by co-modelling is the need for communication between disciplines. It was very difficult to find all problems without one co-modeller having at least some knowledge of the other domain's model. Again, this could perhaps be addressed through greater documentation in addition to the co-simulation contract. Communication between geographically separated sites also slowed down the co-modelling process due to slower cycles of testing and more protracted discussions. The addition of a third tool (the co-simulation engine) also caused problems, for example the problem of the suppressed error message described previously could not be reproduced in single-domain testing. This may not be an problem in future however as tool support improves.

Looking to future work, the experience and guidelines produced as a result of the study will be validated by the industrial partners of the DESTECS project. The experiences here will be linked with industrial experience to produce a valuable "methods manual" for collaborative modelling. Effort is also being directed toward formally modelling consistency between DE and CT models such that some guidelines may be automatically checked. A clear goal is to continue collaborative modelling with the robot, expanding the co-model from a single robot to a "swarm" of multiple robots collaborating on a more complex task, such as collective transport.

Acknowledgments

The authors would like to thank Stefan Groothuis for his help with 20-sim modelling; John Fitzgerald and Sune Wolff for their comments on draft versions; and other colleagues from the project for their valuable input. 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 (DESTECS). Additionally the work of Pierce and Gamble is supported by the UK EPSRC Platform Grant on Trustworthy Ambient Systems.

References

[1] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef, "The Overture Initiative – Integrating Tools for VDM," *ACM Software Engineering Notes*, vol. 35, no. 1, January 2010.

[2] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object–oriented Systems*. Springer, New York, 2005. [Online]. Available: http://www.vdmbook.com

[3] M. Verhoef, P. G. Larsen, and J. Hooman, "Modeling and Validating Distributed Embedded Real-Time Systems with VDM++," in *FM 2006: Formal Methods*, J. Misra, T. Nipkow, and E. Sekerinski, Eds. Lecture Notes in Computer Science 4085, 2006, pp. 147–162.

[4] J. F. Broenink, "Modelling, Simulation and Analysis with 20-Sim," *Journal A Special Issue CACSD*, vol. 38, no. 3, pp. 22–25, 1997.

[5] V. Duindam, A. Macchelli, S. Stramigioli, and H. Bruyninckx, *Modeling and Control of Complex Physical Systems*. Springer, 2009.

[6] G. Nicolescu, H. Boucheneb, L. Gheorghe, and F. Bouchhima, "Methodology for efficient design of continuous/discrete-events co-simulation tools," in *High Level Simulation Languages and Applications*, J. Anderson and R. Huntsinger, Eds. San Diego, CA: SCS, 2007, pp. 172–179.

[7] 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, January 2003.

[8] C. G. Cassandras, *Analysis and design of hybrid systems: a proceedings volume from the 2nd IFAC conference*. Elsevier, Jun. 2006. [Online]. Available: http://www.sciencedirect.com/science/book/9780080446134

[9] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[10] R. E.Gamma, R.Helm and J.Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software.*, ser. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.

[11] J. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef, "A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems," *To appear in Mathematical Structures in Computer Science*, 2011, see also Tech. Report 1264, School of Computing Science, Newcastle University, UK, July 2011.

[12] D. Pumfrey, "The principled design of computer system safety analyses," Ph.D. dissertation, Department of Computer Science, University of York, UK, September 1999. [Online]. Available: http://www.cs.york.ac.uk/~djp/publications/Thesis16.pdf

[13] J. F. Broenink, J. Fitzgerald, C. Gamble, K. Pierce, Y. Ni, and X. Zhang, "D2.2 — Methodological Guidelines 2," The DESTECS Project (INFSO-ICT-248134), available from http://www.destecs.org/, Tech. Rep., December 2011.