

Modelling Faults and Fault Tolerance Mechanisms in a Paper Pinch Co-model

Ken Pierce, John Fitzgerald, and Carl Gamble

School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, United Kingdom.

{john.fitzgerald,kenneth.pierce,carl.gamble}@ncl.ac.uk

Abstract. This paper¹ demonstrates the modelling and simulation of errors and fault tolerance mechanisms for embedded systems, using co-models that combine discrete-event models of control software with continuous-time models of controlled plant. The approach has been realised using the VDM and 20-sim formalisms with a co-simulation engine that coordinates simulations running in their respective tools. The paper introduces the use of patterns for the formal modelling of errors and fault tolerance mechanisms in this setting, giving illustrative safety kernel and voter patterns, demonstrating their application in a case study based on paper processing machinery.

1 Introduction

The creation of dependable embedded control systems is difficult because of increasingly demanding requirements, increasing use of distributed control, and increasing time pressures in a competitive market. In applications where such systems have to be trusted, the need to take account of non-normative behaviours (faults, errors and failures) by means of fault tolerance adds to the complexity of the design and implementation. The use of (formal) modelling and simulation can aid engineers in the rapid construction and evaluation of designs. Development of embedded control systems is however cross-disciplinary, requiring collaboration between engineers and software designers, who are used to different cultures, abstractions and formalisms. Typically, continuous-time (CT) formalisms are used in the design of physical systems, while discrete-event (DE) formalisms used in the design of software.

In the DESTTECS² project, we are building tools and methods that can aid cross-disciplinary collaborative modelling and analysis (through simulation) of embedded control systems. Our approach is to support the construction of models in diverse formalisms and tools, but to integrate their simulation through a tool-supported framework that reconciles their different semantics. Results so far include proof-of-concept tool, validated through a growing set of collaborative models driven by industrial case studies [FLP⁺10]. In this paper, we extend previous work by showing how faults and fault

¹ To appear in *Proceedings of the ERCIM/EWICS/Cyberphysical Systems Workshop at SAFE-COMP 2011, Naples, Italy, September 2011*. Please cite that version in preference to this.

² <http://www.destecs.org/>

tolerance mechanisms can be analysed in this framework, and describe two specific fault tolerance design patterns.

In our terminology, a *co-model* is a system model that includes a CT model of the plant (the part of the system to be controlled) and a DE model of the controller. The DESTECS tool connects 20-sim³, which allows CT models to be expressed as differential equations or in bond graphs, and the Overture tool⁴ [FLS08,LBF⁺10], which supports the construction and simulation of models in VDM (Vienna Development Method). VDM is a well-established formal method that has been extended to include object-orientation and concurrency [FLM⁺05], and features to describe real-time embedded systems [VLH06,LFW09]. The DESTECS tool allows the 20-sim and Overture tools to participate in a *co-simulation* and communicate through shared variables and events. The synchronisation of variables and passing of events is handled automatically by a *co-simulation engine* within the tool. This allows engineers to work in their preferred formalism while collaborating to achieve a coherent system model which better represents the system under design than a single CT or DE model.

Section 2 introduces an example co-model of a *paper path* that is explored later in Sections 3 and 4, which introduce faults and fault tolerance patterns, respectively. Finally, Section 5 draws conclusions and lays out plans for our future work.

2 A Case Study in Co-simulation: a Paper Pinch

Paper paths are cyber-physical systems that transport paper in applications such as printing or envelope filling. Figure 1 shows a simple path in which paper is conveyed from left to right using four *paper pinches*. Each pinch consists of two rollers, one of which is driven by a motor, that impart motion to the paper through friction. A controller sets the speed of the motor using pulse width modulation (PWM). In order to do this, it needs to know how fast the motor is going; to this end, the motor is fitted with a rotary encoder that counts as the motor turns. This system was originally modelled in the *Boderc* project [HM07], with separate CT and DE models. While the modelling process saved many person-years of effort by allowing engineers to skip at least one complete physical machine-build iteration, errors were discovered in the physical prototype that caused paper jams at high speeds due to miscommunications between disciplines. This case study was one of the motivations for the DESTECS co-modelling approach.

Figure 2 shows the elements of a co-model of a paper pinch, with a partial VDM class diagram (left), a graphical 20-sim model (right), and co-simulation contract (bottom). The contract shows the shared variables of the co-model, in this case, the output of the rotary encoder and the PWM value set by the controller. These values are synchronized automatically by the co-simulation engine during co-simulation. Example output from this co-model can be seen in Section 3. The 20-sim model contains a model of the dynamics of the system (paper, rollers, motor, and friction), a model of an encoder, a controller block and a digital-to-analogue conversion block. The contents of the encoder submodel are also shown, indicating that the encoder count is computed as the integral of the motor speed. The VDM class diagram shows a `Controller` class,

³ <http://www.20sim.com/>

⁴ <http://overturetool.org/>

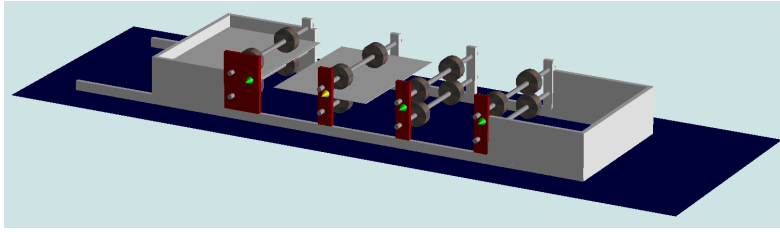


Fig. 1. Graphical model of a paper path generated by 20-sim

which accesses the shared variables through sensor and actuator interfaces. Other details are omitted for brevity. The use of interfaces in this way allows the exploration of different sensor and actuator implementations without the need to change the controller. This is a key aspect of the work on fault tolerance patterns in Section 4. Further details of this approach to controller design in VDM can be found in [FLPV11].

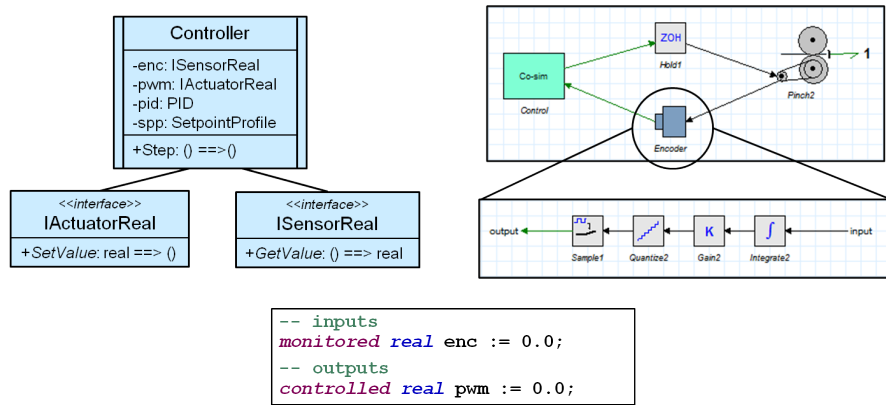


Fig. 2. Elements of a paper pinch co-model: partial VDM class diagram (left), 20-sim CT model (right), co-simulation contract (bottom)

3 Modelling Faults

We introduce two faults into the encoder model in order to observe the failure of the controller to meet the specification (achieving the correct speed of the motor). In the terms of Avižienis et al. [ALRL04], we model a *failure* of the encoder because it deviates from its specification of accurately reporting the distance travelled by the motor. We do this by modelling the *error* state which leads to the failure and not the *fault* that is its root cause. We therefore use the term “fault” in a loose sense. In order to select which faults to model, we have used guidewords suggested by Pumfrey [Pum99], which are

similar to those of hazard and operability studies but with a software focus. From this, we select a *coarse* and a *subtle* failure. Coarse failures are detectable in the measured value, such as stale or missing values (e.g. encoder no longer rotates) or random jumps in values (e.g. flipped bit). Subtle failures are not detectable without a diverse source of data (e.g. drift caused by missing or adding counts).

We model a flipped bit and encoder drift by introducing a “fault” block to the CT encoder model. This block can have different implementations that each introduce different errors. The specific implementation can be selected before co-simulation, and the fault can be activated by the DESTTECS tool during the co-simulation at a given time or in response to a certain event. This captures the idea of a *latent fault* [ALRL04] that becomes active during the co-simulation. The fault blocks are shown in Figure 3. The (subtle) drift is introduced before integration and the (coarse) bit flip is introduced after integration.

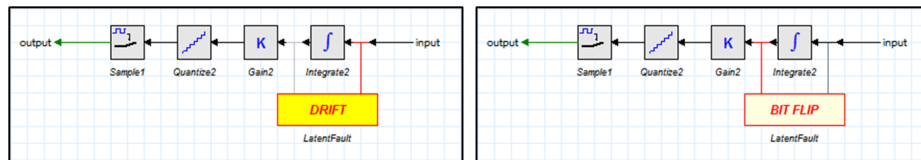


Fig. 3. Fault blocks in the encoder submodel in 20-sim

Outputs from the co-model when these two faults occur are shown in Figure 4. In the top graphs, the straight line shows the setpoint (desired speed) and the fluctuating line the actual speed. Where the actual speed is outside tolerance of the setpoint, a bar is added at the bottom of the graph. The lower graphs show the output of the encoder, with the bar indicating when the latent fault is active. The bit flip event can be identified by the sudden large change in encoder output, consistent with a coarse failure. The subtle drift event, however, is not visible in the encoder output alone. Only observation of the actual motor speed reveals that the fault is active. Note also that the controller naturally compensates for the large change in encoder value caused by the bit flip, but achieves this through a sudden large change in motor speed.

4 Patterns for Fault Tolerance

Patterns for object-oriented software [GHJV95] are templates that outline a possible solution for a specified problem. A pattern description may include the pattern’s motivation, intent, and structure, possible consequences of use, previous experiences of use, and sample code. A key part of our work is to identify and describe patterns for fault-tolerance mechanisms that can aid engineers in achieving dependability through collaborative co-modelling. We believe that patterns for fault-tolerance mechanisms are directly applicable to VDM, and can be adapted for models in 20-sim. In this section, we present two patterns that describe mechanisms that could be deployed in response to the faults outlined in Section 3: the safety kernel pattern and the voter pattern.

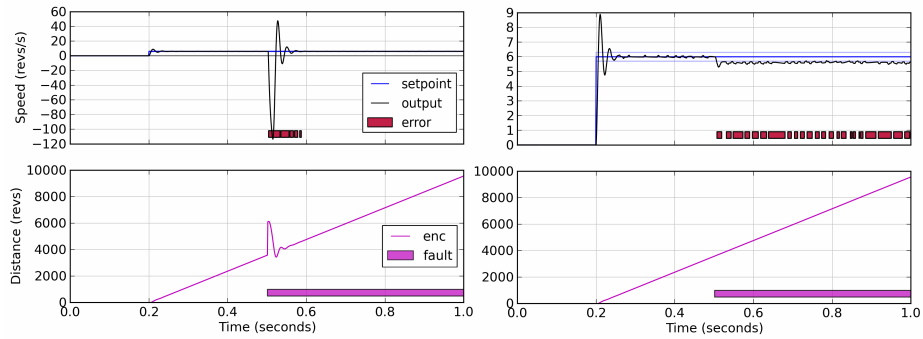


Fig. 4. Graphs showing motor speed and encoder reading for the paper pinch co-model subject to a bit-flip fault (left) and drift fault (right)

4.1 Safety Kernel Pattern

A kernel is a small, verifiable component that guarantees some property of a system, typically security or safety [Rus89], by protecting the controller from making unsafe control actions. This is achieved by guarding access to actuators, thus preventing faults of commission [ALRL04]. We extract a pattern from this idea (Figure 5) which can be applied where controller actions may be unsafe.

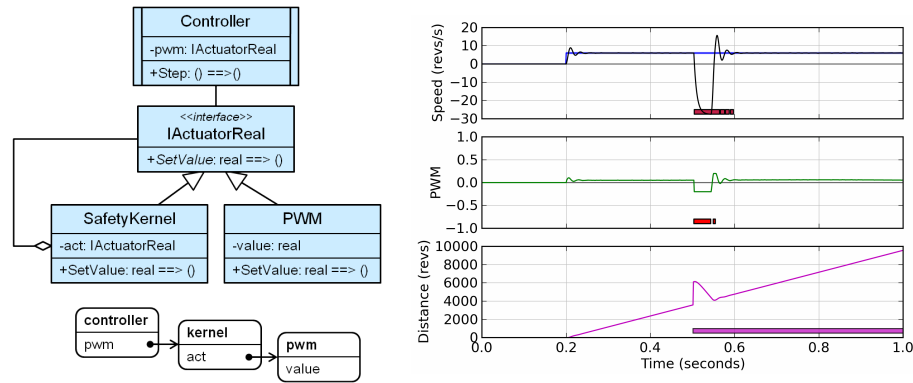


Fig. 5. Safety kernel pattern: class and object diagrams (left) and output graphs (right)

Figure 5 shows a class diagram of the pattern structure (left), with an object diagram (below) and a graph of the output (right). In this pattern, a `SafetyKernel` class is introduced that implements the actuator interface and encapsulates an actuator. By implementing the interface, a kernel object can be passed transparently to the controller (such that the controller does not need to be altered) and can intercept all calls to the actuator, avoiding unsafe control access when necessary. This structure is clearer in the

object diagram, which shows that the controller must access the actuator through the kernel (the variable *value* in the **pwm** object is the shared variable). In this case study, we assume that the sudden large increase in speed caused by the bit flip fault is deemed unsafe (see Figure 4). In the output graph in Figure 5, the kernel was instantiated to limit PWM values to ± 0.2 (20% power). The plot of the PWM value shows the safety kernel intervening almost immediately after the bit flip fault occurs, limiting the output.

4.2 Voter Pattern

Where sensors can fail, multiple sensors can be introduced as a way to achieve dependability. This can be done through replication (using copies of the same sensor) or diversity (using different sensors). In order to gain a single value from various inputs, a voter can be used. A simple voter could take the mean of the incoming values, or use a majority vote to ignore erroneous readings. A *voter pattern* based on this idea (Figure 6) can be used to make use of replicated (or diverse) sensor inputs.

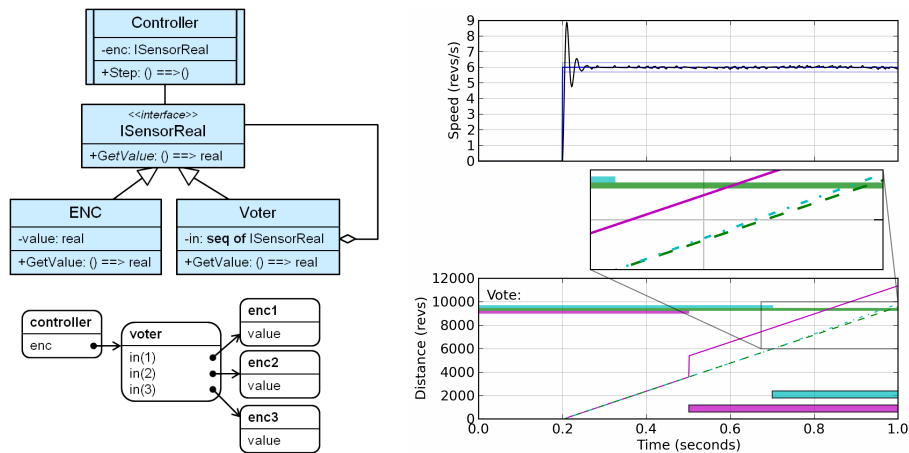


Fig. 6. Voter pattern: class and object diagrams (left) and output graph (right)

Figure 6 shows a class diagram of the pattern structure (left), with an object diagram (below) and a graph of the output (right). In this pattern, a **Voter** class is introduced that implements the sensor interface. This class also aggregates one or more sensors. By implementing the sensor interface, a voter object can be passed transparently to the controller (such that the controller does not need to be altered) and provides a single value from the multiple inputs. The voter pattern can be combined with the *strategy pattern* [GHJV95] by describing the voting algorithm as an interface and providing different implementations of this interface to explore alternative voting routines.

In order to demonstrate this pattern, the co-model was altered to include three encoders. By selecting different implementations for the fault block in each encoder, we assert that our model uses diverse sensor inputs. In the output graph in Figure 6, the

three lines (labelled with ‘Vote’) indicate the encoder(s) that won the vote at each point in time. The bit flip fault occurs in one encoder after 0.5 seconds and this value is therefore ignored by the voter. At 0.7 seconds, a drift occurs in another encoder (see highlighted portion of the lower graph in Figure 6). In response to this, the voter chooses the non-faulty encoder by default. This default was implemented for demonstration purposes, but could represent the result of a choice to have a high-cost, better-trusted encoder alongside two lower quality but cheaper encoders. Note that the speed of the motor remains correctly within tolerance.

5 Conclusions and Further Work

We have demonstrated the modelling of errors and failures in co-models combining CT models of plant (in 20-sim) with DE models of control software (in VDM). The results generated by co-simulation on the paper pinch and other examples suggest that the patterns proposed are intuitive and function satisfactorily.

Our work supports heterogeneous formalisms in co-models of distributed embedded systems without requiring common operational semantics. Work on Matlab/Simulink, such as [CAH⁺06], relies on Simulink semantics, while other tools take more formal approaches, such as Scilab/Scicos [CCN06]. MODELISAR⁵ provides a *functional mockup interface* for tool coupling. Ptolemy-II [LZ07] supports heterogeneous modelling and design, while integration of behaviour models with component-based environment models has been proposed for Modelica [MDF11]. However, little current research addresses the methodology of fault and fault tolerance modelling in this context.

The DESTTECS project is developing further support tools for performing multiple co-simulations on alternative co-models, allowing exploration of the design space of alternative fault tolerance mechanisms under a range of scenarios. It is also conducting industrial case studies which include paper path management, control of heavy machinery, and a metastable Segway-like personal transporter device. The last study in particular provides opportunities to explore the modelling of safety kernel and monitor architectures.

Much remains to be done to develop the approach further. We intend to develop a much wider range of fault and fault tolerance models, on both the CT and DE sides, including n-variant mechanisms and self-checking. Although the state of the CT model is easily represented by graphs during runtime, there is no corresponding visualisation of the DE controller state. In addition, techniques are needed to manage the set of co-models, scenario inputs and test results that grows as co-models evolve and the range of fault tolerance mechanisms is explored during development.

Acknowledgements

The authors’ work is supported by the EU FP7 project DESTTECS and by the UK EPSRC Platform Grant on Trustworthy Ambient Systems.

⁵ <http://www.modelisar.com/>

References

- [ALRL04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.
- [CAH⁺06] A. Cervin, K.-E. Arzen, D. Henriksson, M. Lluesma, P. Balbastre, I. Ripoll, and A. Crespo. Control loop timing analysis using truetime and jitterbug. In *Computer Aided Control System Design, 2006 IEEE Intl. Conf. on Control Applications, 2006 IEEE Intl. Symp. on Intelligent Control*, pages 1194–1199. IEEE, October 2006.
- [CCN06] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhan. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer, 2006.
- [FLM⁺05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object-oriented Systems*. Springer, New York, 2005.
- [FLP⁺10] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, Marcel Verhoef, and Sune Wolff. Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems. In D. Méry and S. Merz, editors, *IFM 2010, Integrated Formal Methods*, volume 6396 of *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, October 2010.
- [FLPV11] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *To appear in Mathematical Structures in Computer Science*, September 2011. Preliminary version in Technical Report 1264, School of Computing Science, Newcastle University, UK, July 2011.
- [FLS08] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *ACM Sigplan Notices*, 43(2):3–11, Feb. 2008.
- [GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.
- [HM07] Maurice Heemels and Gerrit Muller, editors. *Boderc: Model-based design of high-tech systems — A collaborative research project for multi-disciplinary design analysis of high-tech systems*. Embedded Systems Institute, Eindhoven, The Netherlands, 2007.
- [LBF⁺10] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *ACM Software Engineering Notes*, 35(1), January 2010.
- [LFW09] Peter Gorm Larsen, John Fitzgerald, and Sune Wolff. Methods for the Development of Distributed Real-Time Systems using VDM. *International Journal of Software and Informatics*, 3(2-3), October 2009.
- [LZ07] Edward A. Lee and Haiyang Zheng. Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems. In *Proceedings of EMSOFT '07*, Salzburg, Austria, September 30–October 3 2007. ACM.
- [MDF11] T. Myers, G. Dromey, and P. Fritzson. Comodeling: From requirements to an integrated software/hardware model. *IEEE Computer*, 44(4):62–70, April 2011.
- [Pum99] David Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, York, UK, Sep 1999.
- [Rus89] John Rushby. Kernels for safety? In *Safe and Secure Computing Systems*, pages 210–220. Blackwell Scientific Publications, 1989.
- [VLH06] Marcel Verhoef, Peter Gorm Larsen, and Jozef Hooman. Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 147–162. Lecture Notes in Computer Science 4085, 2006.