Resilience Modelling Through Discrete Event and Continuous Time Co-Simulation

Zoe Andrews, John Fitzgerald
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK
{Z.H.Andrews, John.Fitzgerald}@ncl.ac.uk

Marcel Verhoef
Chess and Radboud University Nijmegen
PO Box 9010, 6500 GL Nijmegen, NL
Marcel.Verhoef@chess.nl

Abstract

We propose an approach to discrete event and continuous time co-simulation that permits the analysis of alternative fault-tolerance strategies in formal models of embedded systems at early design stages. The approach is based on the use of a model-oriented specification language with a continuous time simulator modelling controlled processes. This permits the explicit modelling of faults and the analysis of the resilience properties of a design.

1 Introduction

In the early stages of developing an embedded system, the designer is often faced with highly volatile requirements. In such a context, understanding faults and selecting fault tolerance mechanisms is a considerable challenge. Modelling tools should help to alleviate this problem by providing rapid feedback on the consequences of selecting design alternatives. However, few such tools provide facilities for the explicit modelling of faults and the analysis of their effects. We propose an extension to an established modelling framework to allow the specification of faults in hybrid models of real-time systems. We outline the technology (Section 2) and indicate how it has been extended to accommodate continuous time simulations of controlled processes. We then consider the integration of faults into such a hybrid model (Section 3). Finally, we consider the potential for further work on the semantics and pragmatics of this framework.

2 VDM++ Technology

VDM++ [1] is an object-oriented, model-based formal specification language supported by industry strength tools (http://www.vdmtools.jp/en). It supports the construction of abstract system models composed of class specifications, each of which contains definitions of data types, instance variables and methods. Types may be simple such as *bool* or *nat*, or abstract collections types such as maps, records and object references. Functionality is described in terms of operations that can be described explicitly or underspecified. A class can be made "active" by specifying a thread, i.e. a sequence of statements which are executed to completion. Extensions to support modelling real-time embedded and distributed systems have been proposed [2], including primitives for modelling deployment to distributed hardware and support for asynchronous communication. Co-simulation of continuous time models of controlled processes in 20-SIM (http://www.20sim.com) from within a discrete-time model of the controller in VDM++ have recently been demonstrated [3]. However, the technology lacks a clean approach to explicit specification of faults that would support comparison of fault tolerance strategies.

```
class Controller
instance variables
  level : real := 0.0
  valve : bool := false
operations
  async public  open: () ==> ()
  open () == valve := true;
  async public close: () ==> ()
  close () == valve := false;
end Controller
```

Figure 1. Controller model in VDM++

A classical control problem is used to discuss co-simulation in VDM++ and 20-SIM. Consider the design of a control system that maintains an acceptable level of water in a tank. There is a constant flow of water into the tank and a tap at the bottom to control the water level. The controller responds to both time triggered events (the controller queries the current water level at regular intervals) and state triggered events (an event occuring in the continuous time simulator, such as the water level reaching a limit). At the occurence of each of these events the controller determines whether the tap should be open or closed depending on the water level provided to it by the continuous time simulator. Here we focus on state triggered events only. Figure 1 shows a VDM++ model of the controller. The instance variables *level* and *valve* are shared with the continuous time model of the watertank, which is described by differential equations. The asynchronous operations *open* and *close* are executed when the high- or low water sensor is triggered.

3. Modelling Faults

Currently the main focus of formal methods is to check a model of a system for correct behaviour. We are interested in extending this to include consideration of the likelihood of correct behaviour in the presence of faults. Our approach is to extend the VDM++ framework [1] to allow fault assumptions to be explicitly stated within a model.

Extending the VDM++ framework in this way allows a system designer to experiment with fault tolerance strategies and get early feedback about which works best. For example, in the water tank scenario it would be possible to model the situation where the level sensors can fail. The designer would need to investigate different strategies for coping with such failures. Assuming a cost limitation of six sensors, how should they be positioned to get maximum benefit and reduce the probability of the water level going past the limits? Two configurations are shown in Fig. 2. In Fig. 2a the sensors are replicated at each water level limit and majority vote is used to determine the level. This would tolerate one faulty sensor and detect two sensors failing independently. In Fig. 2b, the two sensors placed within the limits provide a warning signal when nearing the boundaries, which could trigger an event that increases the frequency of the time-triggered level checks. The sensors outside the limits provide an emergency indication that action is needed to bring the water level back within limits. Both solutions are designed to improve the chances of the water level remaining within acceptable limits, but which gives the higher probability of this? We aim to link VDM++ tools with some probability tools to provide both analytical and simulation-based exploration of questions like this.

For the tank case study, initial experiments have shown that it is possible to inject faults in the interface between the continuous time simulator and the discrete event controller, and have highlighted some interesting issues. This provides us with the capability to inject, reject or modify events or continuous variables on the fly, in both the time and value domain, without affecting either the 20-sim or the VDM++ model. We believe this is conceptually very powerful, since our explicit failure models are kept orthogonal to the system model. For example, it is straightforward to inject a constant discrepancy between the actual value of the water

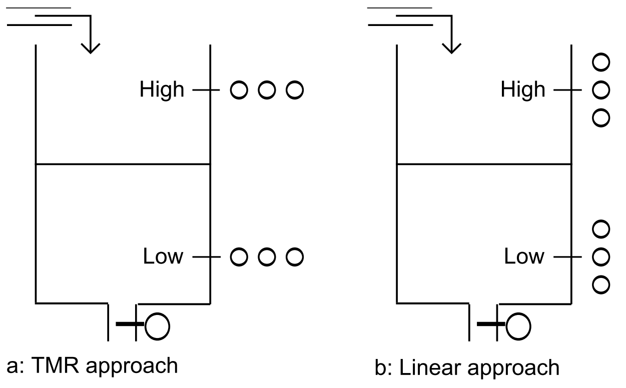

Figure 2. Sensor configurations

level and the value that the discrete event controller sees. An effect of such a fault, which is highlighted by the simulation tools, is that the continuous time model could signal a state event as the water level reaches one of the limits, but because the controller sees a different value (one within acceptable limits) no action is taken. The designer may wish to consider having a way in which the discrete event controller can distinguish between state and time triggered events. Such design alterations would be harder to make if they were not discovered until the testing phase of production, thus it is important to have tools to highlight these issues in the early stages of design.

4. Further Work

The experiments using the tank example as outlined in this paper will provide an initial idea of how best to integrate stochastic properties and reasoning into VDM++ models. Further case studies are proposed to extend and validate this approach. These will increase not only in complexity, but also in the level of assurance required, and as such the semantics of any extensions made to VDM++ would need to be formally defined to allow reasoning about properties of the model.

References

[1] J. Fitzgerald, P. Larsen, P. Mukherjee, N. Plat, and M. Verhoef. *Validated Designs For Object-Oriented Systems*. Springer-Verlag, 2005.

[2] M. Verhoef, P. Larsen, and J. Hooman. Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods*, LNCS 4085, pages 147–162. Springer Verlag, 2006.

[3] M. Verhoef, P. Visser, J. Hooman, and J. Broenink. Co-simulation of distributed embedded real-time control systems, 2007. To appear in Proc. IFM 2007: Integrated Formal Methods, Springer Verlag, 2007.