SOME ISSUES IN MODEL-BASED DEVELOPMENT FOR EMBEDDED CONTROL SYSTEMS

Paul Caspi
caspi@imag.fr
Verimag-CNRS
www-verimag.imag.fr

Abstract This presentation aims to discuss the needs for better and more solid foundations of model-based development in embedded control systems. Three particular points are discussed: a comparison between model-based development in control and in computer sciences, the need for a sampling theory of discrete event systems and the need for precise implementation methods based on preemptive scheduling.

Keywords: model-based development, embedded control, approximation, sampling, voting, distances

1. INTRODUCTION

Model-based development is widely recognised as a method of choice for efficiently and safely designing computing systems. After all, isn't it the way other branches of engineering have followed for achieving such a goal? Just think of how bridges and buildings are designed. Yet, though the need for model-based development is widely recognised, it is true that advances in this direction are quite slow and charges are put on both the youth of computer science and the intrinsic complexity of computing to account for this state of affairs.

There is however a particular subdomain, the embedded control domain, where things have progressed faster. For instance, automatic code generation from high level model have been in use at Airbus in the fly-by-wire department for more than twenty years [7]. Since the beginning of the nineties, the Simulink/Stateflow tool-box also allows automatic code generation (RealTime Workshop) and has achieved an impressive diversity of possible implementation platforms ranging from simple cyclic monoprocessor ones to multi-threaded ones, based on preemptive scheduling and to distributed ones based on specialised CAN or TTA libraries. It is not unfair to say that embedded control is by now one (if not the only one) computing subdomain that has reached the highest possible level of model-based development.

However, as it is often the case, this fast progress has been achieved rather empirically, without taking much care of foundations. Basically, it is the accomplishment of practitioners rather than of theoreticians and the latter have cast very little attention to it. The thesis we would like to support in this presentation is that times have come to strengthen the foundations of the method. Not only this effort can be expected to be fruitful for intellectual purposes but it is also likely that practitioners can benefit from it by getting better, with wider scope and simpler development tools.

So the aim of this presentation is to discuss this issue: which are the foundation needs? Three points will be more precisely considered:

1 What is the use of models in control and how this use differs from what is currently considered in computer science?

2 Is there a well-admitted theory of computer implementation for control models, in particular concerning the sampling of discrete event systems?

3 How can we guarantee behaviour equivalence between models and implementations in case of preemptive scheduling?

2. MODEL-BASED DESIGN IN COMPUTER SCIENCE AND CONTROL

Model-based design is advocated in both theories as a method of choice for efficiently and safely building systems. However these theories differ in the way of achieving this goal:

In computer science, the proposed method (see for instance [1]) is based on successive refinements: a large specification is designed first, imprecise (non deterministic) in general, but sufficient for meeting the desired system properties. Then implementation details are brought in progressively, making the specification more and more precise, while keeping the properties, up to a point when it can be implemented. Clearly, this is an ideal scheme which is seldom fulfilled in practice, but which has a paradigmatic value.

In control science, on the contrary, an exact model is built first, which allows a control system to be designed. Then the various uncertainties that may affect the system behaviour are progressively introduced and it is checked that the designed controller is robust enough to cope with these uncertainties.

Clearly, these two schemes are not, in practice, too far from each other. But, as control systems are mostly implemented by now on computers, some effort is needed if these two schemes have to match more closely. This can be valuable in the perspective of achieving an easier communication between computer and control cultures. A way to reach this goal would be to see the initially precise control model as representing a large class of models, those models which fall within some given "distance" from this model. This distance would then represent the maximally admissible uncertainty around the model and further refinements would make this uncertainty smaller. This goal requires thus some notion of control system distance and approximation.

3. SAMPLING DISCRETE EVENT AND HYBRID SYSTEMS

Large modern control systems mix very closely continuous and discrete event systems. This is due for instance, to mode changes, alarms, fault tolerance and supervisory control. From a theoretical point of view, computer implementation techniques for these two kinds of activity are quite different. Continuous control is dealt with through periodic sampling (time-triggered computations as defined by [5]) while discrete event systems use event-triggered implementations. However, in practice, many mixed continuous control and discrete event control systems are implemented through periodic sampling. This is the case, for instance, in Airbus fly-by-wire systems [7] and many other safety-critical control systems. The problem is that there are no solid foundations to periodically sampling discrete event systems and practitioners rely on in-house "ad-hoc" methods. Building a consistent sampling theory for mixed continuous control and discrete event systems would help in strengthening these practices.

A situation where such lack of theory is particularly critical concerns fault-tolerance: though the theory of distributed fault-tolerant systems [8; 5] advocates the use of clock synchronisation, still many critical real-time systems are based on the GALS (globally asynchronous, locally synchronous) and, more precisely, the "Quasi-Synchronous" [3] paradigm: in this framework, each computer is time-triggered but the clocks associated with each computer are not synchronised and communication is based on periodic sampling: each computer has its own clock and periodically samples its environment, *i.e.,* the physical environment but, also, the activities of the other computers with which it communicates. When such an architecture is used in critical systems, there is a need for a thorough formalisation of fault tolerance in this framework.

4. FAITHFUL IMPLEMENTATIONS BASED ON PREEMPTIVE SCHEDULING

A key question in model based development is the possible discrepancy between models and their computer implementations. As a matter of fact, if this discrepancy is too large, the benefits gained from the use of models can be spoiled. This is the general question investigated in section 2 where this question is considered in terms of distances and topologies. Yet there are particular situations where other approaches can be used. A typical example is found when implementations are based on multiple theads and preemptive scheduling. This kind of implementation is mandatory in several cases, for instance:

- in multi-periodic models for efficiency reasons;
- in event-triggered systems when urgent events have to be handled.

In such systems, inter-task communication is likely to be strongly non deterministic [2]. In some cases, for instance when discrete events are considered, critical races can take place and the "distance" between models and implementations may become too large. There is thus a need for more precise implementation techniques, which do not spoil the benefits of model-based development such as those described in [4; 6].

REFERENCES

[1] Abrial, J.-R. (1995). *The B-Book.* Cambridge University Press.

[2] Caspi, P. and Maler, O. (2005). From control loops to real-time programs. In Hristu, D. and Levine, W., editors, *Handbook of Networked and Embedded Computing Systems.* Birkhäuser.

[3] Caspi, P., Mazuet, C., Salem, R., and Weber, D. (1999). Formal design of distributed control systems with Lustre. In *Proc. Safecomp'99*, volume 1698 of *Lecture Notes in Computer Science.* Springer Verlag.

[4] Henzinger, T. A., Horowitz, B., and Kirsch, Ch. M. (2003). Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91:84–99.

[5] Kopetz, H. (1997). *Real-Time Systems Design Principles for Distributed Embedded Applications.* Kluwer.

[6] Scaife, N. and Caspi, P. (2004). Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In Pushner, P., editor, *Euromicro Conference on Real-Time Systems, ECRTS04.*

[7] Traverse, P., Lacaze, I., and Souyris, J. (2004). Airbus fly-by-wire: A total approach to dependability. In *IFIP World Congress, Toulouse.* IFIP.

[8] Wensley, J.H., Lamport, L., Goldberg, J., Green, M.W., Lewitt, K.N., Melliar-Smith, P.M., Shostak, R.E, and Weinstock, Ch.B. (1978). SIFT: Design and

analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255.