Formal methods for analysis of heterogeneous models of embedded systems

Simin Nadjm-Tehrani
Dept. of Computer & Information Science, Linköping University,
S-581 83 Linköping, Sweden, e-mail: simin@ida.liu.se
tel: +46-13-28 24 11, fax: +46-13-14 22 33

1 Introduction

Technological development in micro-electronics have made digital control an indispensable component in all engineering systems. The rapid pace of development and the demands on modern systems in terms of novel functions and shorter development cycles has led to many challenges in system design and verification. The down side of improved functionality is the unmanaged complexity: never have we had systems built with so many different disciplines simultaneously at work - each with their own collection of conceptual and concrete tools.

To manage complexity in this setting it is essential to recognise and accommodate the diversities as early as they arise. For most application domains this results in a multi-paradigm development process, and is most visible in the design modelling stage. In this paper we discuss how mathematical modelling and analysis of system properties is affected by having several disciplines at work. We show that soundness in design models can be obtained both through static analysis based on properties defined for a meta-model, and through formal verification of an instance of a model – the latter being defined in terms of conformance to a requirements specification.

2 What is meant by multi-paradigm?

A major dividing line inevitably exists between the digital system and the non-digital environment in which it is embedded in. During the nineties a large volume of work has arisen from attempts to bridge the gap between traditional engineering models in mechanics, hydraulics, and electrical circuits on the one hand, and digital systems on the other. The first group are unified at the mathematical modelling phase in the sense that continuous differential and algebraic equations (DAE) provides them with a common mathematical fundament. In contrast, models for digital hardware and software span a large spectrum based on logic, automata, algebra and graph theory in the area of theoretical computer science, roughly classified as discrete-systems modelling. The work in the nineties has led to an instance of multi-paradigm modelling called hybrid systems, managing the complexities arising from the *discrete-continuous* dichotomy.

There have also been attempts to unify traditional engineering modelling prior to the mathematical modelling phase using DAEs. In this paper we are not concerned with how the continuous mathematical models have been derived. The interested reader is referred to the literature on energy-based physical modelling with the language of bond graphs, for example [21] and references therein. Also, considering simulations of traditional engineering systems, there has been a marked interest in common model management environments based on object-orientation [12, 11].

In this paper we focus more on digital hardware and software. These in turn contribute to a number of additional dichotomies at the mathematical formalisation stage. The notion of digital (for both hardware-like software and electronics), is used here to emphasise the fact that a design can be realised in terms of either software or hardware at a later design stage, after passing initial tests at a high level design stage.

Since digital systems are easily extended and made more complex, the need for concepts of hierarchy and information sharing between subsystems arises. To mathematically characterise these, a global notion of computation step is considered. Thus, the dichotomy between *synchronous and asynchronous* models appears. Also, to model the behaviour of a digital system in response to changes in inputs (as a so-called reactive system) can be described in an *event-triggered or time-triggered* fashion respectively.

In addition, the application domains bring a rich flora of modelling preferences to digital systems. A major such division is between the control-oriented applications (where the complexity arises due to massive numbers of control locations in a computation) and data-oriented applications (where there is much structure in the data on which a large number of operations can

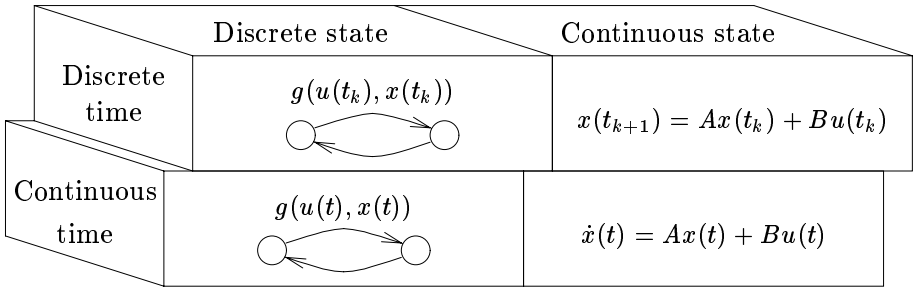

Figure 1: Continuous and Discrete (state and time)

be performed in few control locations). Many applications, of course, combine these two types of computations leading to the need for a combination of *state-based* and data-flow modelling.

Now, no matter how the individual subsystems are modelled and analysed on their own, eventually the composed system will be subject to analysis. One of the main reasons for developing hybrid continuous-discrete models, for example, is to ascertain that the closed loop model consisting of a digital system in a physical environment exhibits desired behaviours only. Thus, a very natural way to model an embedded system is by including elements of the continuous state and the discrete state in the same model. Note the distinction between discrete states and treating the notion of time in a discrete manner. Figure 2 clearly shows the four resulting types of models.

3 Early design modelling

Many embedded systems are used in safety-critical applications. Several others are used in such volumes that discovering a design flaw after the production stage is considered as a major failure. Just in the same way that bridges and buildings are never made without a mathematical model and calculations to ensure a certain degree of confidence before major investments are made, today's complex computer-based systems benefit from a thorough analysis of a mathematical model prior to implementation. It has in fact been shown that many major failures have their roots in misconceptions or omissions in the early design stages of the system.

Formal methods are promoted in order to assist discovering errors at an early design stage. These methods are however geared towards a particular modelling language or a tool, suiting a particular style of implementation. Multi-paradigm development of systems implies that there are isolated islands of captured knowledge, no uniform analysis techniques (or tools) at system level, and a lot of integration testing to compensate for this. A major challenge is therefore to combine existing analysis techniques from various paradigms and to devise a coherent verification methodology for multi-paradigm systems. In the rest of the paper we describe how some of the above dichotomies can be reconciled in the same system using appropriate meta-models. That is, how can systems composed of continuous/discrete, synchronous/asynchronous, state-based/data-flow, event-triggered/time-triggered components be methodically developed based on well-defined underlying semantics? In particular, which aspects of the analysis benefit from the existing capabilities of each paradigm? We give examples of multi-paradigm system verification techniques, illustrating with an industrial example.

4 Merging dichotomies

4.1 State-based vs. data-flow

The family of formal languages known as synchronous languages have shown that they are simple enough to appeal to the engineering community and expressive enough to model non-trivial applications in embedded control. LUSTRE and SIGNAL [14, 13] have a data-flow style (declarative) whereas ESTEREL and STATECHARTS are considered as state-based (imperative) [15, 5]. Each language comes with a bunch of analysis techniques and well-developed toolboxes. One of the major benefits of SIGNAL, LUSTRE and ESTEREL is the clearly documented formal semantics which acts as a description of a meta-model. The clock calculus in LUSTRE and SIGNAL and the constructive semantics of ESTEREL, for example, can be used for static checking of desired properties of an instance (an application model) based on formal semantics of the languages and defined correctness criteria. Major such properties are determinism in a controller and causal consistency at every computation (macro) step [6, 23]. The Statemate tool based on STATECHARTS checks type-coherence of the variables in a model and performs some simple consistency checks.

These tools are finding their ways into modelling the digital parts of several embedded applications,

e.g. energy and power systems and digital signal processing (SIGNAL), electronic circuit design and aerospace (ESTEREL), rail transportation and aerospace (LUSTRE). These tools also provide efficient automatic code generation. So, once the design is "sanity checked" at compilation stage it can be subjected to further formal verification and code optimisation, eventually leading to automatically generated controller code (in C, Ada, or design model in VHDL).

STATECHARTS has had its original popularity in the aerospace sector, but it is gaining popularity for general embedded system design due to inclusion in the UML family of languages [9]. The tool Rhapsody, though no longer in the framework of synchronous languages, is a valuable tool for modelling object-oriented distributed systems.

All of the above-mentioned tools, however, have so far been applied on an individual basis in the respective applications. Considering the growing needs of multi-paradigm modelling, two recent European projects have been exploring the combination potentials of these tools - SACRES for combining SIGNAL and STATECHARTS, and SYRF for combination of SIGNAL, LUSTRE and ESTEREL. The work in SACRES has resulted in relating synchrony with asynchrony [4], and the conditions under which these paradigms can be combined. The work in SYRF has resulted in cross-compilation tools for LUSTRE, SIGNAL, and ESTEREL (loose integration) [22, 17], an environment for multi-paradigm modelling (tight integration) [1], and code distribution for digital systems [8].

4.2 Event-triggered vs. time-triggered

As mentioned above, each member of the synchronous family has been extensively used for design of digital systems. A recent activity has been to combine analysis of continuous systems (as modelled in MATLAB SIMULINK) with the meta-model verification and efficient code generation capabilities of the SIGNAL environment. This is one approach in a series of attempts at the problem of analysis of hybrid systems - an approach we resort to for the case where the plant in itself has abrupt structural changes, and its modelling benefits from a study of simulation runs. Alternatively, we use the simulation environment for the study of behaviours in a closed loop system with a non-linear plant.

To this end, a co-simulation environment has been developed whereby automatically generated C-processes from each tool can be run in a pseudo-parallel fashion [26]. Thus, SIGNAL being a powerful tool for development of hierarchical controllers with complex structures is used for the discrete parts, while MATLAB capabilities for generating simulation models of a continuous plant are used for that purpose. The development of the co-simulation environment followed an analysis of the communication mechanism between the two subsystems. In consequence, it was decided that a protocol based on event-triggering of each (discrete) control component should be combined with the time-triggering of the various plant sub-components during simulation [25].

In recent years MATLAB has been extended with a modelling facility for describing a discrete controller (STATEFLOW - with a syntax reminiscent of STATECHARTS). However, the underlying computation mechanism for simulation of the discrete part of a model is the same as the continuous part of the model. That is, all signals are defined over continuous time and the simulation is time-triggered based on the lowest sample period. In other words, in our SIGNAL-SIMULINK co-simulator one can combine models from the top left part of Figure 2 with models from the bottom right part (where there is no regular metric distance between subsequent steps in the discrete model), whereas the SIMULINK STATEFLOW models combine models from the bottom row of the table. Another difference is that meta-model analysis, formal verification, and code optimisation based on the discrete clock calculus are features present in our multi-paradigm approach and absent in the current MATLAB implementation.

4.3 Synchronous vs. asynchronous

As it was mentioned above, not all applications can naturally be modelled as a globally synchronous system. A recent development has been to relate the notions of synchrony and asynchrony in the context of data-flow languages (in particular SIGNAL) [4, 3]. This work introduces the theoretical notions which can be used to characterise an asynchronous network of locally synchronous nodes, and compositionality properties as a meta-model property in this context. Similar ideas are developed in the context of imperative languages where it is shown how constructively checked ESTEREL can be used as an input language to the POLIS environment, compiling into co-design finite state machines communicating over one-place buffers [7].

5 Mathematical analysis: continuous/discrete

Recent years have seen the extension of application of formal methods to models with both continuous and discrete elements. A typical goal of verification is to show that an invariance holds over a model. In particular, a bad property does not hold in any reachable state of a system. Since digital controllers are increasingly complex with mode changes and multiple inputs and outputs, and the goal of the controller is typically to avoid a bad state in the physical environment, the traditional methods for proving the invariance are not applicable (neither the computer science methods for

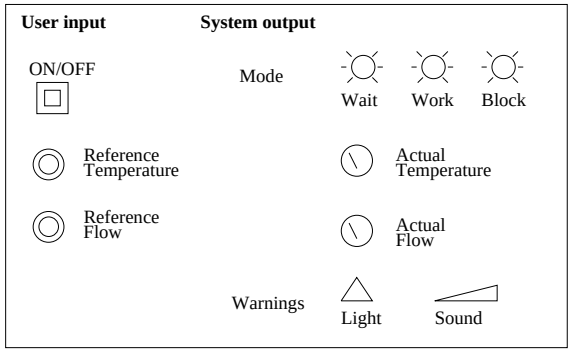

Figure 2: The external interface to the system.

proving properties of discrete systems, nor control theory methods for analysis of continuous systems).

Several techniques for dealing this inherently difficult problem have been proposed, see e.g. [2, 16, 10]. We have studied the specific instance of the problem where the digital controller is modelled by a synchronous program and the controlled environment is modelled by DAEs [19]. We have attempted two approaches to verification: compositional verification and one-shot verification. In each case we provided systematic transformations to one or more parts of the model, arriving at instances of the model which are formally analysable.

5.1 Compositional verification

In this approach we have automatically translated models of the controller in LUSTRE or STATECHARTS to a logical representation analysable by the first order theorem prover NP-Tools. This tool is based on the Stålmarck method [24] and deals with propositional logic and integer arithmetic. Proofs of a property R in the closed loop system is then performed by finding sub-properties $R_1, \ldots, R_n$ such that $\bigwedge R_i \rightarrow R$. Each R_i is then locally proved in the controller by theorem proving, in the plant by continuous analysis, or by further refinement into a new conjunction of sub-properties [18].

5.2 One-shot Verification

In this approach both the plant and controller are represented in the same verification environment (in the same language), and the properties of the system are proved directly in the closed loop model. This is of course dependent on abstractions of the plant model in order to represent it in the same environment – the abstraction being geared towards particular properties of the system [20].

In particular, transformations of a non-linear plant model to a piece-wise linear model leads to an abstraction as a mode-automaton [17] in which a set of difference equations (specified in LUSTRE) are associated with each mode. These models can be translated to flat LUSTRE which in turn can be translated to the input format of NP-Tools via the tool Lucifer (see work-package 3 in [22]).

5.3 Example application

Here we briefly explain an application on which several of the above modelling and verification paradigms were studied. It consists of a climatic chamber with a heater and a fan. The multi-mode control and monitoring case study provided by the industrial partners (Saab AB) exhibited the same types of problems which appear in aircraft air control systems, including undesired mode changes and fluctuations in the heat and flow levels. The external interface is depicted in Figure 2 showing three of the four modes of the system (idle, normal "work", and emergency "block" mode).

Analysis of the textual requirements document from Saab led to identification of the following overall goals for the controller.

- Keeping the reference values constant,
  - the work light shall be lit within a time bound from the start of the system, and
  - the system shall be stable in the work mode.
- Chamber temperature never exceeds a given limit.
- Whenever the reference values are (re)set, the system will (re)stablise within a time bound or warnings are issued.

Proving bounded response properties of a multi-mode synchronous controller relies on two factors:

1. How many "steps" it takes to reach a particular discrete state.
2. How variations in the duration of the step affects the real-time response.

Of course, the first question can not be answered by considering the controller alone. The environment (plant) behaviour is a major part of that. Here we describe the transformations needed before theorem proving can be applied on the closed loop system for confirming the answer to the first question. This is done on "unfolded" models of the system, which means that the transition function for the closed loop system has to be specified in terms of the language of a theorem prover. In the case of NP-Tools, this is propositional logic and integer arithmetic.

The second question is typically assumed to have a well-defined answer. The software engineer assumes

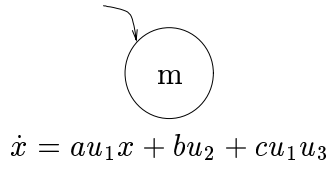

Figure 3: Continuous plant model.

that the 'right' period will be delivered by the control engineer. In the Saab case study we see that this is not necessarily to be taken for granted (for details see [18]). Also, the physical modelling performed in the early stages of the case study was used for determining a reasonable step size, in turn affecting the number of 'unfoldings' mentioned above. This model is shown in Figure 3, where x is the chamber temperature, u_1 the airflow as an input to the temperature equation, u_2 the controller signal based on voltage, and u_3 the temperature of incoming air.

This DAE model does not directly lend itself for plugging into the closed loop model in the theorem prover. First, the plant model is "simplified" by restricting *s*ome inputs to piece-wise constant signals – replacing the non-linear continuous dynamics with discrete modes and linear dynamics in each mode. Then, using the knowledge that the remaining input signals are control signals issued by a synchronous (and sampled) controller, each linear differential equation in each mode is transformed to difference equations (LUSTRE programs with real variables).

Finally, using the same scheme applied for compilation of mode-automata till flat LUSTRE, we obtain a multi-mode LUSTRE model of the plant. This model is now equivalent to the original model modulo restrictions on (physical) input signals. One last step is to approximate the variables in the LUSTRE program from reals to integer. This scheme is depicted in Figure 4.

6 Summary and future works

The transformation scheme presented here has been successfully applied to the climatic chamber case resulting in some improvements in the translators and compilers. Also, a multi-paradigm model was obtained based on LUSTRE SIGNAL and ESTEREL and a further distribution of controller has been obtained within the SIGNAL environment [8]. There is however more work to do in improving the verification tools and techniques, and the combined discrete-continuous simulation tool is a step towards combining some verification possibilities on the controller, followed by co-simulation in the closed loop system.

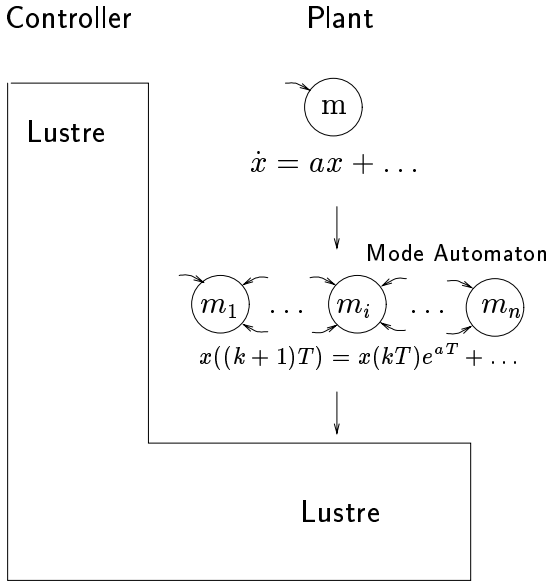

Figure 4: Transformation scheme for obtaining a closed loop model in LUSTRE.

References

[1] A. Poigné and M. Morley and O. Maffeïs and L. Holenderski. The Synchronous Approach to Designing Reactive Systems . *Formal Methods in System Design*, 12(2):163–187, March 1998.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Journal of Theoretical Computer Science*, 138:3–34, 1995.

[3] A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in Dataflow Synchronous Languages: Specification and Distributed Code Generation. *Information and Computation.* To appear.

[4] A. Benveniste, B. Caillaud, and P. Le Guernic. From Synchrony to Asynchrony. In J.C.M. Baeten and S. Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 162–177. Springer Verlag, 1999.

[5] D. Berry and L. Cosserat. The Esterel synchronous programming language and its mathematical semantics. *Lecture Notes in Computer Science*, 197:389–448, February 1985.

[6] G. Berry. The Constructive Semantics of Pure Esterel. Technical report, Centre de Mathematiques Appliquees, 1999. Draft book, available from `www.esterel.org`.

[7] G. Berry and E. M. Sentovich. An implementation of Constructive Synchronous Programs in POLIS. Draft report, appeared on

www-sop.inria.fr/meije/esterel/, November 1998.

[8] L. Besnard, P. Bournai, T. Gautier, N. Halbwachs, S. Nadjm-Tehrani, and A. Ressouche. Design of a Multi-formalism Application and Distribution in a Data-flow Context: An Example. In *Proceedings of the 12th international Symposium on Languages for Intentional programming, Athens, June 1999*, pages 8–30. World Scientific, June 1999.

[9] G. Booch, J. Rumbaugh, and I. Jakobsson. *The unified Modelling Language User Guide.* Addison-Wesley, 1999.

[10] Z. Chaochen, A. P. Ravn, and M. R. Hansen. An Extended Duration Calculus for Hybrid Real-Time Systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Proc. Workshop on Theory of Hybrid Systems, October 1992, LNCS 736*, pages 36–59, Lyngby, Denmark, 1993. Springer Verlag.

[11] The Modelica design group. Modelica, Language Design for Multi-Domain Modeling. See http://www.Modelica.org, 1999.

[12] H. Elmqvist. *Dymola User's Manual.* Dynasim AB, Lund, 1994.

[13] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, September 1991.

[14] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[15] D. Harel. STATECHARTS: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[16] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O Automata. In R. Alur, T.A. Henzinger, and E. Sontag, editors, *Proc. of the DIMACS International Workshop on Verification and Control of Hybrid Systems, LNCS 1066*, pages 496–510. Springer Verlag, 1996.

[17] F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. In *Programming Languages and Systems, Proceedings of the 7th European Symposium On Programming, Held as part of ETAPS'98, Lisbon, Portugal, LNCS 1381.* Springer verlag, March 1998.

[18] S. Nadjm-Tehrani. Integration of Analog and Discrete Synchronous Design. In *Hybrid Systems: Computation and Control, Proceedings of the second international workshop, March 1999, LNCS 1569*, pages 193–208. Springer Verlag, March 1999.

[19] S. Nadjm-Tehrani. Time-Deterministic Hybrid Transition Systems. In *Hybrid Systems V, Proceedings of the fifth international workshop on hybrid systems, September 1997, LNCS 1567*, pages 238–250. Springer Verlag, 1999.

[20] S. Nadjm-Tehrani and O. Åkerlund. Combining Theorem Proving and Continuous Models in Synchronous Design. In *Proceedings of the World Congress on Formal Methods, Volume II, LNCS 1709*, pages 1384–1399. Springer Verlag, September 1999.

[21] S. Nadjm-Tehrani and J-E. Strömberg. Verification of Dynamic Properties in an Aerospace application. *Formal Methods in System Design*, 14(2):135–169, March 1999.

[22] The SYRF Project. Deliverables for Work packages 1 to 7. Available from www-verimag.imag.fr//SYNCHRONE/SYRF/syrf.html, 1997-99.

[23] T. R. Shiple, G. Berry, and H. Touati. Constructive analysis of cyclic circuits. In *Proc. International Design and Testing Conference*, Paris, 3 1996.

[24] Prover Technology. *NP-Tools v 2.3 User's Guide.* Stockholm, Sweden. Contact: http://www.prover.com.

[25] S. Turodet. Signal-Simulink: Hybrid system Co-simulation. Technical report, Dept. of Computer and Information Science, Linköping university, February 2000. available from www.ida.liu.se/labs/eslab/publications/report.shtml.

[26] S. Turodet, S. Nadjm-Tehrani, A. Benveniste, and J.-E. Strömberg. Co-simulation of Hybrid Systems: Signal-Simulink. In *Proceedings of the 6th International Conference on Formal Techniques in Real-time and Fault-tolerant Systems, LNCS.* Springer Verlag, September 2000. To appear.