On Verification Modelling of Embedded Systems

Ed Brinksma and Angelika Mader *

Department of Computer Science, University of Twente
PO Box 217, 7500 AE Enschede, Netherlands
{brinksma,mader}@cs.utwente.nl

Abstract. Computer-aided verification of embedded systems hinges on the availability of good verification models of the systems at hand. Because of the combinatorial complexities that are inherent in any process of verification, such models generally are only abstractions of the full design model or system specification. As they must both be small enough to be effectively verifiable and preserve the properties under verification, the development of verification models usually requires the experience, intuition and creativity of an expert. We argue that there is a great need for systematic methods for the construction of verification models to move on, and leave the current stage that can be characterised as that of "model hacking". The ad-hoc construction of verification models obscures the relationship between models and the systems that they represent, and undermines the reliability and relevance of the verification results that are obtained. We propose some ingredients for a solution to this problem.

1 What is the problem?

Many embedded systems are subject to critical applications, massive replication and/or widespread distribution. The quality of their design, therefore, is an issue with major societal, industrial, and economic implications. Systematic methods to design and analyse such systems are consequently of great importance. In this paper we focus on a number of methodological aspects of the analysis of embedded systems.

In computer science formal methods research much progress is being made in tool-supported, model-based system analysis, in particular in the areas of model checking, theorem proving and testing. A prerequisite for such analysis is the availability of a formal model of the system. For pure software systems the (semi-)automated extraction of abstract models out of (a specification of) the source code is a possibility in principle. For embedded systems, however, the modelling task is intrinsically more complicated. As interaction with their (physical) environment is an essential ingredient, good models must integrate the relevant aspects of the system hardware, software, and environment.

* This work has been partly funded as part of the IST Ametist project funded by the European Commission (both authors), and the Moms project funded by the Netherlands Organisation for Scientific Research (second author).

The current modelling and verification practice for embedded systems can be characterised by the slogan: "*model hacking precedes model checking*". Constructing a model is typically based on the experience, intuition and creativity of an expert. In an initial phase the model is improved in trial and error fashion: first verification runs show errors in the model, rather than errors in the system. Once the model is considered stable and correct, the subsequent verification runs are considered analyses of the system.

Processes of model construction are mainly described in case studies, but in most cases the applied design decisions and paradigms remain implicit. As a consequence, it is difficult to compare, and assess the quality of the models. Therefore, different analysis results are also difficult to interpret. The ad-hoc construction of verification models obscures the relationship between models and the systems that they represent, and undermines the reliability and relevance of the verification results that are obtained. Moreover, most of the existing case studies are done to show that a given algorithm is faster, or that some tool is applicable to a given problem.

So far, the *method* of modelling has not been a topic of systematic research. Quoting the NASA Guidebook on formal methods [17]: "The observation [that there is a paucity of *method* in formal methods] focuses in particular on the absence of 'defined, ordered steps' and 'guidance' in applying ... methodical elements that have been identified." A first collection of relevant empirical data exists in the form of various case studies consisting of different teams applying different modelling approaches to a common problem, such as the steam boiler case study [14], the RCP-memory case study [8], the VHS batch plant case study [16], and an industrial distributed data base application [9]. What is mostly missing, however, is the extraction of the commonalities and differences of the resulting models, and their comparative evaluation against qualitative criteria such as ease of modelling, quality of analysis, tool support, adaptability, maintainability, etc.

A wealth of material exists on the topic of specification (formalisms) and their application, but this is essentially aimed at the construction of complete models (specifications) of system behaviour, as unambiguous statements of the desired functionality, where the resulting size and complexity are of secondary interest. Our interest, however, is in models for selected properties of interest where simplicity and size are of prime concern to control the combinatorial explosion that results from their analysis. Only such an approach offers hopes for tool-supported analysis.

2 What do we need?

The previous section indicated that we need methods for the construction of verfication models that combine the following properties:

- They are of *limited complexity*, meaning that they can be analysed with the help of computer-aided verification tools;

– They are *faithful*, meaning that they capture accurately the (verification) properties of interest;
– They are *traceable*, meaning that they have a clear and well-documented relation to the actual (physical) system.

It is clear that it is not easy, generally, to satisfy all of these constraints concurrently. The construction of verification models really is a design problem in its own right that may share many characteristics of the design problem of the system itself, but is not identical to it because of its different purpose and complexity constraints. Being a design problem, it generally also involves a strongly creative element that defies automization.

Below, we discuss the above points in some more detail.

2.1 Limited complexity

Modern formal methods research is tightly coupled to the development of analytical software tools such as animators, model checkers, theorem provers, test generators, simulators, Markov chain solvers, etc. Only computer-aided analysis can hope to overcome the combinatorial complexities that are inherent in the study of real systems.

Even then, one of the main obstacles to overcome is the infamous combinatorial explosion problem, causing the size of search spaces, state spaces, proof trees, etc. to grow exponentially in the number of (concurrent) system components. This makes it essential to keep the complexity of verfication models within the range that can be effectively handled by tools.

Of course, the effective limit is growing rapidly itself, due to both exponential growth of hardware speed and memory miniaturisation (Moore's law), and improvements the algorithms and data structures used in the tools. Still, it seems reasonable to assume that effective computer-aided analysis must always be based on models that simplify or abstract from the full functionality of the system at hand.

2.2 Faithfulness

The purpose of verification is to show that a system satisfies (or does not satisfy) various properties. It is obvious that the model we investigate for verification should share the properties of interest with the original system. Therefore, the abstraction steps that are applied in the verification modelling process should preserve the relevant properties. Under most circumstances this is a tall order for two reasons:

1. *We may not know the relevant formal properties*. In fact, finding the right formal properties to verify is often as much a part of the verification design problem as finding the right model. In practice, the design of the properties

and model go hand in hand. Together with the necessity to keep models small this leads to a collection of different models, each geared for a different (set of) formal property (properties) [5].
2. *We may not know whether our abstractions preserve them.* Showing that our abstractions preserve the intended properties may be as hard as our original verification problem, and the set of abstractions with known preservation properties is usually too small to suffice for practical problems.

A common way out of the second complication is to be satisfied with approximating models that may generate *false negatives* or *false positives*, i.e. report nonexistent errors, or obscure existing errors, respectively. In the first case, spurious errors may be filtered out if they cannot be traced back to the original system, e.g. by counterexample trace analysis in model checking. In the second case, one is reduced to *falsification* or *debugging*, where the presence of errors can be shown, but not their absence.

The insight that all verification is in some sense reduced to debugging, due to the untimately informal relationship between model and real system, is misleading. Although it is true in the Popperian sense of falsification, it should not be interpreted as a reason not to seek the positive identification of model properties. This minimalistic approach is not enough when a model must have certain properties to be relevant at all for the verification task at hand. For example, when deriving schedules by means of model checking [6], one needs to know positively that the relevant aspects of time are represented in the model. The modelling process should therefore guarantee in some way the presence of the properties of interest.

An interesting development in software model checking is that of (semi-)automated model abstraction from code [3, 2, 11], allowing for many approximating verification models to be analysed concurrently. This way of debugging can under circumstances achieve a good error coverage. For embedded systems, however, the modelling task is intrinsically more complicated. As interaction with a (physical) environment is an essential ingredient, good models must integrate the relevant aspects of the system hardware, software, and environment. This cannot be achieved by automated code abstraction. Libraries of succesful modelling fragments coupled to specific system domains may provide a way forward here.

2.3 Traceability

Verifying an erroneous model is both useless and time-consuming. As pointed out above already obtaining faithful models is not an easy task, and cannot be achieved by formal means only. In fact, one of the most important ways of establishing the relevance of a (verification) model is by keeping track of the design and abstraction steps that relate it to the actual system, the choices that were made, and the reasons behind them.

There is not a unique starting point for model derivation. In an a posteriori verification case one could start from a piece of embedded software together with

an engineer's diagram of what the physical part of the embedded system does. It could be a standard or another informal description. In an a prioriverification we might start from a desired behaviour specification. In any case we have to get from a system description that is likely not to be a formal object to a formal model. The preservation of relevant properties is therefore (in many design steps) also not a formal notion. In such cases the only form of evidence that can be given is by insight. Insight requires a transparent representation of the design steps taken. Another advantage of a clear derivation traject is that is easier to check (by others). Transparency makes it easier to detect errors in the modelling process.

A further relevant aspect in this context is the comparability of verification approaches and results. In many research projects different groups work with different tools on the same case studies. The comparison of the results, however, is difficult due to the fact that there are no criteria available to compare the different models. Their design track records provide valuable information that can help to compare and relate them.

3 How do we get there?

Anyone attempting to obtain methods for transparent model construction for embedded system verification along the lines sketched above, is immediately confronted with two facts:

- *domain dependency*: design methods and concepts depend strongly on particularities of the application domain at hand. Moreover, different domains, specifications, system descriptions, and creative elements provide a huge range of problem settings.
- *subjectivity*: transparency is a relative and imprecise notion. What constitutes a clear documentation of a modelling step depends very much on the kind of the step, the problem domain, the level of informality and formality, and certainly also on personal taste. As clear as possible always means to restrict to the information necessary and find a good representation for it, which could be a diagram, an informal explanation, a table a formal mapping, etc. However, diagrams with ambiguous semantics can be as difficult to understand as some formalism that requires too much details.

Under these circumstances, what is needed is a *protocol* for the construction of verification models. It must adopted by the (embedded systems) verification community at large, so that it can be a point of reference for the construction, evaluation, and comparison of verification models. At the same time, it can provide the basis for the collection and organisation of succesful models and modelling patterns as ingredients for a true discipline of verification model engineering.

Even in the inherent absence of a universal approach to model construction, each modelling process should be *guided* by a number of basic considerations. The guidance is obtained by systematically considering the different aspects:

- *scope of modelling*;
- *properties of interest*;
- *modelling patterns*;
- *effective computer-aided analysis*.

We digress shortly on each of these points.

scope of modelling

It is necessary to make explicit what the model includes and what not. What part of the system, assumptions about the environment, operating conditions, etc. The system domain under consideration is also relevant here, as it usually introduces particular concepts an structuring principles that must be referred to. It has turned out te be useful to make *dictionaries* of the domain specific vocabulary used: it helps to agree, e.g., with system experts, on the interpretation of the description, to filter the relevant notions, and to have instrumentation for unambiguous explanations in later modelling steps.

properties of interest

The properties of interest should be stated as explicitly as possible. As indicated earlier, initially we may not know what will be the correct formalisation of these properties and their preservation principles. Nevertheless, their informal statement and intention should be stated as completely and unambiguously as possible. At each modelling step the best possible argumentation should be put forward on why they can be assumed to be preserved. The definitions and arguments should become more precise as formalisation progresses in the course of the model construction.

modelling patterns

Important in any more systematic approach is the identification of characterising structural elements that occur in a system under verification, and allow the sharing of modelling patterns in models that have such elements in common. The concept of solution patterns is central to many branches of engineering; an excellent elaboration of the idea of design patterns in software engineering by Micheal Jackson as *problem frames* can be found in [13].

Characterising structural elements can be representative for a particular application area (e.g. network controllers, chemical process control), but not necessarily so. Very different systems, such as e.g. a steel plant [10] and a multimedia juke

box [15] have been found to share significant similarities, viz. on an abstract level both are comparable transport scheduling problems with a resource bottleneck.

Some examples of characterising structural elements that play a role in in many embedded systems are controller polling policies, scheduling mechanisms, (programming) languages, operating system and network features such as interrupt handling, timers, concurrency control, communication architecture, time granularity, etc. at various levels of abstraction depending on the properties of interest.

effective computer-aided analysis.

Modelling and analysis of realistic systems is impossible without proper tool support. Therefore, it is reasonable to focus on model classes for which efficient tool support is available. As analysis tools are a very active area of research where continuous progress is being made, it is important to be open to new developments. Tool performance is improving dramatically by such devices as better data structures, sophisticated programming techniques, and parallelisation [4] of algorithms. New, more powerful approaches such as guided [7, 1] and parametric model-checking [12, 18] are extending the applicability of tools significantly.

4 Conclusion

We have argued that the construction of verification models for embedded system is a non-trivial task for which there are no easy solutions. It is our feeling that the issues that we have raised deserve more more attention from both the community of researchers and the industrial appliers of formal methods. It is of great interest that the designs of verification models are made available to be able to evaluate, compare, improve, collect and use them in a more systematic way, and leave the current stage of model hacking. To be able to do so a generally agreed protocol for their documentation as transparent and guided design processes is much needed. We have outlined some ingredients for such a protocol.

References

1. R. Alur, S. La Torre, and G. Pappas. Optimal paths in weighted timed automata. In *Proc. of the Fourth International Workshop on Hybrid Systems: Computation and Control*, volume 2034 of *LNCS*. Springer, 2001.
2. T. Ball, S.K. RajamaniJ. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubacj, and H. Zeng. Bandera: Extracting finite-state models from java source code. In *Proceeding 22nd ICSE*, pages 439–448. IEEE Computer Society, 2000.
3. T. Ball and S.K. Rajamani. The SLAM project: debugging system software via static analysis. In *Proceeding 29th POPL*, pages 1–3, 2002.
4. G. Behrmann, T. Hune, and F. W. Vaandrager. Distributed timed model checking - how the search order matters. In *Proceedings CAV'2000*, volume 1855 of *LNCS*. Springer, 2000.

5. E. Brinksma. Verification is experimentation! *Journal of Software Tools for Technology Transfer (STTT)*, 3(2):107–111, 2001.
6. E. Brinksma, A. Mader, and A. Fehnker. Verification and optimization of a PLC control schedule. *International Journal on Software Tools for Technology Transfer*, 4(1):21–53, 2002.
7. K. G. Larsen et al. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proceedings of CAV'2001*, volume 2102 of *LNCS*. Springer, 2001.
8. M. Broy et al., editor. *Formal System Specifications - The RCP-Memory Specification Case Study*, volume 1169 of *LNCS*. Springer, 1996.
9. P. H. Hartel et al. Questions and answers about ten formal methods. In *Proceedings of the 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, pages 179 – 203. ERCIM/CNR, 1999.
10. Ansgar Fehnker. Scheduling a Steel Plant with Timed Automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press, 1999.
11. G. Holzmann and M.H. Smith. Software model checking. In *Proceeding FORTE 1999*, pages 481–497. Kluwer, 1999.
12. T. S. Hune, J. M. T. Romijn, M. I. A. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. In *Proceedings of TACAS'2001*, volume 2031 of *LNCS*, pages 189–203. Springer, 2001.
13. M. Jackson. *Problem Frames*. ACM Press, Addison-Wesley, 2001.
14. H. Langmaack, E. Boerger, and J. R. Abrial, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *LNCS*. Springer, 1996.
15. M. Lijding, P. Jansen, and S. Mullender. Scheduling in hierarchical multimedia archives. Submitted.
16. A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a batch plant - VHS case study 1. *European Journal of Control*, 7(4):416–439, 2001.
17. NASA's Software Program. *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems*. eis.jpl.nasa.gov/quality/Formal_methods/.
18. R. L. Spelberg, R. de Rooij, and H. Toetenel. Experiments with parametric verification of real-time systems. In *Proceedings of the 11th Euromicro Conference on Real Time Systems*, 1999.