# 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. Such models must be much simpler than full design models or specifications to be of practical value, because of the unavoidable combinatorial complexities in the verification of any non-trivial system. Good verification models, therefore, are "lean and mean", and cannot be obtained easily or generated automatically. Current research, however, seems to take the construction of verification models more or less for granted, although their development typically requires a coordinated integration of the experience, intuition and creativity of verification and domain experts. 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.**

## I. What is the problem?

Research on formal methods in computer science has made much progress in tool-supported 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. Although the construction of good models is of central importance to many applications of formal methods, so far it has received little attention by the research community. Indeed, the following list of some common misconceptions shows that the nature and purpose of model construction are generally not very well-understood:

- Modelling is done by using language XYZ.
- Modelling is done by using tool UVW.
- Modelling is a formal activity.
- Being formally precise right from the beginning produces better models.
- A model is always an abstraction of something real.
- Verification can be done by plugging a model checker to the model of the overall system design.

In this paper we address modelling, and most of the above misconceptions, in the context of the verification of embedded system designs. The critical nature of many embedded system applications makes their systematic analysis an important topic of research. The construction of reliable verification models for such systems is a non-trivial task that raises a number of important methodological issues.

For software systems, being more or less formal objects in themselves, models can be extracted out of the (specification of the) source code using abstraction principles, algorithms, and tool-support, which forms a promising area of research [3], [11], [4]. For embedded systems, however, the modelling task is intrinsically more complicated. Here, an essential ingredient is the interaction between the system hardware, software, and environment, and a system model must integrate their relevant aspects, which are often of very different nature.

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 a number of implicit decisions that derive from the state of insight, experience, intuition and creativity of an expert. In an initial phase a 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. There, in most cases, the focus is more on algorithmic and tool aspects than on modelling, i.e. the goal of a case study is to show that a given algorithm is faster, or that some tool is applicable to a particular problem or problem class. Therefore, design decisions that were applied and paradigms leading to the model often remain implicit.

As a consequence, it is difficult to assess the quality of a model, or to compare different models. Also, different analysis results, produced by different tools in different frameworks, are 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.

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-

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).

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.

## II. What do we need?

The verification of inadequate models is both useless and time-consuming. The problem statement above indicates that we need methods to provide a systematic construction of verification models that combine the following features:

- 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 system.

It is clear that it may not be easy to satisfy all of these constraints at the same time. In fact, the construction of verification models shares many characteristics with the design of the system itself, such as aspects of traceability and faithfulness. Also, being a design problem, it involves a creative element that cannot be automated easily. The complexity constraints and its special purpose, however, really make it a design problem in its own right.

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

### A. 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 verification models within the range that can be effectively handled by tools. Of course, this effective range itself is growing rapidly, due to both exponential growth of hardware speed and memory availability (Moore's law), and improvements of the algorithms and data structures used in the tools. Still, it seems reasonable to assume that also in the future effective computer-aided analysis must be based on models that simplify or abstract from the full functionality of the system at hand.

A practical consequence of the above is that verification modelling is strongly guided by the requirements and capacity of the available tools. This makes it distinct from most other modelling activities, which usually focus on some form of completeness. It also presents the additional challenge of developing models that meet tool requirements, while at the same time keeping them sufficiently generic to ensure their tool-independent meaning and understandability. This is also important to facilitate the portability of models between different tools and tool versions.

### B. Faithfulness

The purpose of verification is to show that a system satisfies (or does not satisfy) a selected set of properties. It is obvious that the verification model we investigate should share the properties of interest with the original system. Therefore, the design 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 simple and tool-compatible this leads to a collection of different models, each geared for different (subsets of) formal properties [6].
2) *We may not know whether our design decisions preserve them*. Showing that our design steps preserve the intended properties may be as hard as our original verification problem. Also, the set of transformations and abstractions with known preservation properties is usually too small to suffice for practical problems.

Approximating models provide a pragmatic way forward to obtain simpler models when the preservation of all relevant properties cannot be maintained. They approximate the intended system in the sense that they may generate *false negatives* or *false positives*, i.e. report nonexistent errors, or hide existing errors, respectively. In the first case, artificial 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. Working with a large set of different approximating models, each exposing a different set of potential errors, can be an interesting technique to improve error (or property) coverage.

An interesting development in software model checking already mentioned in the first section is (semi-)automated model abstraction from code [3], [4], [11], allowing for many approximating verification models to be analysed concurrently. This way of debugging can under circumstances achieve rather 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 successful modelling fragments coupled to specific system domains may provide a way forward here.

### C. Traceability

As pointed out above, obtaining good models is not an easy task, and cannot be achieved by formal means only. The proof of adequacy of a (verification) model ultimately relies on insight, and one important way of establishing its relevance 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. The aim should be to make model construction *transparent*, so that models may be more easily understood and checked by others, making their quality measurable in (at least) an informal sense.

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 priori verification 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. In such design steps, therefore, the preservation of relevant properties is not a formal notion either, and the only form of evidence that can be given is by insight. Communication of insight requires a transparent representation of the design steps taken. Such transparency, moreover, makes it easier (for others) to check the sequence of design decisions of the model construction, and to detect possible errors in the modelling process.

Another relevant point is that traceability facilitates the interpretation and relation of verification results across different models. Often different groups work with different tools on the same case studies. Differences between the underlying models complicate the comparison of their analytical results. Traceable model construction helps us characterize and compare the models and results on the basis of the design decisions that were used in their construction.

## III. 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 a number of facts:

- Modelling is a creative process. Often, part of the solution lies in the choice of the "right" model. Finding a good level of representation is based on both experience and intuition. Moreover, a model is not necessarily just an abstraction of the real system. It can be intentionally "wrong" (e.g. by approximation or simplification), if the verification results can be related in a well-understood manner to the original system. Also, models can contain additional structure that is not part of the system, but helps to find solutions.
- Design methods and concepts are strongly domain dependent. Different domains tend to have their own ways of problem description, representation, and abstraction. A modelling method should not offer one specific language or tool as a straightjacket for all possible problem domains.
- Transparency is a subjective and imprecise notion. Certainly, it should not be confused with the explicit representation of every detail of a design. Abundance of information can easily obscure the overall picture and become counterproductive. Complete formalization of a design can be an instance of this: obvious facts may require lengthy formalizations in which crucial facts may be lost.

  What constitutes a clear documentation of a modelling step, therefore, depends very much on the kind of the step, the problem domain, the right mix of informality and formality, and certainly also on personal taste. A proper representation of design decisions illuminates their essential ingredients, while keeping the obvious obvious. Semi-formal notations, such as diagrams and tables can be very helpful in this respect. It should be emphasized that non-formal representations can be quite precise when used with care. Of course, informality that produces ambiguity in the context of its use must be avoided at all cost.

In the light of these facts, the best to offer for the construction of verification models is a kind of *protocol*. It must be 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 organization of successful models and modelling patterns as ingredients for a true discipline of verification model engineering.

Even if a universal approach to model construction cannot exist, each modelling process is *guided* by a number of basic concerns. Each of these provides a different view of the modelling process, and, at the same time, suggests ingredients for design documentation. A simple initial checklist of such (interdependent) concerns is:

- *the scope of the model*;
- *the properties of interest*;
- *the available tools*;
- *the available modelling patterns*.

We digress shortly on each of these points.

*Scope of the model*

This concerns the collection of basic assumptions about the system (parts) under investigation. Specifically, information about the following aspects must be available:

- What part of the complete system is modelled. This already requires an initial understanding of the system structure and its decomposition.
- Essential assumptions made about the environment, domain knowledge, operational parts of the system, operating conditions, etc. It has turned out to 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 identify the relevant notions, and to have a framework for unambiguous explanations in later modelling steps.

*Properties of interest*

As indicated earlier, initially we may not know the correct formalisation of the properties to be verified and their preservation principles. Nevertheless, their informal statement and intention should be as complete and unambiguous 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 grow more precise as formalization progresses in the course of the model construction.

*Available tools*

Verification modelling makes sense only if it is focussed on model classes for which effective tool support is (or will become) available. Such classes can entail specific requirements that must be fulfilled by the models and/or properties under consideration interms of size, complexity, types, etc.

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 [5] of algorithms. New, more powerful approaches such as guided [7], [1] and parametric model-checking [12], [19] are extending the applicability of tools significantly.

*Available 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 Michael Jackson as *problem frames* can be found in [13], "*The idea of problem frames is to classify and analyze the common types of simple problems so that you can hope to recognise them when you meet them, to anticipate their concerns and difficulties, and to apply familiar and effective techniques to their solutions.*"

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. The AMETIST project [2] tries to identify, develop, and exploit a generic theory and related tools based on timed automata models to resource scheduling and planning problems in such diverse areas as automotive collision detection and avoidance, lacquer production planning, smart card personalisation, memory management, etc.

Other examples of central concepts that play a role in many embedded systems and provide a basis for modelling patterns are time models (locality, granularity), polling policies, operating system and network features, such as interrupt handling, timers, concurrency control, communication architecture, etc. Patterns can exist at different levels of abstraction depending on the properties of interest. Also related to modelling patterns is the particular style of modelling that is chosen, such as e.g. architectural, functional or process-based models. The choice of style has a profound influence on the kind of modelling primitives that can be used.

The use of modelling patterns can also be exploited by verification algorithms: by taking specific structures into account efficient, domain-specific versions of verification tools can be built. An example of this is provided by the Bogor model checking framework [18].

## IV. Conclusion

The construction of a good verification model of an embedded system is not an easy task. It must bridge the gap between informality and formality, integrate knowledge about the analysis methods and the application domain, and be sufficiently simple to allow for tool-assisted analysis, yet be rich enough to represent crucial properties faithfully. Especially the last dilemma makes verification modelling a discipline in its own right.

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.

**Acknowledgement:** The authors would like to thank Roel Wieringa for inspiring discussions on the topic.

## V. 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] AMETIST - Advanced Methods for Timed Systems, Esprit-LTR project IST-2001-35304. *http://ametist.cs.utwente.nl*.

[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] T. Ball, S.K. Rajamani, J. 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.

[5] 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.

[6] E. Brinksma. Verification is experimentation! *Journal of Software Tools for Technology Transfer (STTT)*, 3(2):107–111, 2001.

[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] John Hatcliff Robby, Matthew B. Dwyer. Bogor: An extensible and highly-modular model checking framework. In *Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, 2003.

[19] 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.