A Taxonomy of Modelling Decisions for Embedded Systems Verification

Angelika Mader*, Hanno Wupper, Mieke Boon**, and Jelena Marincic***

[1] Department of Computer Science, University of Twente, The Netherlands, a.h.mader@ewi.utwente.nl
[2] Computing Science Department. Radboud University Nijmegen. The Netherlands, wupper@cs.ru.nl
[3] Department of Philosophy, University of Twente, The Netherlands, m.boon@gw.utwente.nl
[4] Department of Computer Science, University of Twente, The Netherlands, j.marincic@ewi.utwente.nl

Abstract. During model construction a number of decisions has to be taken. Often, decisions remain implicit, which has consequences for the correct interpretation of the model, and provides a source of miscommunication between different stakeholders and domain experts. We present a taxonomy of decisions guiding the modelling process and helping to distinguish, document and order the choices and assumptions involved.
We focus on modelling of embedded systems, the purpose of the models we construct is formal verification. As a consequence we model both, control software and the controlled environment. Moreover, we aim at models in a formal representation, suitable for, e.g., model checking.
The explicitness of modelling decisions allows to assess the quality of models and to clarify how meaningful verification results are.
The taxonomy is a result of conceptual analysis and not bound to certain languages, methods, and tools. It can be used as complement for existing modelling methods.

1 Introduction

Modelling subsumes a large spectrum of different activities, as representation, formalisation, transformation, or integration. Accordingly, there are modelling methods, languages and tools that support the different activities. Our modelling efforts focus on the earliest point in this spectrum, when no descriptions and formalisations are available, and tools can not yet give support.

Each model has a purpose, it is not constructed to merely reflect reality, but to serve some function. We have design models, verification models, prototype models, etc. The

* supported by NWO project nr. 632.001.202, Methods for modelling embedded systems
** supported by NWO project 016.038.321, Using science in technology: towards a philosophy of the engineering sciences
*** Supported by NWO, project 600 065 120 241420, Modelling Control Aspects of Embedded Systems

process of modelling, as we consider it, is to collect the knowledge necessary to satisfy this purpose, and the representation of this knowledge is the model.

In the process of model construction a number of decisions has to be taken. Often, decisions remain implicit, which has consequences for the correct interpretation of the model, and which provides a source of miscommunication between different stakeholders and/or domain experts.

Our goal is to make as much of these decisions explicit. To this end we present a taxonomy of decisions which can guide the modelling process and help to distinguish, document and order the choices and assumptions involved. In the practical case, the taxonomy can be used as a checklist of questions to answer. Such a list has the following advantages:

1. It helps to recognize what decisions at all have to be taken, and gives the chance to explicitely address them.
2. It makes the modelling process also more efficient and structured.
3. It helps to assess the quality of a model, check whith stakeholders and domain experts whether the adequate decisions are taken, and whether there is consensus about them.

We focus on modelling of embedded systems, the purpose of the models we construct is formal verification. A consequence of the first is that we do not only model control software, but also the controlled environment. A consequence of the latter is that we aim at models in a formal representation, suited for model checking or theorem proving. Several aspects of our taxonomy are related to this choice. The majority of questions is more general, and can be of use for modelling of different systems or different purposes.

In the following we focus on the role of modelling in the verification process. The *formal* part of formal verification, depicted in the upper half of the diagram in figure 1, establishes by means of computer analysis that a mathematical object that is considered a *model* of an artefact has certain properties. In the end, however, we are not interested in the properties of a model, but in the correct behaviour of artefacts in physical reality, as depicted in the lower half.

Modelling bridges between the physical world and the mathematical world. Because of this bridge function it cannot live in the mathematical world only - some steps of modelling are inherently non-formal. By making the non-formal steps of modelling explicit and structured, we make them more accessible and controllable.

When answering the question to what extent the result of verification is meaningful for the behaviour of the artefact a number of different positions can be taken [4, 15, 13], briefly: The rationalists' position says, that formal verification cannot guarantee that an artefact has a certain behaviour in physical reality, it can only find faults in the model that can possibly be traced back to the artefact. This is according to a sharp distinction between the two phases in scientific research, the *discovery* and the *justification*. Only the justification of knowledge is considered to be formalisable, and is therefore the objective and rational part. The *discovery of a model*, on the other hand, is left to the domain of psychology. In this sense formal verification can only be used as a debugging technique. Discovery of the verification model, in our context, is an act of unpredictable creativity.

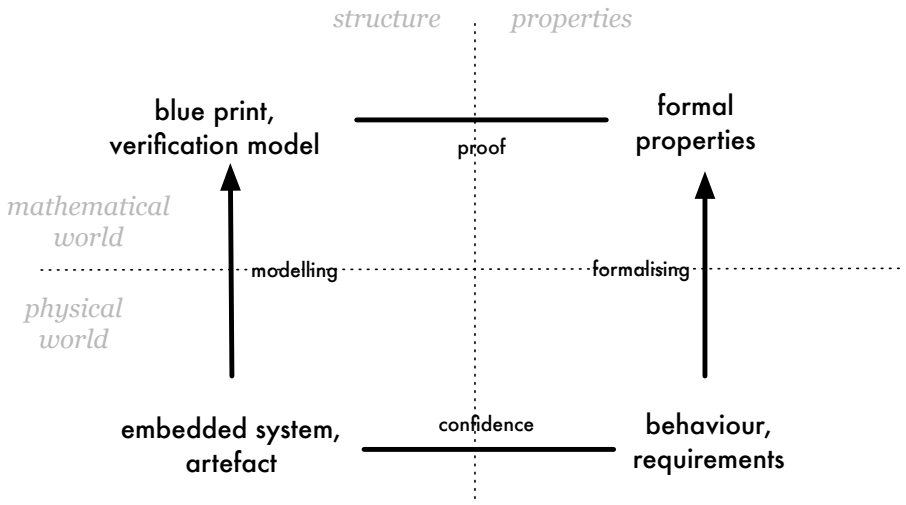

Fig. 1. Objects relevant in the context of verification

The engineering position is: there are models of different quality, and the meaningfulness of the verification result depends the quality of the model that went into verification. Models are not *discovered*, but constructed in a systematic way following a discipline of thinking, proven standard-arguments and steps. In most cases, modelling is not subject to *radical design*, where something entirely new is created, but we apply the known, adequate abstractions, decompositions, idealisations and patterns ([18, 2, 6]).

The meta-modellers position is that after several artefacts of a certain class have successfully been modelled, the modelling process for this class can itself be formalised as a *meta*-model.

In our work, we take the engineering point of view. Our argument against the rationalists' position is that the purely creative interpretation of the design process as described by the context of discovery does not cover the engineering aspects in the modelling process. Our choice is against the meta-modellers' position, because a meta-model is already based on a *model* of reality. The transition from a fragment of physical reality to a formal model is an inherently non-formal process. Therefore, even if a modelling process can be meta modelled a posteriori, the construction of a (new) model for a (new) piece of reality (a priori) is not a formal activity, i.e. it is an activity that contains both, non-formal and formal aspects.

We consequently investigate what constitutes the quality of models and what is the systematic part of the modelling process. In the paper presented here we focus on the decisions taken in the modelling process, which often are taken sub-consciously and seldom are documented.

We suggest a taxonomy of design decisions, presented in the form of seven questions about the model under discussion. The taxonomy is a result of conceptual analysis

and not bound to certain languages, methods, and tools. It can be used as complement for existing modelling methods.

The structure of the paper is as follows. In section 2 we propose a taxonomy of decisions for model construction. Related work is contained in section 3. We conclude with section 4.

2 A taxonomy of decisions for model construction

In the literature on modelling methods we find different activities described as part of the modelling process. Typically, it includes drawing of certain diagrams, writing down specifications, performing transformations. Assumptions about what is given in the beginning of the modelling process are also diverse. Sometimes a complete specification of the requirements is expected, sometimes the specification of the requirement is a result from the modelling process.

We assume, that in the beginning of the modelling process the plant and possibly also the control are given. There is no formal description that can be directly used for verification purposes. Knowledge about plant and control is distributed among different domain experts, and the modeller does not have the complete knowledge about the system. Moreover, also the verification property may exist only informally.

The purpose of the modelling process is therefore to gather the necessary knowledge about the plant, to formalise it, to document the assumptions made about the non-formalised aspects, and to get a formalisation of the verification property.

We suggest a taxonomy of design decisions for model construction. It can be used as guideline or checklist for both, the modelling process as also the justification. In the following we present the taxonomy in the form of questions that will be discussed and illustrated by small examples.

A number of decisions that define and constrain the verification model have to be taken prior to the actual model construction. Many of these can only be approximative and have to be refined during the actual model construction, when the necessary knowledge about what is given and what is needed can be matched.

The structure and organisation of the decisions during model constructions should be subject of modelling methods, which we treat in other work.

2.1 What are the epistemological criteria of the model?

For formal verification a formal model is needed. The questions here are: What makes a description in some formal language a formal model? What is the relation between the formal model and the artefact it is supposed to model? And what defines the quality of a model?

We can identify different epistemological criteria. Some of them, e.g., simplicity and completeness, may be conflicting in many settings. When constructing a model we have to decide which criteria should be satisfied, and be aware of the consequences of criteria not being satisfied. When re-using the model for verification of other properties it has to be checked whether the same epistomological criteria are relevant as for the initial verification problem.

The criteria to some extent stem from the requirements for the verification, to some extent from requirements of the model construction process, which should be efficient and lead to a reusable (evolvable) result.

Truthful. Considering our initial question, this is the most obvious quality criterium. The model has to represent the relevant behaviour of the artefact. Coming back to the rationalist view on modelling, we only can falsify models by, e.g., testing. This is reflected in the typical process of model construction, where the first runs of, e.g., a model checker have the purpose of improving the model and removing bugs, rather than proving something about the system. A justification argument constructed together with the model can be used for non-formal evaluation of truthfulness.

Truthfulness, however, is not only obvious. When modelling, we often simplify in a way that the model shows different behaviour than the original system. In protocol modelling, for example, we often do not consider time intervals for sending a message but idealise an interval to a point in time, much like physicists idealise objects to points with a mass. In this case, the model shows behaviour that is not present in the real system, and we could call it *wrong*. Nevertheless, the properties of the model (may) say something about the properties of the original system. Here, the model is not true in the strict sense, but with an additional argument (that can be formal or non-formal) we can consider the model as truthful. The most prominent examples for models that do not coincide with reality are the atom model, and also the models of light, the wave model on one hand, and the particle model on the other hand. They differ from the physical reality, but still, as a model they are useful, because certain effects can be desribed and predicted with them.

Complete. All parts of the system that influence the property to prove are represented. Completeness is always relative to a property, as to be explained below. The evaluation whether a model is complete is in most cases as difficult as the evaluation of truthfulness.

Simple. By simple we mean small, or having a small representation. Simplicity is crucial for computer aided verification: if a model is too large we get stuck in a state space explosion. It is easy to evaluate whether a model is simple enough: by trying it out on a verification tool. Considering the rapid development in tools and in hardware, a model that is too complex today may be simple enough tomorrow.

Understandable. It should be clear *which* design decisions went into the model derivation, *where* they were taken, and *what* are the consequences, including, ideally, what are the faults that are avoided. Here we claim that the understandability of the model is closely related to the understandability of the justification argument that is constructed together with the model. Understandability can be evaluated by showing the model derivation steps to another person and find out whether she understands the steps taken.

Tracable. The relation between artefact and model should go in two directions. The first leads from the model to the artefact: a fault found in the model should be easily traced back to a fault in the artefact. The second interpretation leads from the artefact to the model: it should be obvious which elements of the artefact went into which design decision and are reflected at which point in the model. The justification argument, as well as a well-documented modelling process link artefact and model, and therefore support tracability in both directions.

Efficiently constructable and maintainable. This quality criterium concerns the modelling process and not the model. Obviously, it is to some extent a result of the criteria understandability and tracability. Evaluation of this criterium can only be performed in pactice.

2.2 What is the purpose of the model?

An explicit statement of the purpose is prerequisite for both construction and justification of a model. The question of truthfulness depends on the purpose and is meaningless when no purpose is stated. Depending on the purpose, mantainability can be crucial or irrelevant. Understandability can be a tradeoff between what human readers understand and what a computerised tool requires. The following list may help to clarify the purpose.

Verification vs. design. For formal verification of a certain property, a model only needs to contain what is needed to prove that property. If, for example, we take for granted that a system performs the correct computations and we only want to verify that it is deadlock-free, we may abstract from the specific functionality of its components and from the contents of messages. (This is why process algebra supports internal actions with unspecified functionality.) The completeness of a *verification model* thus is a function of the property to be verified. A *design model* in contrast is used as a "blueprint" to construct the physical artefact and to validate that it has been constructed correctly. Design models thus must contain all information necessary to build it, and they often will be far too complex for formal verification of a property.

A computer program in some high-level language can be understood as a design model for the corresponding machine program. It is complete by definition, it is as truthful as its compiler and operating system is and as understandable als the programmer made it and the language admitted. Real life computer programs, however, usually are far too complex for automatic verification of the system in which they are embedded.

Views. Often the same artefact is considered under different *views*, each of them having its own model. For a coffee machine, for example, we might use different models for timing aspects, such as *the time distance between two cups of coffee is not less than 20 seconds*, and for quality aspects, such as *there there will never be soup powder in the coffee*. The view depends on the purpose, and which idealisations, simplifications and abstractions of an artefact are legitimate depends on the view.

Knowledge integration. Specific for embedded systems is the interdisciplinary character of the knowledge involved. Therefore, an additional purpose of models of embedded systems is *knowledge integration*. Typically, the physical part of embedded systems is built and understood by domain specialists different from software engineers, like, e.g., chemical engineers, mechatronics experts, and process engineers. During modelling the relevant knowledge of these experts has to be identified and transformed into the model. Think of a valve in a chemical plant, opened and closed by a controller. How a valve is working precisely, and how the behaviour of the valve will effect the quantities flowing through a pipe, and what this means for the chemical process, typically is knowledge beyond the scope of a modeller. The modeller learns from a domain expert about the embedded system under consideration, but only when fixing the knowledge in form of

a model the domain experts can identify misunderstandings, or, often, lacking domain paradigms. In this process the model has also the purpose of knowledge communication and transfer. This in contrast to models that treat only one perspective on a system.

The decisions that have to be made explicit here belong to the domain knowledge of the engineers. These decisions become mainly relevant during the model construction phase, where one typically detects that for a further step more knowledge is necessary.

Maintainability and evolvability. Other purposes can be to support *maintainability* and *evolvability* of a system, requiring that small changes in the artefact can be transformed efficiently into the model.

Justification. In addition to the purpose of the model, there is also a purpose of the modelling process, which can be twofold. In the first place, of course, the goal of the modelling process is a model. Second, in the course of modelling a *justification argument* can be constructed, consisting of all decisions taken during the modelling process. Having a justification argument related to the model construction supports different quality criteria of models and their evaluation such as understandability, tracability, and maintainability as discussed in section 2.1.

2.3 What is the object of modelling?

A claim "we have formally verified the correctness of the XXX-protocol (or chemical plant PPP)" is meaningless as long as the exact object is not stated that was modelled (a machine program? a program in a higher level language? an abstract algorithm? the autor's interpretation of an informal standard? a piece of hardware?). In the first place, the choice depends on the property to verify, which may in full detail only be clarified during the modelling phase. Nevertheless the focus should be defined in the beginning and only be changed of there are good reasons to do so. A lot of energy can be wasted in the process of modelling if the object of modelling is a moving target.

Fragment of reality. The first choice concerning the object of modelling is typically the restriction to a fragment of a (bigger) system. When we want to verify, for example, that the window lowering and raising system in an automotive system works properly, we focus on the responsible subsystem. In the first place, the choice of the adequate fragments depends on the property to verify, which may in full detail only be clarified during the modelling phase. Moreover, the automotive example shows immediately a difficulty: as controllers (ECUs) in such a system are connected by buses, subsystems are not working in isolation, but are interfering.

Embedded systems. Traditionally, only software was object to formal verification in computer science. In later work, also hardware, i.e. circuits, was verified by model checking. In the context of embedded systems the overall behaviour is constituted by the plant *and* the control (consisting of controller and control software). Therefore, both (all) parts that contribute to the requirements have to be modelled for verification. Software focussed approaches reduce the plant (often called environment in the software centered approach) to a number of assumptions, i.e. give only a very high abstraction of the physical environment. In our modelling approaches we address also the modelling of the physical aspects. Software in itself is already a formal object, and in this sense it is easier to transform it to a formal model. This does not hold for the physical environment, which makes modelling more difficult.

Levels. A straightforward approach for verification would be to incorporate the source code of the control program in the model of an embedded system. However, due to complexity reasons and state space explosion such an approach will not succeed for real-size problems. A solution to this problem is to split the verification effort into different levels:

At a higher level we incorporate a control *specification* in the model of the embedded system. In general, the specification of the control will be less complex than the control software, and the model at this level is easier to verify. At this level we try to show that the embedded system model satisfies the overall goal, e.g. that a computer controlled plant produces the

At the lower level we try to verify that the actual control program meets the control specification used in the higher level. Here, the focus is that program alone, and we are in the well-understood area of program correctness.

Environment. In any case, some aspects of the *environment* of the focus will also have to be modelled. The environment is constituted by everything around the focus. (Note, that we call the controlled physical part of the embedded system plant, and the environment is outside the embedded system, and not controlled.) Most artefacts will require certain environmental conditions, for example power supply, temperature, humidity, acceleration within a certain rage or, that the busses used for communication are not congested. It is tempting (but infeasible) to start by modelling *all* knowledge about the required environment. It is more economic to formalise the necessary assumptions while constructing the correctness argument, as only those assumptions about its environment need to accompany the model of the focus that are necessary to verify the desired property.

Granularity. Having chosen a fragment of reality as focus, we do not necessarily have to model all its components, and for those that are modelled not the highest granularity is needed. If, for example, we assume that the controller's hardware and operating system behave correctly, we do not need to model circuit and operating system code. If we assume that the engineers have done their work properly, we can take the blueprint of the plant as basis for a model. Also here, it is relevant to make explicit which components are modelled and which components are assumed to be correct.

2.4 What is the structure that we want to model?

Decomposition is one of the most relevant modelling techniques in order to deal with complexity. Decomposition requires the choice of a structure of the artefact, i.e. the identification of components and their interaction.

Structure is not a property of an artefact itself. It is something we impose on the artefact by taking a certain view on it. One of the goals of engineering education is to teach the standard views and structures of the discipline. Different disciplines teach different structures. The software engineer has a different view than the power engineer or the mechatronics expert, and, consequently, they impose different structures. Even within one discipline views can differ. For example, when we identify the processes running on a system, we can choose the borders between processes in different ways. Some structures of an artefact we cannot see at all, because we have not learnt to see

them. Accordingly, the views to be taken are for the greatest part education, and to some extent also creativity.

Taking a certain view on an artefact abstracts at the same time from other views. When we draw an electric circuit, we abstract from the colour of the resistors.

For modelling we have to decide what the most suitable structure is. The structure on the artefact defines, consequently, the possible decompositions. The following criteria have influence on the suitability of a chosen structure and decomposition:

1. The verification property of interest has to be captured, possibly a whole class of verification properties. Typically, we take this as an assumption when we choose a structure.
2. The structure allows for a decomposition that can be represented in a compact form, i.e. allows for a small model.
3. The possible decompositions are understandable and/or intuitive.

As illustration we mention (without definitions) some possible structures and respective decompositions: functional, service-based, process-based, recipe-based, workpiece-based, instrumental, communication-based, event-based, etc. Often, we find different structures mixed within one model.

When verification models are also used for design, as it is done within the model-based approaches, integration of models is an issue. One relevant question here is, whether the different structures imposed allow for composition and, if composition is possible, whether the resulting model is small enough.

In companies, the structure imposed on artefacts that are developed and built is to a great extent prescribed by the organisational structure. Typically, there is a hardware department and a software department, that enforce the splitting into hardware and software in an early phase of modelling and design, even if such an early splitting is not most suitable for the problem to solve.

2.5 In which mathematical domain do we describe our model?

The mathematical domain in which a model is embedded decides which theories can be applied easily, as the following example demonstrates by analogy. A chess board can be modelled like its digital photo as a huge two-dimensional array of RGB-pixels. This would be necessary if a chess-board has to be visualised under various influences of light and shadow. But if we want to reason about chess positions and moves, an 8 by 8 array will be more appropriate. If we only want to solve the eight queens problem [**?**] a vector of 8 integer values may be even more appropriate.

Formal verification models have to be represented in some formal language, in contrast to, e.g., natural language descriptions. Still, the number of possible formal representations is huge. The choice of a suitable mathematical domain is ideally guided by the answers on the previous questions, and related to the expressivness of this domain and the available theories. In practice, often a language is chosen because it is the input language of a powerful tool, or simply because we are familiar with it. As a consequence we often have to “squeeze” a model into a certain language, and idealisations and simplifications are the techniques used then. Ideally, we should justify our

choice, discussing alternatives and understanding the consequences w.r.t. to the purpose of the model, the structure we want to express, the theories we wish to apply, and the epistomological criteria that have to be satisfied.

2.6 What idealisations and simplifications are applied?

Often we idealise behaviour described in a model: e.g. messages arrive at a certain point in time, whereas in the system messages arrive during an interval (as discussed under truthfulness). Then, we (should) have an argument why the properties of the idealised model still say something about the artefact. Such an argument can also be formal, e.g. using equivalence classes.

When we consider a physical object that we want to model, e.g. a valve, then, typically, we identify the observable behaviour of a valve and invent a description of a valve that mimics the same behaviour. A first choice would be to model a valve as an object with two states, open and closed. This may be enough in one certain context. In another context a more detailed observation may be neccessary: valves do not open and close instantaneously, and in the opening and closing phase less amount of fluid passes through than in the completely open state. If small amounts of the fluid are relevant to consider, we need here a model addressing differential equations describing the flow. Possibly, we can also opt for a simplification of differential equations, and possibly we understand that also differential equations provide only an approximation of the physical process and we have to decide whether the error of the approximation is acceptable or not. The choice we take is guided by domain knowledge from, e.g., the chemical engineer, who knows all about valves. It is related to the epistomological criteria as it concerns truthfulness, simplicity, and completeness. It is also related to the purpose of a model, as, e.g., the chemical processes controlled have to be represented in the model adequately, and finally, it is also related to the object of modelling, where the level of granularity of modelling is concerned.

2.7 What is the pragmatics of the model?

Modelling is a labour-intensive process. The effort put in modelling should reflect the pragmatics. The questions below have to be answered beforehand. They belong to the economical aspects of modelling, not to the engineering aspects:

1. How much may it cost to make the model?
2. How long is the model needed? Can we throw it away immediately, or later?
3. If we must keep it for a while, must it be maintained?
4. How long is it going to be maintained, by whom, and what is that allowed to cost?

These aspects belong to the pragmatics of the *model*.

There are also aspects that belong to the pragmatics of the *modelling process*. For example, in an industrial context modelling and design processes have to follow the organisational structures present. These may not necessarily reflect the *ideal* modelling and design steps that would lead to a product in the most efficient way, but optimises (hopefully) on other criteria, economically or socially.

3 Related work

A lot of work has been done around formalising the first steps of the waterfall model of software engineering, e.g. [14]. *Requirements analysis* (see e.g. [12], [7]) starts at an informal level, deals with misconceptions and conflicting stakeholders' wishes, and aims at a formal *conceptual model* of the application domain. Our work shares the position that a systematic analysis of non-formal aspects has to procede the formalisation. However, only a part of the aspects we consider come from stakeholders. Many aspects stem from the embedded system level of from the verification method and technique. The formal description of the resulting model is for our application domain even more relevant.

The problem frame approach [10, 19] describes and decomposes the requirements, the plant and specifes the software. A part of the frames technique is called *frame concerns*. The frame concerns are described as the ingredients needed to solve the problems that are not described in the problem diagrams of this technique. Different types of these concerns are not strictly defined, as there are many different concerns for different classes of problems. The goal of the problem frame approach is not verification models, but, essentially, the construction of verification models could be done following the frame technique. The taxonomy we suggest could be combined seamlessly with the problem frames approach defining number of frame concerns.

In a similar way we expect that our work can complement the KAOS method [3], and the "agendas" of [9].

In [1] a General Property-oriented Specification Method [1] is introduced which aims at the derivation of formal specifications in a systematic way. For example, properties are written in cells of a table whose both columns and rows are indexed with components found while decomposing. Some elements of our taxonomy coincide with steps in their approach.

Hall, Rapanotti and Jackson developed a formal conceptual network based on problem-oriented perspective [8] where they described formally (in sequent calculus) the following steps: the identification and clarification of system requirements; the understanding and structuring of the problem world; the structuring and specification of a hardware/software machine that can ensure satisfaction of the requirements in the problem world; and the construction of adequacy arguments that show the system will satisfy its requirements. Similar questions are addressed in our work, but in this paper we focus on the decisions that are below them.

In [16] Seater, Jackson and Gheyi proposed a technique for software specification starting from the requirement for the plant. Instead of formally proving that the composition of the plant and the machine satisfies the requirement, they move from the initial requirement toward the software requirement. The goal there is to find the requirement at the software interface. While doing this, they write down the assumptions (they call them 'breadcrumbs') on the plant, and an argument why they replace the requirement in the previous step with the one closer to the software interface. The biggest shortcoming of their technique is, as they say, is "the burden placed on the analyst to come up with breadcrumbs..." Our work presented here can contribute to the systematic collection of the "breadcrumbs".

Many papers on formal methods describe successful verification on the basis of verification models. But these usually concentrate on the formal part of the story and pay less attention to model construction and quality.

More general and related principles of design are discussed in, e.g., [4,17]. The resulting models, however, which aim to be complete, can be too complex for present day verification tools. This is why we focus on *verification models*: models that contain just enough to verify certain properties but are much smaller than complete construction models. For them we have in principle the same list of quality aspects as they are in the list of attributes of a well-written software requirement specification [5].

The engineering position on design can be found in [15,18,6]. In [4] design within computer science, following the engineering position, is elaborated. The construction of verification models shares many aspects with these. The difference is mainly contained in the requirement for small models and in the fact that in the context here, we follow the design of the artefact for the design of the model.

In other work [11] we addressed the role of assumptions in the model construction process. Assumptions are closely related to modelling decisions in the sense that often an assumption (e.g. about the environment) provides the basis for a decision (e.g. abstracting from a certain phenomenon in the environment).

4 Conclusion

We do not suggest another modelling method here. As many modelling methods start with the assumption that a number of decisions we suggest have already been taken, the list of decisions here can be seen as complement to exisiting modelling methods.

We propose a taxonomy of modelling decisions for verification models. The focus of our application area is verification of embedded systems. We see the usefulness of the taxonomy in the following aspects:

1. It can provide guidance in taking modelling decisions. Taking the taxonomy as a list of possible decisions, the modeller can find the decisions s/he has to take or has already taken implicitly, or suggests what *can* be made explicit at all.
2. When a proof of the correctness of an artefact is delivered, it also has to be made explicit what constituted the verification model. This is important for both cases, where the overall verification is finished and the restrictions have "to be printed on the box", and, where the verification concerned only a fragment of a bigger system and the verification of this fragment has to be positioned within a bigger correctness argument. In both cases we have to clarify the focus and the restrictions of the verification trajectory. Modelling decisions as well as assumptions in the modelling process are a part of such a description.
3. Modelling languages are formal languages, and often not very readable. The modelling decisions in explicit form can help to explain the model to (laymen) stakeholders.
4. Constructing a verification model is a time consuming effort. Efficiency of the verification process improves when models can be re-used, when a product is slightly changed, and also when a similar product (e.g., in a product family) is developed.

5. Ideally, the result of a modelling process is not only a model, but also its explicit justification. The meaningfulness of a result of formal verification depends on the degree to which the choices of theory, abstraction and idealisation can be justified. Answers to the question in the taxonomy contribute to the justification argument. Having it in an explicit form also supports a review process which improves the quality of the model.

Alltogether, our main claim is that making modelling decisions explicit helps to improve the quality of the models, makes the justification argument stronger and supports reusability of models.

The taxonomy suggested grew during years working on verification of embedded systems. It helped in our work, with modelling as well as with communicating our results to colleagues. Also, some enthousiast colleagues found the taxonomy useful for their own work. However, we have no study on the improvement that the taxonomy provides for modelling in contrast to modelling without this taxonomy.

References

1. C. Choppy and G. Reggio. Towards a formally grounded software development method. Technical Report DISI-TR-03-35, Universita di Genova, Italy, 2003.
2. Edward W. Constant. *The Origins of the Turbojet Revolution*. The John Hopkins University Press, 1980.
3. Anne Dardenne, Stephen Fickas, and Axel van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *IWSSD '91: Proceedings of the 6th international workshop on Software specification and design*, pages 14–21, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.
4. Subrata Dasgupa. *Design Theory and Computer Science*. Cambridge University Press, 1991.
5. Al Davis. *Software Requirements: Analysis and Specification*. Prentice Hall, 1990.
6. Eugene S. Ferguson. *Engineering and the Mind's Eye*. MIT Press, 1992.
7. M. D. Fraser, K. Kumar, and V. K. Vaishnavi. Informal and formal requirements specification languages: Bridging the gap. *IEEE Trans. Softw. Eng.*, 17(5):454–466, 1991.
8. J. G. Hall, L. Rapanotti, and M. Jackson. Problem oriented software engineering: A design-theoretic framework for software engineering. *sefm*, 0:15–24, 2007.
9. M. Heisel and J. Souquières. A method for requirements elicitation and formal specification. In *Proc. of the 18th International Conference on Conceptual Modeling*, volume 1728 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 1999.
10. M.A. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2000.
11. J.Marincic, A.Mader, and R.Wieringa. Classifying assumptions made during requirements verification of embedded systems. 2008. (Accepted for publication in the proceedings of the International Working Conference on Requirements Engineering (REFSQ) 2008).
12. Anna Perini, Marco Pistore, Marco Roveri, and Angelo Susi. Agent-oriented modeling by interleaving formal and informal specification. In *AOSE*, pages 36–52, 2003.
13. Karl Popper. *The Logic of Scientific Discovery*. Routledge Classics, 1959.
14. Winston W. Royce. Managing the development of large software systems. pages 1–9, 1970.
15. D.A. Schön. *The reflective practitoner*. Basic Books, 1983.
16. R. Seater, D. Jackson, and R. Gheyi. Requirement progression in problem frames: deriving specifications from requirements. *Requir. Eng.*, 12(2):77–102, 2007.

17. Herbert A. Simon. *The Sciences of the Aartifical*. MIT Press, Cambridge, 1981.
18. Walter G. Vincenti. *What Engineers Know and How They Know It*. The John Hopkins University Press, 1990.
19. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.