A Modelling Method for Embedded Systems

Ed Brinksma* **— Angelika Mader*** **— Jelena Marincic*** **— Roel Wieringa***
* University of Twente, The Netherlands
{Brinksma | Mader | J.Marincic | R.J.Wieringa} @ ewi.utwente.nl

Abstract. *We suggest a systematic modelling method for embedded systems. The goal is to derive models (1) that share the relevant properties with the original system, (2) that are suitable for computer aided analysis, and (3) where the modelling process itself is transparent and efficient, which is necessary to detect modelling errors early and to produce model versions (e.g. for product families). Our aim is to find techniques to enhance the quality of the model and of the informal argument that it accurately represents the system. Our approach is to use joint decomposition of the system model and the correctness property, guided by the structure of the physical environment, following, e.g., engineering blueprints. In this short note we describe our approch to combine Jackson's problem frame approach [1, 2] with a stepwise refinement method to arrive at provably correct designs of embedded systems.*

1 Introduction

Successful verification of embedded systems needs *good* models, i.e. models that share the relevant properties with the original systems (truthful), that are small enough to be analyzed by verification tools (feasible), and, that can be derived in an efficient and transparent way (trustworthy). In state-of-the-art verification most focus is on verification algorithms and tools, and less on modelling issues (model hacking).
For embedded systems modelling decomposes in a natural way into modelling of the control and modelling of the physical environment. In general, modelling software is easier, because software in itself is already a formal object. For the physical environment modelling has to bridge between an informal, physical object and a formal one. This process is far from understood, i.e. different people tend to produce different models of the embedded system, and our trust in the models often depends upon our trust in the people who made them. We want to improve the quality of the modelling process that is at the heart of the verification process. By this we mean that (1) In the modelling process, reusable and validated knowledge about the modelled system is used, making the modelling process more efficient and less dependent on the genius of the modeller and (2) The justification that the model accurately represents the system (wrt a correctness property) is clear and understandable, making the resulting model more reusable. Our solution idea is to construct the system model and the proof of its correctness wrt the property of interest at the same time. We produce the model by means of stepwise non-monotonic decomposition of both the environment and the property of interest, so that the accuracy of the model wrt the property is justified by a structural similarity between the model and the system.

2 The Method

Our goal is to derive *good* models of embedded systems that can be used for verification. Quality criteria for a *good* model are: (1) The model shares the properties of interest with the original system. (2) The model is suitable for computer aided analysis, i.e. small sized models. (3) The derivation process is transparent and efficient, such that design errors can be detected easily and variations of the model (for, e.g., product families) can be applied straightforwardly.

The systems we consider are control applications, consisting of a control machine interacting with a physical environment. The control machine itself consists of a physical part, such as a programmable logic controller (PLC), sensors and actuators, and a software part. We want to achieve the quality criteria mentioned above by a systematic modelling method with the following ingredients:

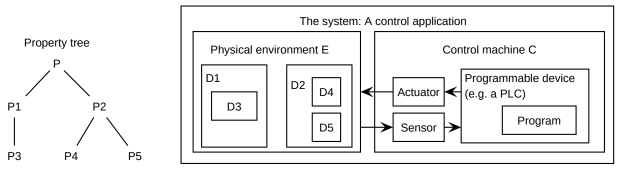

Figure 1: Stepwise decomposition of the environment model and the correctness proof. P is a property of desired behaviour D of the environment. D is decomposed into Di and P_i is a property of D_i.

A verification theorem that describes the logical relation between environment, control and properties: $E \wedge S \models P$, where E describes the potential or assumed behaviour of the environment (without the control machine) and P describes the desirable behaviour of the environment (interacting with the control machine). S is a specification of the control machine. The verification theorem guides our method in the sense that we will have a number of instances of the theorem, that are constructed by addition of (physical) details, design decisions, decomposition, etc.

Non-monotonic decomposition of environment, control and properties with respect to the desired behaviour (see figure 1). The justification of the decomposition steps comes from different sources, such as engineering diagrams, problem patterns, experiments.

Non-monotonicity means that adding new details can require more assumptions, natural laws etc, such that the refined instance of the verification theorem may no longer imply the one at the previous level. Moreover, adding more details can also result in weaker properties, e.g. when a technical restriction only allows for a less general goal. Describing all possible behaviour of the environment typically ends up in models that are too large for computer aided analysis. Therefore, a relevant aspect here is that we restrict the description to the desired behaviour (and the error cases we do want to consider).

Patterns. Another relevant ingredient of our method is the use of problem and modelling patterns, following the idea of problem frames introduced by Jackson [1, 2]. Using patterns gives (at least) two advantages: first, it makes a decomposition argument more transparent, because we can use some form of standard argument, and, second, it makes the decomposition steps more reliable, because we make use of well-proven solution fragments. To make the idea of problem frames useful we need a library of identified patterns with modelling solutions.

3 An example

Due to lack of space we will only demonstrate some features of our method and cannot present the formal representation of the design. However, in earlier work we have proven a description of this example with the theorem prover PVS and the model checker Uppaal [3].

The example is a small PLC-controlled Lego-plant that sorts yellow and blue blocks according to their colour (see figure 2). A number of blocks is in a queue. When the belt is moving the first block of the queue moves to the belt. At the scanner the colour of a block can be detected. A block moves further on into the sorter. The sorter consists of two fork-like arms, each driven by its own motor and equipped by an angle-sensor that can be used to check whether the sorter arm has made a full rotation.

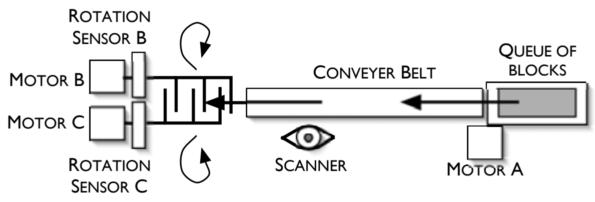

Figure 2: Top view on the Lego plant

We start with a description of the system by a problem diagram in figure 3. The problem frame chosen is the simple *required behaviour frame*. We have a Requirement $\mathcal{R}_0$ and the interface description **a**. $\mathcal{R}_0$ represents the desired property, which is a statement only about the plant behaviour, not the machine. The interface description **a** contains the phenomena that should occur in both models, the one of the plant and the one of the machine. Informally, we have $\mathcal{R}_0$: All blocks will be sorted eventually by the plant. The verification theorem in its first version is:
$\mathsf{Plant} \wedge \mathsf{Machine} \implies \mathcal{R}_0$

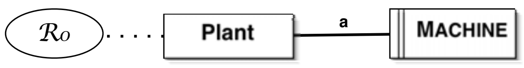

Figure 3: Problem diagram of the Lego plant and the Machine (Control)

At this level of abstraction we do not yet have a meaningful description, so we continue with the first refinement. Figure 4 contains the first refinement step, which corresponds to an *architectural decomposition* (in contrast to a process based or a functional decomposition). Here, we have two kinds of interface descriptions, a_i's for the interface between the PLANT and the MACHINE, and p_i's between the (architectural) components of the plant, which mainly consists of the description of the movement of blocks.
Additionally, the component behaviour has to be described in order to relate the phenomena on its interface to its internal phenomena. The QUEUE consists initially of a pile of blocks. This is an extra initial condition INIT that will be added to the left-hand side of our verification theorem. The first block of the queue goes to the belt. The BELT description has to represent the blocks on the belt and their behaviour when the belt moves. Additionally, when proving (instances of) the verification theorem, it turnes out that we also need a specification of the behaviour of blocks, when the belt is not moving, viz. ”staying where they are”. We regard this as a natural law $\mathcal{N}$ and add this also to the left-hand side of our verification theorem. The SCANNER can distinguish three different values, for a yellow block, a blue block and no block. This implies a restriction on the requirement $\mathcal{R}_0$. We get a weaker requirement $\mathcal{R}_1$ saying, if there are only blue and yellow blocks in the queue then these will eventually be sorted. The SORTER can either be idle or sorting. A block can only enter the SORTER if it is idle. The MACHINE must contain an abstraction of the phenomena in the plant. With respect to the SORTER the MACHINE has to “know” whether there is a block or not in it. Unfortunately, we have no sensor that would detect the presence of a block in the SORTER. The MACHINE can only derive the presence of a block in the SORTER, if the block had been at the SCANNER before and the belt was running long enough to transport the block to the sorter. We therefore have to determine by

experiment the maximum time that a block needs to get from the SCANNER to the SORTER while the belt is running (open loop control). Additionally, we also have to assure that within this maximum time it cannot happen that two subsequent blocks arrive at the sorter. A sufficient condition is that subsequent blocks on the belt have a minimum distance. Also this condition will be an additional requirement $\mathcal{D}$ on the left-hand side of the new instance of our verification theorem.

The second instance of our verification theorem therefore has the form: $\mathsf{Init} \wedge \mathcal{N} \wedge \mathcal{D} \wedge \mathsf{Queue} \wedge \mathsf{Belt} \wedge \mathsf{Scanner} \wedge \mathsf{Sorter} \wedge \mathsf{Machine} \wedge \mathsf{Natural\_Laws} \implies \mathcal{R}_1$
It is obvious that between this instance of the verification theorem and the previous one no implication holds: informally, there is an increasement of knowledge about the plant and machine which has the non-monotonicity of the decomposition as consequence. A further decomposition of the system could address the belt motor, the sorter motors, rotation sensors etc.

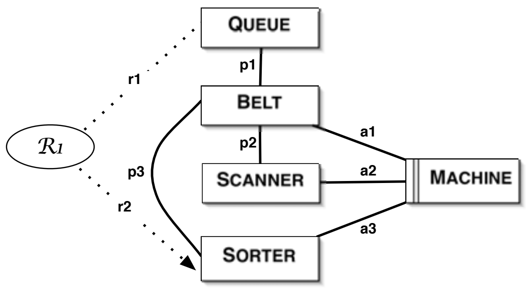

Figure 4: Refined problem diagram of the Lego plant and the Machine (Control)

4 Discussion and conclusions

Our modelling method progresses from a top-level understanding of the system in its environment into more detailed understanding, decomposing the environment model and its desired properties in parallel. We stop decomposing when we reach environment phenomena that can be observed or controlled by the control machine. At that level, we have obtained a correct specification of the controller. We have applied this method on several cases, including the one discussed in this paper [3, 4]). Many research questions remain, some of which we mention here.
The idea of problem frames (rather than solution patterns) was introduced by Jackson [1, 2] as a means to identify reusable problem structures in different problem domains. In our example we used a simple *required behaviour frame*, as described in Jackson's sluice control problem. Specific issues to be addressed for frames are: (1) The compositionality of problem (solution) frames,
(2) The robustness of decomposition trees under small changes in properties and/or environments and (3) Systematic analysis of environment assumptions and consequences of their failure to hold.
Another research question is if informal engineering argumentation (physical, graphical, ...) can be combined with formal reasoning. We believe that the decomposition process should provide the justification of the accuracy of the model wrt the required property.
Acknowledgment. This paper benefited from extensive discussions with Michael A. Jackson.

References

[1] M.A. Jackson. *Software Requirements and Specifications: A lexicon of practice, principles and prejudices*. Addison-Wesley, 1995.

[2] M.A. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[3] J. Kratz. A case study in PLC control: two ways of verifying PLC control software for a lego plant, 1999.

[4] 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.