Classifying Assumptions Made during Requirements Verification of Embedded Systems

Jelena Marincic[*], Angelika Mader, and Roel Wieringa

Department of Computer Science, University of Twente, The Netherlands,
P.O.Box 217, 7500 AE Enschede, The Netherlands
{j.marincic,mader,roelw}@ewi.utwente.nl

Abstract. We are investigating ways to improve the process of modelling of embedded systems for formal verification. In the modelling process, we make a mathematical model of the system software and its environment (the plant), and we prove that the requirement holds for the model. But we also want to have an argument that increases our confidence that the model represents the system correctly (with respect to the requirement). Therefore, we document some of the modelling decisions in form of a list of the system assumptions made while modelling. Identifying the assumptions and deciding which ones are relevant is a difficult task and it cannot be formalized. To support this process, we give a classification of assumptions. We show our approach on an example.

1 Introduction

Models have increasing relevance in embedded system design. Our focus is on the construction of embedded systems verification models. Our goals are:

(1) We want to develop a modelling method. We share the observation of [1] that more research is spent on developing new languages and tools than on providing methods for using the existing ones. A major difficulty here is that modelling cannot be purely formal. We claim that the non-formal steps do not follow unpredictable irrationalism, but are part of educated creativity, following a systematic way of thinking.

(2) Having constructed a verification model we also want its justification - a correctness argument that makes us convinced that successful verification of the model reflects the desired behaviour of the embedded system. The correctness argument includes the assumptions and modelling decisions about the embedded system we have taken during modelling. Changing the assumptions can invalidate the model justification. Therefore, we propose to write down a list of the assumptions made while modelling.

(3) Identifying an assumption and deciding whether it is relevant are informal activities, difficult to capture by a formal approach. To help the modeller, we present a classification of assumptions. The classification presented in this paper

[*] Supported by the Netherlands Organisation for Scientific Research (NWO), project 600 065 120 241420, Modelling Control Aspects of Embedded Systems.

does not depend on a formal modelling or verification technique. The classifications of assumptions also help us understand the modelling activity itself. We believe that checking the assumptions we made against the classes we identifed, gives more structure to the way of thinking and argumentation.

Terminology and Basic Concepts. An embedded system consists of a controller and a controlled, physical part. By **plant** we denote the controlled, physical part, and by **environment** everything outside the embedded system. The control software is abbreviated to **control**.

A **model** is a formal representation of the system, e.g., a diagram or a timed automaton. We model both the plant and the control and verify them against the required behaviour. The verification problem is to prove that a plant P and a control C together satisfy certain requirements R, denoted by $C \wedge P \models R$. This is analogous to [2, 3], but different from other approaches, where only the control software is modelled.

To conclude that the real system satisfies the required behavior, we need a **model justification** - an argument that the model and the formal requirement represent the system and the required behaviour. Such a justification can be given by reconstructing the modeling process into a rational process. In this paper we focus on the role of assumptions in rationalizing the modeling process.

In Sect.2 and Sect.3 we will present our classification of assumptions and will demonstrate it on an example. After briefly discussing related work in Sect.4 we will draw conclusions in Sect.5.

2 Classification of Assumptions

We define an assumption as a statement that refers to the plant and environment, and is taken for granted to be true for the purpose of the model justification. As control specifiers, we place constraints on the control behaviour, but we cannot place constraints on the plant; we can only make assumptions on its behaviour. Assumptions can be stated formally - then they are part of the formal proof, or non-formally - in that case they are part of the justification argument. The first two classes below answer the question *what* the assumptions are describing. The next two are focusing on the criteria of their changeability. The third group of the classifications focus on the relevance for the system users.

C1: Assumptions about system components. The requirement we want to verify determines where we draw the border between the system and its environment and what system aspects we will describe in the model. After that, we decompose the system, describe each component and, if necessary, decompose further. When decomposing the system, we simultaneously decompose the requirement, where each sub-requirement should be satisfied by a system component, and all sub-requirements together should imply the original requirement.

We can decompose the system in many different ways. We can make a process decomposition, a decomposition to the physical components, functional decomposition etc. The components can be described through assumption-requirement pairs in the form $assum(i) \implies req(i)$, where req is the subrequirement we found while decomposing the system. For example: "If the wire is not longer than 12m (assumption), then the signal strength is sufficient for correct transmission (requirement)".

C2: Assumptions about system aspects. A system aspect is a group of system properties, usually related to one knowledge domain. An embedded system has electrical, mechanical aspect etc. When designing the control and verifying the system requirement, we might need assumptions coming from these different knowledge domains. If, e.g., we are designing shut-down system procedure for an embedded system, we want to know the electrical characteristics like capacity and resistance of the circuit that delays power off, to calculate the time the procedure has to save the data.

C3, C4: Necessary and Contingent Assumptions. Depending on the context in which we use the system, some of the assumptions we take as true and do not consider them as changeable.

Natural laws, like for example physics formulas, are considered to be true. If we have a system with a conveyor belt that transports bottles from the filling place, we will assume that its users will put the conveyor belt on a horizontal surface. Some of the plant components can be described with *engineering formulas* which we do not doubt. For example, the signal transmission through fiber optic cable is described with formulas that precisely calculate optical signal properties.

Contingent truths on the other hand may change. There are some facts about the system for which we are not sure whether they will change or not. In practice, it often happens that we have the plant and start designing the control software as if the plant is fixed, whereas in practice components are replaced. For example, if we have a conveyor belt that has to move faster, we can replace the existing motor with a more powerful one. (Then, we would have to change some parameters in control law implemented by control software.) Another example is that the plant is fixed, but our knowledge about it is changed. A domain expert can provide an improved formula describing the system behaviour.

C5: Constraints on the Plant and Embedded System Environment Some of the assumptions we make pose constraints on the plant and users. We cannot be sure in advance that they will be fulfilled. The best we can do is to list them and deliver them together with the system. These assumptions are not part of the model - they can be seen as a label on the 'delivery box' of the system. For example: "If the weight in the cabin is larger than 20 and less than 150kg, the lift will go to the floor determined by the button pressed in the cabin."

3 Example - The Lego Sorter

The Lego sorter is a PLC (Programmable Logic Controller)-controlled plant made of Lego bricks, DC motors, angle sensors and a colour scanner [4]. Bricks of two colours are stored in a queue. They enter a belt one after another, and possibly more than one brick is on the belt. The belt is moved by a motor. Bricks are transported by the conveyor belt to the scanner and further on to the sorter.

The scanner can distinguish a yellow, blue or no brick in front of it. Putting a brick of another colour in front of it would cause the scanner to enter into an unknown state. The sorter consists of two fork-like arms. Each arm can rotate a brick to one of the sides of the plant. Each sorter arm is controlled by its own motor and has its own rotation sensor that senses the angle of the arm. The starting angle is 0, and as the arm rotates it changes to 360 degrees.

Classification C1: Assumptions about components

Components	*Assumptions*
Queue	*A6:* There are only blue and yellow bricks in the Queue.
Belt	*A7:* The motor moving the Belt is working properly. *A12:* There is a minimal distance between bricks so that *A12.1:* There is always "nothing_at_scanner" observed by Scanner before the new brick is observed and *A12.2:* There will be no new brick in front of the Scanner before a previous brick moves to the Sorter. *A13:* The order of a brick observed at Scanner and a new brick entirely on the Belt is not deterministic. *A15:* The order of a brick arrival to the Scanner and previous brick sorted Is not deterministic.
Scanner	*A8:* The Scanner observes the color all the time (not interrupt-driven). *A8.1:* The Scanner transmits the signal with the delay D1 *A12, A13, A15*
Sorter	*A9:* The rotation sensors and motors work properly and transmit signals with no delay.. *A9.1:* The rotation sensors show only relative arm position. *A11:* The sorter starts with proper initial position. *A12, A15* *A14:* Upon start, Sorter and Belt are empty.
Bricks	*A16:* Bricks are standard Lego bricks (50mm x 15mm x 7mm) that fit in the Queue. *A12, A13, A15*
Operator	*A5:* An operator will put the bricks in the Queue. *A11:* The sorter starts with proper initial position. *A14:* Upon start, Sorter and Belt are empty.
Controller hardware ' and OS	*A1:* Computer hardware works properly. *A2:* The OS supports the control we design. *A3:* The sampling period is such that control can observe rotation angle with sufficient granularity. *A4:* The PLC controller supports the desired sampling frequency.

Classification C2: Assumptions about aspects

Aspects	*Assumptions*
Control eng.	*A3:* The sampling period is such that control can observe rotation angle with sufficient granularity. *A8.1:* The Scanner transmits the signal with the delay D1 *A9:* The rotation sensors ….transmit signals with no delay. *A9.1:* The rotation sensors show only relative arm position.
Mechanical	*A1:* Computer hardware works properly. *A12:* There is a minimal distance between bricks so that *A12.1:* There is always "nothing_at_scanner" observed by Scanner before the new brick is observed and *A12.2:* There will be no new brick in front of the Scanner before a previous brick moves to the Sorter. *A13:* The order of a brick observed at the Scanner and a new brick lying entirely on the Belt is not deterministic. *A15:* The order of a brick arrival to the Scanner and previous brick sorted is not deterministic. *A16:* Bricks are standard Lego bricks (50mm x 15mm x 7mm) that fit in the Queue.
Electrical	*A7:* The motor moving the Belt work properly. *A9:* The rotation sensors and motors work properly *A11:* The sorter should start with proper initial position.
Implementa-tion	*A2:* The operating system supports the control that we design. *A4:* The PLC controller supports the desired sampling frequency. *A8:* The Scanner observes the color all the time (not interrupt-driven).

Classifications C3, C4: Necessary and contingent assumptions

Natural laws	If on the Belt for T1 seconds, and the Belt is moving, the brick changes its position. If the Belt is stopped, and the brick is on it, it won't change its position.
Engineering formulas	The Belt speed *v* is a function of the value *m* put on the Belt motor: v = k x m
Contingent truths	Here, all the descriptions about the Scanner are contingent. We might replace the Scanner that for, example distinguishes more colours or that has a zero delay The same holds for other plant assumptions; we might, for example, reconstruct the Plant in such a way that the Sorter arms have to move in different direction when sorting

Classification C5: Plant and environment constraints

A5: An operator will put the bricks in the Queue.

A6: There are only blue and yellow bricks in the Queue. (If a brick of any other colour is there, the Scanner will recognize it as Yellow or blue and will sort it on one of the sides.)

A11: The sorter should start with proper initial position.

Fig. 1. List of the assumptions shown according to the classification criteria we found

The requirement is: *"Eventually all the bricks from the queue will be moved by the sorter to the side corresponding to their colour"*. We designed the control, and modelled and verified the system (see [5] and [6]). The assumptions that we identified are presented in table in Fig.1. Some of the assumptions are part of the model, but are also listed in the table.

4 Related Work

From the work following the approach of [2] we mention only the most similar to ours. The problem frames technique [3] defines *frame concerns* through examples of issues that have to be addressed and that are not described in the problem diagrams, e.g., initialization of the software and hardware.

In [7] a technique for software specification is described, starting from the requirement for the plant. Assumptions ('breadcrumbs') on the plant are collected, and an argument for each modelling step. No guidelines for finding assumptions are given.

In [8] a formal conceptual network based on problem-oriented perspective is developed, where modelling steps are formally described. We, on the other hand, are looking for ways to systematically perform these steps.

In the area of requirements engineering, the goal-oriented methods have a significant place. Our classification of assumption could be useful in the phases of requirements analysis of the KAOS method [9]. In the Tropos methodology [10], when defining the circumstances under which a given dependency among two actors arises, a modeller has to learn about the system, so our assumption classification might be useful there, too.

The problem of modelling method is addressed in [1] by *agendas*, a list of modelling steps. The transition from informal to formal is performed in one of the first steps of the requirements elicitation, while we formalize only the last steps when the complete knowledge about the system is available.

In [11] a General Property-oriented Specification Method is introduced, where assumptions are collected in the cells of a table made while decomposing the system. This framework is restricted to the use of labelled transition systems.

5 Discussion and Conclusion

Formal methods are applied in a non-formal world and we cannot give an algorithm how to collect the assumptions. Instead, we found different classes of assumptions that are made in the modelling process and different ways of identifying the assumptions.

Making assumptions explicit is not so much a matter of using the appropriate languages or tools. In the first place it requires a discipline of thought, and being aware what we do during modelling activity can help here by saying at which point of the modelling process we have to look for assumptions, and which form these can have. Different categories of assumptions mean that we have different views to the system, even if we chose one decomposition. If we restrict ourselves into one single view or decomposition, we might omit an important assumption. Therefore, classification of assumptions is useful as a checklist to go through when describing the system; this is a hypothesis that needs further proving. An experiment in which a group of modellers will be presented with assumptions classification and one not, is needed to make this statement an empirical claim. This is the part of our further work.

We plan to look closer into subclasses of embedded control systems for which we can make specialized, more concrete modelling guidelines. We will focus on the communication of control engineers and verification experts while doing formal verification, to identify the boundaries of these two knowledge domains, and to make more clear what one expert has to know about other expert's area.

References

[1] Heisel, M., Souquières, J.: A method for requirements elicitation and formal specification. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., Métais, E. (eds.) ER 1999. LNCS, vol. 1728, pp. 309–325. Springer, Heidelberg (1999)

[2] Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. 6(1), 1–30 (1997)

[3] Jackson, M.: Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, Reading (2000)

[4] MOCA project - ongoing work, `http://moca.ewi.utwente.nl/WORK.html/`

[5] Marincic, J., Wupper, H., Mader, A., Wieringa, R.: Obtaining formal models through non-monotonic refinement. Technical report TR-CTIT-07-33, CTIT, Univ. of Twente, The Netherlands (2007)

[6] Marincic, J., Mader, A., Wieringa, R.: Capturing assumptions while designing a verification model for embedded systems. Technical report TR-CTIT-07-03, CTIT, Univ. of Twente, The Netherlands (2007)

[7] Seater, R., Jackson, D., Gheyi, R.: Requirement progression in problem frames: deriving specifications from requirements. Requir. Eng. 12(2), 77–102 (2007)

[8] Hall, J.G., Rapanotti, L., Jackson, M.: Problem oriented software engineering: A design-theoretic framework for software engineering. sefm 0, 15–24 (2007)

[9] Dardenne, A., Fickas, S., van Lamsweerde, A.: Goal-directed concept acquisition in requirements elicitation. In: Procs of IWSSD 1991, pp. 14–21. IEEE Computer Society Press, Los Alamitos (1991)

[10] Giorgini, P., Mylopoulos, J., Sebastiani, R.: Goal-oriented requirements analysis and reasoning in the tropos methodology. Engineering Applications of Artifcial Intelligence 18/2 (2005)

[11] Choppy, C., Reggio, G.: Towards a formally grounded software development method. Technical Report DISI-TR-03-35, Universita di Genova, Italy (2003)