Modelling Embedded Systems by Non-Monotonic Refinement

A.Mader, J. Marincic*, and H.Wupper

[1] University of Twente, Department of Computer Science, Enschede, The Netherlands, mader@cs.utwente.nl
[2] University of Twente, Department of Computer Science, Enschede, The Netherlands, j.marincic@cs.utwente.nl
[3] Radboud University Nijmegen, Institute for Computing and Information Sciences Nijmegen, The Netherlands, Hanno.Wupper@cs.ru.nl

Abstract. This paper addresses the process of modelling embedded systems for formal verification. We propose a modelling process built on non-monotonic refinement and a number of guidelines. The outcome of the modelling process is a model, together with a correctness argument that justifies our modelling decisions. After explaining the method, we demonstrate it on a small example.

1 Introduction

Many descriptions of modeling methods and introduction to modelling start at a point where the requirements are clear, and the relevant knowledge about a system is present. At this point formalisation is an important issue, as well as representation, transformation of models, and integration. This paper addresses the earlier stage: the requirements are only roughly known, when we do not know which parts and properties of the system are relevant to satisfy the requirements.

We understand modelling as the process that accumulates knowledge that is necessary to satisfy a certain purpose. The model is then a representation of that knowledge. The purpose of the models we construct is verification.

In this paper we suggest a top-down approach that guides the compilation of knowledge about a system that must be included in the model. It shares with classical refinement approaches the stepwise decomposition and detailing. The main difference to classical refinement is the growth of knowledge, built into the modelling process, where a more refined version of the model can invalidate earlier versions. Therefore, our modelling approach is called non-monotonic refinement.

The systems we consider are embedded systems. We do not focus on correctness of the embedded software alone *with respect to a given specification.* Rather, we follow an approach similar to the one of Jackson [5] and consider the correctness of the software or, if available, its specification *with respect to the*

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

system in which it is embedded, i.e., we describe the whole system - the software, the plant controlled by it and the requirement for the overall system.

Software in itself is already a formal object. Modelling software, in principle, can therefore rely on formal manipulation. This does not hold for the physical part of an embedded system, the plant. There modelling has to bridge between the real world and a formal world. According to this bridge-function modelling cannot live only in the formal world, but necessarily involves also non-formal elements. The formal elements of modelling can be supported by tools and languages, that are developed in a great number. The non-formal elements of model construction are often considered as purely creative, where the spark of inspiration is something that cannot be captured. We, however, claim that the non-formal aspects of modelling are not only creative, but are to a great extent rational, derived from a modellers education and experience. Our goal is to develop a modelling method, where the systematics behind the non-formal aspects is made explicit. We do this by providing modelling guidelines and non-monotonic refinement. Accordingly, the reader should not expect a formalism that describes the modelling process or a modelling language. Our contribution is in covering the non-formal steps that are inevitable part of the modelling process.

When constructing a model we also want to build a correctness argument - a formal, non-formal or semi-formal argument that convinces us and other stakeholders that the model is adequate. The correctness argument captures the rationale of our modelling decisions. We claim that having the rationale documented increases our trust in the model as well as reusability, evolvability, and tracability of the model.

In this paper we suggest a modelling process built on non-monotonic refinement and a number of guidelines. The outcome of the modelling process is a model, together with a correctness argument that justifies our modelling decisions. Note, that in AI the term "nonmonotonic refinement" is used differently. What we do *not* suggest here is an algorithm that constructs a model, nor do we suggest a modelling language - for the paper here we use logic, any other language could be used, textual or graphical.

The paper is organised as follows. A tiny example in section 2 is intended to provide the basic intuition. Section 3 describes the formal steps of non-monotonic refinement. In section 4 we give guidelines for our modlling method. A more elaborated example can be found in section 5. In section 6 we discuss related work. The paper concludes with section 7.

2 A small example.

This example is intended to provide intuition how the method works, presented step by step. We will use a washing machine as running example. We hope that the reader sees how the informal specifications we suggest can be formalised in her favourite language.

Each physical part of the washing machine is required to perform its task ("commitment") only as long as its environment guarantees certain conditions like power supply, and the necessary input ("assumption"). The specific notation of such (assumption, commitment) pairs will depend on the languages used. Here, we shall use the notation $a_i \rightarrow c_i$.

Start from C_0 - a simple, general version of the property to be verified.

A formula (depending on the language used) that says: "I want clean clothes, at any time, immediately!". We know that we mean "as soon as possible", but that will not be expressible in most specification languages.

Use expert knowledge about the artefact to identify a sub-process that somehow contributes to this goal. An obvious choice is: move the clothes in warm water with a detergent.

For that process, find a component in the physical artefact that is used to perform it. In our example, this will be a tumbler together with its engine.

For this component, write a formula $a_1 \rightarrow c_1$ that contributes to C_0 but does not contain more knowledge than necessary to contribute to the overall goal. If clothes, warm water and a detergent are in the tumbler in the beginning, and if the engine is powered for a certain period, the clothes will be rather clean - and wet - in the end. At this moment we may realise that we forgot to require sufficiently dry clean clothes in C_0.

Obtain knowledge about the quantities the component requires and can handle; incorporate these in $a_1 \rightarrow c_1$. This is the moment to decide whether we are modelling a household washing machine of a professional one. In casu, no more clothes than 4.5 kg can be handled, and the process will require 30l of water of 60C and last one hour if it has to remove 99% of the dirt. The c_1 will also tell us that there is a lot of dirty water te be disposed of, and that the clothes are wet.

Decide what of the a_1 will be ensured by another component of the artefact and what has to be provided by the environment of that artefact. In our example the warm water will be provided by a built-in heater, while electricity and detergent will have to be provided by the environment.

Decide what of the c_1 will have to be dealt with by another component. As we want rather dry clothes, we need some kind of spin-dryer and a pump. The necessary drain we shall require form the envorinment.

Replace C_0 by a weaker $A_1 \rightarrow C_1$ that reflects the new state of knowledge. A_1 will require a certain amount of electricity and detergent as well as drains. C_1 will contain the quantities 4.5 kg, 1 hour, and 99%.

Now look at one of the processes that were discovered during the previous step to decrease the "difference" between $a_1 \rightarrow c_1$ and $A_1 \rightarrow C_1$. Specify it in the same way as $a_2 \rightarrow c_2$. For example, we will find a heater that provides warm water provided it gets cold water and a certain amount of electrical energy.

Adapt $A_1 \rightarrow C_1$ accordingly, which gives us $A_2 \rightarrow C_2$. A_2 is stronger than A_1: cold water and more electricity is required. C_2, however, is weaker: even more

time is taken. $A2 \rightarrow C_2$ is therefore weaker than $A_1 \rightarrow C_1$.

Continue this method, adding new processes and their system components until you arrive at a provable $a_1 \rightarrow c_1, a_2 \rightarrow c_2, ..., a_n \rightarrow c_n \models A_n \rightarrow C_n$ In our example, only a pump has to be added. The tumbler, together with its engine powered with a higher voltage and with the pump will do the job.

Not before we have obtained this complete picture, we start with the difficult part of conflicting resources. The specification of one of the processes in our example requires that the water stays in the tumbler for a certain period, while the specification of another one requires it to be pumped out while no new water streams in. A physical tumbler cannot implement both specifications unless it has valves that can be opened and closed. Likewise, engines and pumps must have switches to activate and deactivate them. Adapt the a_i of all processes so that they inlude the settings of the necessary valves and switches, giving a_i^*. The process pumping the water out of the tumbler, for example, will require the tap to be closed, the drain to be pened and the pump to be switched on. **From these adapted specifications,** $A_n \rightarrow C_n$ **can no longer be proved. This is where the control program comes in: Either take a given control program or specification of the control pogram** X **(if we do a posteriori verification), or find a suitable specification** X**, and prove:** $a_1^* \rightarrow c_1, a_2^* \rightarrow c_2, ..., a_n^* \rightarrow c_n, X \models A_n \rightarrow C_n$

3 Non-Monotonic Refinement

In order to understand the contribution and purpose of Non-Monotonic Refinement it is useful to contrast it to the classical refinement approach (see, e.g. [7] that we call here Monotonic Refinement. Monotonic Refinement was developed for programming languages that are anyway already formal. It works under the following preconditions:
The specification

- Is given and cannot be changed, at least not considerably,
- is complete (specifies all relevant properties),
- is consistent (contradiction-free),
- is realisable in the solution domain.

Examples are mathematical functions to be impemented as programs for a universal machine. Monotonic refinement is efficient, theoretically sound, and works very well. But only if a definitive, implementable specification is given in the beginning.

Assuming that such a specification is not given we choose the approach of non-monotonic refinement, which is described here in a formal way.a

Whichever languages, methods and tools are used for formal verification, the essence of the formal part of verification of embedded systems consists in finding out by means of computer support whether

$$P \wedge X \models S \tag{1}$$

where $S = (A \rightarrow C)$, the specification of the goal, is a formula defining a desired property of the artefact under discussion, X is a specification of the control program, and $P = (a_1 \rightarrow c_1, a_2 \rightarrow c_2, ..., a_n \rightarrow c_n)$ is a set of formulae specifying the behaviour of the physical parts of the artefact that are to be controlled.

Typically, we have also a number of domain properties and physical laws that are necessary to prove 1 which are collected in a formalisation N and we get the form:

$$N \wedge P \wedge X \models S \tag{2}$$

In order to achieve a provable representation of 2 we start with a general form of 2, choose a decomposition of P and :

- weaken C, when seeing that the system can only realise weaker overall requirements;
- strengthen A, when recognising that stronger assumption about the overall environment are needed;
- strengthen c_i, when understanding that the components meet stronger requirements;
- weaken a_i, when understanding that the component has to work in a less co-operative environment.
- strengthen N, when more domain properties and physical laws are identified that are necessary to prove 2

We iterate the steps above, and, recursively apply it to the components, until we can prove 2

4 Guidelines

The more formal description of non-monotonic refinement in the previous section needs a number of guidelines for applying it practically.

4.1 How to start NMR

We try to start with the most general statement we can think of, e.g., the Lego sorter sorts all blocks, the coffee machine makes coffee, the batch plant produces batches, the airplane can brake. It is important not to think of possible restrictions right in the beginning. The danger is that by focussing on a certain solution and restriction we reduce the space of possible solutions before we have understood the problem properly. When we go down the decomposition into the details of components the restrictions weakening our requirement will appear in a natural way.

Often, we know that there are timing aspects in the requirement, and we would like to express something like “the coffee machine produces coffee as soon as possible”. The way we deal with such vague timing requirements is that we formulate it in the strongest possible way: “the coffee machine produces coffee immediately”. During the refinement steps this requirement will be weakened,

when we learn, e.g., that the steam boiler needs a warming up phase of 15 seconds, and therefore we will weaken the requirement to "the coffee machine produces coffee 15 seconds after the request button was pushed." This allows to trace precisely at which point and from which component a restriction came in.

4.2 Decomposition

The goal of decomposition is to identify the appropriate components, their interfaces, and requirements. In the standard case we will model a system of a type that we are familiar with, i.e. we are doing *incremental design* in the sense that we design a model which is a variation on known models. Based on experience and education we observe a structure of the artefact and decompose it into components defined by the structure. When we are exposed to *radical design*, i.e. we have to model a system that we are not experienced with, the appropriate decomposition has to be detected. In this kind of model design, the strategy to follow is to first choose one component and model it according to the other steps discussed in full detail. When having identified the assumptions and requirements for one component, we can choose another one, and in this way build the model. The washingmachine example from section 2 is performed in this way.

4.3 Interfaces

When decomposing we fix the components and their interaction. Typically, interaction is described at the interfaces of the components: which inputs do they get from other components and which outputs do they produce for other components. Of course, also the description of interfaces depends on the description language. We often use as description language first order logic. The interfaces are here only shared variables that are used by different components. By conjunction of components the shared variables identify the same phenomena, and hence the interfaces remain somewhat implicit. An explicit representation of interfaces, like e.g. in problem diagrams, may have the advantage of more clarity.

4.4 Focussing on a component

Typically, after decomposing we elaborate the components. This might be in cooperation with a domain specialist. We start with a component that has to satisfy a requirement. Often, when elaborating one component, we notice that we cannot make true the original requirement, but due to new knowledge and insights we see that (only) a different requirement can be made true. This is a normal phenomenon in system design and modelling, and a practical method should cope with this phenomenon. The new requirement has to be shifted back to the revious decomposition level and the effect on other components has to be addressed.

4.5 Weakening

Typically, weakening comes in by physical properties of the components. Examples here are the washing machine and the coffee machine from above, and the computation of a function that we will elaborate here. The essence of this example is the difference between a function in the mathematical world, and a machine implementing this function in the physical world. Assume we have to model a part S_f of a bigger system that has to calculate the function f. We also assume that the machine S_f has an interface consisting of input points and output points where we can observe values at all possible moments. Formally, we define a predicate obs stating that at a certain moment (here: t) at a certain interface point a certain value occurs and can be observed. As a first specification of the machine S_f we get:

DEF $\mathsf{S}_f := \forall t : \mathcal{TIME}.\forall v : \mathcal{R}obs(t, input, v) \rightarrow obs(t, output, f(v))$

Saying, at all moments of time t, if we can observe a real number v at the input, then we can observe the value f(v) at the output of Sf, immediately. When we want to realize this machine we will come across the following restrictions:

- The computer will sample its input with a period δ.
- The machine can only handle input values in the range [minreal..maxreal].
- Using the machine arithmetic, the result will be inaccurate.

Therefore we cannot really implement specification S_f. We can only implement a part P_f where:

- Time is only considered in a multiple of the sampling period δ.
- Input values have to be in the range [minreal..maxreal].
- We get only a value g in an ϵ-interval of the intended value $f(v)$.

DEF Pf:= t: N*. v:[minreal..maxreal]. obs(t, input, v) (g:[f(v)-..f(v)+]. obs(t+, output, g))

4.6 Assumptions

An assumption is any statement that we take for granted without further proof. Assumptions can be, e.g., about the environment, about physical laws, or about the behaviour of parts of the system. Some of the assumptions go into the model, some not. In [6], we classified the assumptions according to different criteria.

4.7 Stopping Refinement

We stop with refinement when:

- the control phenomena between control and plant (signals of sensors and actuators) are addressed. It is the (checkable) real interface between plant and control.

– a sufficient level of detail describing the plant is reached. This is an argument that domain experts have to decide on. An example is a valve, where a valve can be modelled as a boolean variable, or a continuous process where flow decrease (increase) is described by a function over time. Which of the both possibilities is the adequate one depends on how relevant little amounts of fluid are in the process.
– we agree on that we we can take all preconditions, assumptions and laws appearing in the theorem for granted, and do not require further proof of these.

5 Example

In this section we give a more detailed example. We assume that the plant – in this case a coffee machine (Fig.1) – is given in reality, but not accompanied by a complete formal model. We assume that the control software has to be designed. We will explain how a formal model of the "plant" and a control specification can be developed in such a way that one can prove that they together satisfy the requirement.

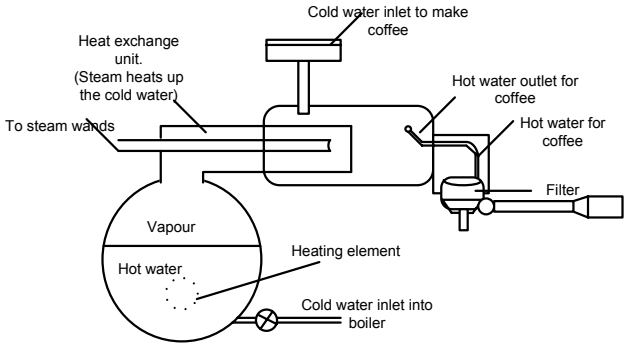

Fig. 1. The coffee machine blueprint.

For formal verification the descriptions should be written in a suitable formal language. If we choose to use a theorem prover to verify the system, the choice would be propositional logic. If we use a model checker we might use process algebra or automata.

We first have to translate a customer's wish into a system requirement. It is necessary to start with the most general form of the requirement that we can think of, in order not to restrict the solution space too early. In this case the requirement is:

R_0 := *"When a user pushes the 'coffee' button, a cup of coffee is poured out as soon as possible."*,

and the first instance of the correctness argument is:

$Coffee\_machine \longrightarrow R_0$,

where *Coffee_machine* is the desired behaviour of our coffee machine.

At this moment, we know that the coffee machine does two things: it heats certain amount of water and draws it through the filter with grounded coffee. Therefore we have the following atomic propositions in the model of the coffee machine behaviour:

$Heat :=$ *"Heat water"*
$Draw :=$ *"Draw water through the filter with grounded coffee"*

The new instance of the correctness requirement is:

$Heat \wedge Draw \Longrightarrow R_0$

Let's look closer at the $Heat$ process. There are three (sub)processes that contribute to it. Water is heated in the heat exchange unit (HEU) with steam ($Heat\_Water$). In the boiler, water is heated to provide steam ($Provide\_Steam$). Water for the HEU is taken from the cold water reservoir ($Provide\_Water$). When necessary, the user has to pour water into the boiler and in the cold water reservoir.

$Provide\_Steam :=$ *"Assuming there is hot water in the boiler, the water is heated in the boiler"*
$Provide\_Water :=$ *"Get 50-60ml water from the reservoir into the heat exchange unit (HEU)"*
$Heat\_Water :=$ *"Heat the 50-60ml water in the HEU to 92-96^{o}C"*

The new instance of our correctness requirement is now:

$Provide\_Steam \wedge Provide\_Water \wedge Heat\_Water \wedge Draw \Longrightarrow R_0$

We continue describing the three subprocesses with more details. If the water in the boiler is kept at the temperature just below the boiling temperature (represented with the $Boiler\_Hot$ proposition), then, when the user starts the process of making coffee, i.e. when he pushes the coffee button (represented with the $Push\_Button$ proposition), the steam will be provided in 6 seconds ($Steam\_6s$ proposition). The $Provide\_Steam$ process is refined as follows:

$Provide\_Steam := Boiler\_Hot \Rightarrow (Push\_Button \Rightarrow Steam\_6s)$

Water for the coffee is taken from the cold water reservoir with a pump. If there is enough water in the reservoir (Res_Water), after the pump $P1$ is turned on ($Pump\_P_1\_On$), after 2.5 seconds, 50-60 ml of water will be in the HEU ($HEU\_5060ml\_Water\_2.5sec$):

$Provide\_Water :=$

$$Res\_Water \wedge Pump\_P_1\_On \Rightarrow HEU\_5060ml\_Water\_2.5sec$$

From the domain expert we found out that if the HEU is provided with steam (HEU_Steam) and if there is 50-60ml of water in the HEU (HEU_5060ml_Water), after 11 seconds, the water will be heated to the temperature of 92-96^{o}C.

$Heat\_Water :=$

$$HEU\_Steam \wedge HEU\_5060ml\_Water \Rightarrow HEU\_9296C\_Water\_11sec$$

Let's look now at the *Draw* process. If there is a 50-60 ml water of temperature 92-96°C in the HEU ($50\_60\_ml\_water\_92\_96\_C\_HEU$) and if there is a grounded coffee in the filter ($grounded\_coffee\_filter$) and if the valve is open ($valve\_open$) and if the pump P_2 is on ($pump\_P_2\_on$), then after 6 seconds – the coffee will be poured out ($coffee\_out\_6sec$) and there will be old coffee in the coffee filter ($old\_coffee\_filter\_6sec$).

$Draw :=$
$50\_60\_ml\_water\_92\_96\_C\_HEU \wedge grounded\_coffee\_filter$
$\wedge\, valve\_open \wedge pump\_P_2\_on$
$\Rightarrow coffee\_out\_6sec \wedge old\_coffee\_filter\_6sec.$

The processes described above take time and they take place in different physical components. Providing steam and taking water from the reservoir can be done in parallel. (This is the decision on how we will design the control.) After steam and water are provided, the heating and drawing water continue sequentially. The new requirement is:

$R_1 :=$ *"When the user requests coffee, a cup of coffee is poured out in 23 seconds."*.

The Control. Now that we described the desired plant behaviour, we write the specification for the control that will ensure it. The control will ensure that pumps are turned on and off and that valves are open and closed in appropriate times. The correctness argument becomes:

$$Provide\_Steam \wedge Provide\_Water \wedge Heat\_Water \wedge Draw \wedge Control \longrightarrow R_1,$$

When the process is started, in this case when the button is pushed, the control should enable that there is a steam in the HEU. If the control keeps the valve open, there will be steam in the HEU after 6 seconds. We will use here the operator $\bigcirc$ where $\bigcirc^{xsec}$ means "after x seconds" and interpret the statements below with an implicit "always".

$$\begin{aligned}
&push_button \Rightarrow \bigcirc^{6sec} steam_in_HEU \\
&\wedge\ \ pump_P1_on \Rightarrow \bigcirc^{2.5sec} 50-60_ml_cold_water_HEU \\
&\wedge\ \ steam_in_HEU \wedge 5060ml_cold_water_HEU \\
&\qquad\qquad \Rightarrow \bigcirc^{11sec} 50-60ml_water_92-96C_HEU \\
&\wedge 50{-}60ml_water_92{-}96O^{o}C_in_HEU \wedge fresh_coffee_in_filter \wedge valve_V_1_open \wedge \\
&\qquad pump_P_1_on \Rightarrow (\bigcirc^{6sec} coffee_poured_out \wedge \bigcirc^{6sec} old_coffee_in_filter) \\
&\wedge\ \ Control \\
&\Longrightarrow (Boiler_hot \wedge Res_water \wedge Push_Button_coffee \Rightarrow \bigcirc^{23sec} coffee)
\end{aligned}$$

The previous expression can be simplified by choosing the right timing instantiations into:

$$\begin{aligned}
&push_button \wedge \\
&\bigcirc^{3.5sec} pump_P_1_on_2.5_sec
\end{aligned}$$

$$\wedge \bigcirc^{17sec}(fresh\_coffee\_in\_filter \wedge valve\_open\_6sec \wedge Pump\_P_2\_on\_6sec)$$
$$\Rightarrow \bigcirc^{23sec}(coffee\_poured\_out \wedge old\_coffee\_in\_filter)$$
$$\wedge Control$$
$$\Longrightarrow (boiler\_is\_hot \wedge Res\_Water \wedge Push\_Button \Rightarrow \bigcirc^{23sec} coffee)$$

We can continue now adding process of removing the old coffee from the filter and bringing the new, freshly grounded coffee to it. Further on, we refine our correctness argument, until we can prove the statement.

By writing down all the instances of the correctness theorem, we also document modelling decisions made. In this small example we documented every refinement step, but it is up to the modeller to decide what he will document and what refinement steps he will leave undocumented.

6 NMR Related Work

Zave and Jackson [12] pointed out that requirements are about the environment, not the software. Starting from this idea, Jackson proposed the problem frames technique [5]. With the problem frames technique, one starts looking at the system with regard to the combination of the machine (software) and the plant that together have to satisfy the requirements. Instead of designing the machine immediately, the software designer has to first formulate the requirement for the plant,decompose the plant to the domains, and identify the phenomena on the domain interfaces, relevant for the requirement.

Heissel addressed the problem of modelling method and proposed *agendas* [4], 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.

Seater and Jackson propose the requirement progression [9] – a technique for obtaining software specifications from the requirements. It extends the problem frames (PF) technique [5] by adding the means to write down the correctness argument formally. Here, the correctness argument justifies re-formulations of the requirement stepwise, until it becomes the requirement for the machine interface.

Goal-oriented requirements acquisition in the KAOS (Knowledge Acquisition in autOmated Specification) method [10] starts from high-level goals and refine them to subgoals that have to be achieved by a composite system. This methodology also views the system as combination of the software and its environment (although here, different terms are used for the plant and its components [11].) This method is designed for requirements elicitation and refinement, so it ends before the phase of formal specification and verification.

Tropos methodology is used to support software development in early requirements, late requirements, architectural design and detailed design phases [8]. Again, the system here is described through goals to be achieved by it. The i* model used here describes actors (agents, roles or position.) In "Formal Tropos" [1], one can define the circumstances under which a given dependency among two actors arises, as well as the conditions that permit to consider the dependency fulfilled.

The idea to formally describe some modelling process aspects was proposed by Gunter et al. in [2]. They defined a reference model for requirements and specifications that consists of domain knowledge, requirements, specifications, programming platform, and the phenomena in the system, its environment, and on their interfaces.

Hall, Rapanotti and Jackson developed a formal conceptual network based on problem-oriented perspective [3] 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. Their aim is twofold - to provide structure to modelling process and to give foundations for accompanying tool.

Zowghi and Offen [14] described the logical framework for modelling and reasoning about the evolution of requirements, from high-level and imprecise wishes of stakeholders to a more complete requirements model. The requirements engineering consists of the steps that can be described with monotonic reasoning (classical logic), and the steps where new information about the requirements, machine and the plant is added, which can be described using nonmonotonic methods for reasoning. With their work, the authors give basis for defining methods and tools for management of changing requirements. In later work Zowghi and Gervasi [13] investigated the non-monotonic refinement when eliciting requirements and checking their consistency, correctness, and completeness. They also described which kind of proofs must be carried out at each step during the evolution of requirements to ensure that the final specification of software system satisfies the business goals of the customer.

7 Discussion and Conclusion

We suggested a modelling process that starts at a very early point of model construction, when the requirements are only roughly known, and we do not know which parts and properties of the system are relevant to satisfy the requirements.

The purpose of models we construct is verification, the class of artefacts that we are interested in is embedded systems.

Our method is a top-down approach that guides the compilation of knowledge about a system that must be included in the model, based on non-monotonic refinement (NMR). What we did *not* suggest here is an algorithm that constructs a model, nor a modelling language - for the paper here we use logic, any other language could be used, textual or graphical.

We conclude with the discussion of a few features of the method.

Growth of Knowledge

One of the key properties of NMR is that the growth of knowledge is built into the process of modelling. We believe that in this way the "natural" way of modelling is supported by a method. We start with the most general form of a verification requirement, saying Plant and Control have to satisfy a Requirement. For the Requirement we take the most general thing that we can think of, avoiding unnecessary restrictions before we have understood the problem sufficiently. In a refinement step we learn more about components and the additional assumptions and constraints that come with these components.

Top-Down vs Bottom-Up Thinking

In the first place, we consider an artefact and a model of the artefact, and the modelling process that leads from the first to the latter. However, if we look closer at the process, there is another model involved, our epistemological model of the system. This is our perception of the system and how it is working, and it is possibly distributed among a number of domain experts. The epistemological model is built up by looking at the system, watching it, playing with it, reading documentation, talking to domain experts, etc. We claim that the epistemological model is gained by mainly bottom-up thinking. We cannot say much about this process, apart from that is personal, depends on experience, but also a number of non-rational aspects as association and thinking with analogies. Moreover, this epistemological model remains in the first place implicit. When we construct a model, we take our epistemological model as basis. i.e., in the end we do not model the artefact, but our insight how it works and how it is composed. Non-monotonic refinement is organized in a top-down manner. We need a fragment of a epistemological model to start: otherwise we cannot say anything about a system, if we do not know what it does at all. The top-down development helps to shape the modelling process. We stepwise take more details into account that are relevant to prove the requirement, which makes the process goal-driven. On the way it can be the case that we have to enlarge the epistemological model: when we see that more knowledge about a component or process is necessary. In this way we make use of the knowledge that is relevant to satisfy the requirements, and leave the rest. Our conjecture here is that NMR helps to find "minimal" models (without diving into the formal details of metrics of model size).

Modelling vs. Design, Fragment vs. Overall System

NMR is a very general approach and can be used for both, design and modelling. In design we have a goal what an artefact should do, and we stepwise take decisions how this goal can be realized. After each decision, e.g. use component uvw from supplier xyz, we collect the restrictions and assumptions coming with this component, include it in the next refinement of our verification requirement, and continue. In the modelling process, we do not take a design decision, but we observe that component uvw from supplier xyz is used, and for the rest we

continue as above. However, we use the approach typically for two settings of embedded systems: in the first, we assume both, plant and control are given, in the second we assume the plant is given and we have to design a control. In the first case we can use NMR to derive a plant model, we can use formal methods to transform the control code (that is already a formal object) into a model, and verify the composition of both. In the second case, we construct a model of the plant, and try to find the control model that allows to prove the verification requirement. The second version fits perfectly in the approach of model based design: after having proven the verification requirement, we can take the control model to automatically derive control code. There will be a difference in the size of the models, depending on in which setting we are: for the first one (everything is given) the requirement to prove typically is more restricted, and the models needed are smaller. If we want to design the control, we need some sort of completeness of the model, which typically requires models of bigger size.

Vagueness One of the problems in the modelling process is to deal with vagueness. When the process is finished and we have derived a formal model all vagueness should be eliminated - this is why we want formal models. In the beginning of the modelling process, vagueness is present: we do not know precisely the requirement that we will be able to prove, we do not know about the details of the system, or what details are at all relevant for our requirement.

There are two dangers when dealing with vague and incomplete information, hidden ambiguity and too early formalization. The first leads to unintended interpretations, the other one forces preciseness at a point when the system and requirements are not yet precisely understood.

NMR uses two strategies to deal with vagueness. On one hand phenomena are addressed only when they are understood, which is supported by the top-down approach. On the other hand, the iterative weakening of requirements describes the narrowing of the solution space as we get more information.

Acknowledgement. The authors thank Roel Wieringa for useful discussions and comment.

References

1. Tropos project homepage. *http://www.troposproject.org/*.
2. C.A. Gunter, E.L. Gunter, M.A. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.
3. Jon G. Hall, Lucia Rapanotti, and Michael Jackson. Problem oriented software engineering: A design-theoretic framework for software engineering. *sefm*, 0:15–24, 2007.
4. Maritta Heisel and Jeanine Souquières. A method for requirements elicitation and formal specification. In Jacky Akoka, Mokrane Bouzeghoub, Isabelle Comyn-Wattiau, and Elisabeth Métais, editors, *Proc. of the 18th International Conference*

on Conceptual Modeling, volume 1728 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 1999.
5. M.A. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2000.
6. J.Marincic, A.Mader, and R.Wieringa. Classifying assumptions made during requirements verification of embedded systems. (accepted for publication in the proceedings of the International Working Conference on Requirements Engineering (REFSQ) 2008).
7. G. Rozenberg J.W De Bakker, W.P de Roever, editor. *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*. Number 430 in LNCS. Springer Verlag, 1989.
8. Roberto Sebastiani Paolo Giorgini, John Mylopoulos. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artifcial Intelligence*, 18/2, march 2005.
9. R. Seater, D. Jackson, and R. Gheyi. Requirement progression in problem frames: deriving specifications from requirements. *Requir. Eng.*, 12(2):77–102, 2007.
10. Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE '01: Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, page 249, Washington, DC, USA, 2001. IEEE Computer Society.
11. Axel van Lamsweerde and Emmanuel Letier. From object orientation to goal orientation: A paradigm shift for requirements engineering. In *Radical Innovations of Software and Systems Engineering in the Future*, pages 325–340, 2002.
12. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.
13. Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information & Software Technology*, 45(14):993–1009, 2003.
14. Didar Zowghi and Ray Offen. A logical framework for modeling and reasoning about the evolution of requirements. *re*, 00:247, 1997.