Obtaining Formal Models through 
Non-Monotonic Refinement

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

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

Abstract. When designing a model for formal verification, we want to be certain that what we proved about the model also holds for the system we modelled. This raises the question of whether our model represents the system, and what makes us confident about this. By performing so-called, non-monotonic refinement in the modelling process, we make the steps and decisions explicit. This helps us to (1) increase the confidence that the model represents the system, (2) structure and organize the communication with domain experts and the problem owner, and (3) identify rational steps made while modelling. We focus on embedded control systems.

1 Introduction

Modelling for formal verification is performed while verifying (or designing and verifying) a system with formal (mathematical) methods. If we want to prove certain properties of a system, we have to formally describe (i.e., model) the system, as well as the properties we are verifying.

Figure 1 shows these activities: We would like to know whether some property holds for the system; the system and the property we are investigating are represented by the lower two blocks on the diagram in 1. The only thing that we can do with formal verification is check whether the formalized property holds for the system model; this is represented in the upper part of the Fig. 1.

If we want to be sure that the system model tells us something about the properties of the system, it has to represent the system in such a way that all the things relevant for the property are present in the model. However, we can

* Supported by the Netherlands Organisation for Scientific Research (NWO), project 600 065 120 241420, Modelling Control Aspects of Embedded Systems
** Supported by the Netherlands Organisation for Scientific Research (NWO), project 632 001202, Methods of Modelling Embedded Systems
never know whether the descriptions of all the things relevant for the property are present in the model; this is one of the problems when modelling. We can only be more or less confident that the model represents the system with respect to the property.

![Diagram](image)

**Fig. 1.** The model is a formal description of the system. It can be formally verified or checked if it has the formalized properties.

The description of the system and its environment are not given to us as formal descriptions from which we could easily extract things necessary for the model and the formal properties. They are given vaguely, in a pile of technical manuals and the wishes of different stakeholders. Bringing them to a formal model and formal properties precedes the verification and is not easy. Before letting the powerful mathematical methods and tools check the properties of the model, the model has to be designed and modelling decisions have to be justified, so that we are confident that we prove something about the actual system.

Although creative, the modelling process consists of certain rational steps and results from the education of how to model and previous experience. Still, more effort is spent to develop new languages and techniques, without guidelines on how to use the existing ones in a systematic way.

To increase confidence in the model and identify the rational steps behind the modelling process, we obtain the model using non-monotonic refinement (NMR). This way we make explicit what model designers actually do when they systematically design the model.

In short, modelling using non-monotonic refinement consists of decomposing the system and the property simultaneously, (formally) describing the parts only as detailed as necessary and with as much that we know about them at that moment, and documenting our descriptions in the verification requirement formula. We are updating our knowledge in a non-monotonic way. This forces us to focus on relevant system details rather than unnecessary ones, and is therefore useful in communication with domain experts. Also, by documenting the instances of the verification requirement we hope to contribute to identifying those steps in modelling that are not creative, but rational and possible to be taught to other
people. We used NMR on a small chemical engineering example [6]. Additionally, for many years at the Univ. of Nijmegen in the Netherlands, students have been (successfully) taught to perform modelling and theorem proving in such a way that they use non-monotonic refinement.

This paper describes the process of obtaining a formal model using non-monotonic refinement and explores how it increases our confidence in the model.

We focused on the embedded control systems in our research; we use the term system to describe them here. However, the claims written so far and many of the claims that we will offer, also hold for the modelling process in general and for systems other than embedded control systems.

Section 2 introduces the reader to the basis that we begin with and to the work related to these concepts. In Sect. 3 we explain the process of non-monotonic refinement (NMR). Section 4 shows how NMR works on small examples. In Sect. 5 we point out results of this way of obtaining a formal model and plans for the future work.

2 Basic Concepts and Related Work

Figure 2 shows our view of an embedded system - a plant and a controller. Everything outside the plant that influences the system is called environment.

![Diagram of an embedded system](image)

Fig. 2. Embedded system

The properties we are proving are the system requirements (not the software requirements) though, as we explained in earlier work [5], we do not strive for completeness of the system requirements but choose certain properties of the system to prove formally. This way the model is not too complex for the computer aided verification tools.

Because the requirements always refer to some phenomena in the plant, when designing the control software or just verifying it, we need to describe the parts and aspects of the plant as well. The idea to describe the requirements in the problem world and describe the environment of the software being designed is explained by Jackson and Zave [4,11]. We apply these ideas when modelling for the purpose of formal verification. When describing the plant, we describe its behaviour.
2.1 Verification Requirement

The design logic of a system can be described with the following formula:

\[ S \land E \Rightarrow R, \]  

(1)

where \( S \) is the description of the system, \( E \) is the description of its environment, and \( R \) represents the requirement that our system has to satisfy.

This formula was brought up by different authors as a logical statement that describes the relationship between the system being (or already) designed, its environment, and the requirements that we want the system to satisfy. Roozenburg and Eekels [9] present a form of this statement in a broader context of the product design. Gunter et al. [1] defined a reference model for requirements and specifications that consists of the following elements: domain knowledge, requirements, specifications, and a programming platform; and the phenomena in the system, its environment, and on their interfaces. Finally, Wupper and Mader [6, 10, 3] developed the method of the non-monotonic refinement, discussed further here.

These are all independent claims developed in more or less the same period of time; what they all have in common is that they describe in logic something that designers or verifiers have in mind when they design, model, and verify a system: ‘The system, the environment and the requirement should fit together properly’ [4].

Formula (1) has different names and can contain more or less details about the system, its environment, and the requirement. It can also describe different aspects, depending on what goals and on which implementation level we are focusing. In this paper we refer to it as to a verification requirement. The difference between the work of a designer and a verifier is that the first designs ‘\( S \)’ in the formula (1), while the latter is busy turning the verification requirement into a verification theorem and proving it. When a formal verification expert formally verifies that a property holds for the system model, he actually proves that the verification theorem written in form (1) holds.

2.2 NMR Related Work

Zowghi and Offen [13] 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. In later work Zowghi and Gervasi [12] 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.

This work is similar to the work of Wupper and Mader [6, 10, 3]. The difference here is that we do not use the non-monotonic refinement to elicit requirements from high level requirements, for example business goals. Our goal is to
design a formal verification model starting from the requirements that refer to the plant behaviour. These authors also keep the framework more general and leave the possibility of using both formal and informal methods. We on the other hand are using the NMR to design a formal model. However, in this process we also have a communication with the problem-owner and domain experts.

3 Modelling Using Non-Monotonic Refinement

Before explaining non-monotonic refinement, we will first explain monotonic refinement, a well developed area in formal methods.

3.1 Monotonic Refinement

Monotonic refinement works as follows: we start from a given verification requirement containing requirements and specifications that are not doubted. We derive an intermediate result by refining initial specifications. The properties of the intermediate result have to imply the original specifications, which makes the refinement monotonic. The intermediate result is further refined into another one, again in the monotonic way. This continues until we have specification with enough details to perform the verification. An example of monotonic refinement is mathematical functions to be implemented as programs for a universal machine.

Monotonic refinement works under the precondition that the specification is
- given and cannot be changed, at least not considerably,
- complete (specifies all relevant properties),
- consistent (contradiction-free),
- realisable in the solution domain.

Monotonic refinement is difficult to achieve in practice, if not impossible. As Parnas and Clements explain [8], what can be done is to fake this rational process, by first finding the solution and than representing it as a set of monotonic refinement steps. But this faked process is not the real process followed. With the non-monotonic refinement we try to make explicit what designers and verifiers actually do.

3.2 Non-Monotonic Refinement

When performing a formal verification, we first have to design a verification model of the system. To do this, we define the requirement we are proving, what parts of the system are relevant to be represented in the model, and under which assumptions our model is meaningful. In the process of the model design we also learn about our system and get new insights. We usually have to decompose the system, so we could say that we also design the system structure.

While designing a verification model, we can describe and document each stage with the verification requirement. It contains the system description, the
requirement we want to prove, and the assumptions to be fulfilled. We start with
the more informative form of (1):

\[ A \implies (P \land C \implies R), \]  

(2)

where \( P \) is the description of the plant behaviour, \( C \) is the description of the
controller, \( R \) is the description of the requirement, and \( A \) are the assumptions
to be satisfied.

The goal is to refine this formula, until we get a mathematical theorem that
can be proved (or disproved). We refine \( A, C, P, \) and \( R \) as well as the whole
formula.

The formula (3) documents our description of the system and formalization
of the requirement. In this process we decompose the plant and describe each
component (i.e., domain). The decomposition of the plant into domains directs us
further to decomposition of the requirement and mapping the sub-requirements
to the system parts. If we design the control, it often happens that we make
certain decisions about it, while describing a domain.

When modelling, we also find assumptions about the domains and the control
under which the requirement will hold. Therefore, we refine the verification
requirement by refining \( A, P, C, \) and \( R \). We also refine the whole formula in
the sense that we decide what assumptions to put into the model and what will
remain written informally, as a label that will come together with the verified
system stating the conditions under which the system works.

The next instance of the verification requirement (3)

\[ A_1 \implies (P_1 \land C_1 \implies R_1), \]  

(3)

may or may not be monotonically refined from the previous version. The
process of refinement of each the component in this formula is such that it can
contain monotonic refinement, or correction of errors (which results in some
changes) or an update of our knowledge about it or specifications of the parts we are designing.

When we start decomposing the system, we do not know everything about
the plant; NMR is a structured way to refine requirements and to describe the
domains and the control without putting more details than necessary.

The non-monotonic refinement (NMR) is the way to refine the verification
requirement in such a way that its new instances do not necessarily have to
imply the previous instances of the verification requirement. This helps to

– choose the right abstractions for a formal model that can be used for verifi-
cation,
– define the boundary between the artifact and its environment,
– find a meaningful decomposition,
– not mix up levels,
– make design decisions not before they are really understood,
– define interfaces when it is understood why they are necessary rather than
  at the beginning of the process,
– document all decisions in a logical place,
– understand the impact of design decisions on the overall properties.
4 Examples

In this section we show two examples of modelling for formal verification using NMR. The first is the example of designing and verifying the control for the small plant. The second one is a design of a coffee machine and is one of the tasks that students worked on. In both cases we demonstrate our approach by describing the systems with the predicate logic and the intention to verify the model with a theorem prover. The first example was also described [7] with timed automata and formally verified with the Uppaal model checking tool. However, we want to stay independent of the tools and formal techniques and contribute to model design, whichever technique or tool is used.

Lego Sorter. Our first example is a small PLC-controlled plant made of Lego bricks, DC motors, angle sensors, and a colour scanner. Figure 3(a) shows a top view of the plant and Fig.3(b) a side view. Bricks are stored in a queue. Two wheels at the bottom of the queue move the bottom brick to the conveyor belt. Bricks are transported by the conveyor belt to the scanner and further on to the sorter. The scanner senses the colour of a brick. The sorter consists of two fork-like arms. Each arm can rotate a brick to one of the sides of the plant. Bricks enter the belt one after another; it is possible to have more than one brick on the belt.

The requirement is to sort bricks according to their colour. The plant also has a small program running on it; in our exercise we want to design a new control. To do that we first describe the plant and specify the control.

The wheels of the queue and belt are coupled; a single motor is moving them. The scanner is a sensor that 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 an unknown state. 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, the arm rotates through 360 degrees.

The detailed description of this example is available in an online technical report [2]. Here we present some of the steps.

We start with a simple, general version of the property to be verified:

\((R_1)\): When a yellow or blue brick is offered, it will be sorted into the appropriate side of the plant.

The first instance of the verification requirement is:

\[ A \implies (P \land C \implies R_1), \tag{4} \]

where \( P \) is the description of the plant behaviour, \( C \) is the description of the controller, \( R_\downarrow IS \) is the description of the requirement, and \( A \) are the assumptions to be satisfied.

The next step is to use expert knowledge about the plant and the control to identify a sub-process or components that somehow
contribute to this goal. The result of this is the identification of the following processes: sorting a brick, scanning, transporting a brick to the scanner, and transporting a brick to the sorter.

We call this decomposition process decomposition. There are many other possible decompositions, for example the decomposition to physical parts (instrumental decomposition), functional etc. Sometimes we have to do two or more different decompositions to identify the relevant information about the system.

For each process, we found a component in the physical plant that is used to perform it. The first two processes correspond to queue, sorter and a scanner respectively, while the last two are executed by the belt.

The requirement is refined to $R_1 = R_{11} \land R_{12}$: When a yellow or blue brick is in the queue ($R_{11}$), it ends up on the appropriate side of the plant ($R_{12}$). The assumption $A$ becomes $A_1 = Offered$, the assumption that a brick is placed into the Queue.

The verification requirement at this stage can be elaborated into:

$Offer \implies (\text{Sorting} \land \text{Scan} \land \text{TrToSc} \land \text{TrToSrt}) \land \text{Control} \implies R_{11} \land R_{12},$  

(5)

where $Scan$, $Sorting$, $TrToSc$, and $TrToSrt$ are formal descriptions of the assumptions and processes of offering bricks, scanning a brick, sorting it, transporting a brick to the scanner, and transporting a brick from the sorter to the scanner (yet to be found).
For each component $c_i$ that we identified, we write a formula $a_i \implies c_i$, where $a_i$ is the assumption about the component and $c_i$ is its specification. The formula contributes to $R_1$ but does not contain any more knowledge than necessary to contribute to the overall goal. This helps to avoid thinking about the details that are irrelevant for the problem and that leads us to a minimal model.

Due to lack of space we elaborate the refinement of the verification requirement in the example of the sorting component description. For complete documentation, we recommend the reader to review the online technical report [2].

First we describe the following:

If a yellow (blue) brick is in the sorter (assumptions $a_{s1} \land a_{s2}$), then if the left motor is turned on, after a certain time, the yellow (blue) brick will be sorted to the left (right) side of the plant (description $c_{s1}$)

This refines Sorting to $Sorting_1 = a_{s1} \land a_{s2} \implies c_{s1}$ in the formula 5.

Further on, we increase our knowledge about the sorter, by finding out the value ($a_{mo}$) that has to be put into the variable that interfaces the motors and the maximum sorting time ($t_{max}$). The new instance of the verification requirement is:

$Offer \implies Sorting_1 \land Scan \land TrToSc \land TrToSrt \land Control \implies R_{11} \land R_{12}$

(6)

Additional information we get about the domain is that the sorter arm has to make a full rotation; after that a brick is considered sorted. This brings us to the question how the controller will observe a full rotation. From the domain expert, we find out that the angle sensor measures the relative angle change.

So we decompose the sorting process into the $Sorting_1$ that is already described and $Sense\_angle$ process. The last one states the following:

If the left (right) arm is in home position, than the left (right) angle sensor shows the value that corresponds to the home position.

The verification requirement now contains the description of the $Sense\_angle$ process:

$Offer \implies Sorting_1 \land Sense\_angle \land Scan \land TrToSc \land TrToSrt \land Control \implies R_{11} \land R_{12}$

(7)

We talk with the domain expert and find out more about this value. It turns out that these sensors show 16 different values and that it is not possible for the controller to determine whether the arm is in the home position or not. This means that we have to change the description of $Sorting_1$, and to add assumptions of both arms being in home position. Otherwise, the sorting does not work as specified. $Sorting_1$ is refined to $Sorting_2$:

If a yellow (blue) brick is in the sorter and both sorter arms are in home positions (assumptions $a_{s1} \land a_{s2} \land a_{s3}$), and if the left motor is turned on, then after a certain time, the yellow (blue) brick will be sorted to the left (right) side of the plant ($c_{s1}$).
This information also means that the whole plant when started has to have the sorter arms in the home position. So we refine \( A_1 \) in the formula (7) into \( A_2 \) that now states that:

**When the plant is started we assume that a brick is in the queue and the sorter arms are in home position.**

Because we are using a theorem prover, we have to describe some things that seem natural. For the Sorting process description it means that we have to refine \( \text{Sorting}_2 \) to \( \text{Sorting}_3 = \text{Sorting}_2 \land N_1 \), where

\[ N_1: \text{The sorter arms will not move if the motors are turned off.} \]

When describing the sorter, we made a decision about the control. In the formula 7, \( C \) is refined to \( C_1 \):

**The value put to the arm motor is the value that will be constant (we will not change it) and it is such that the processor can read the changes of the angle sensor.**

To distinguish the initial position with the full rotation, we will read the sensor after a few processor cycles, so \( \text{Sorter}_3 \) becomes \( \text{Sorter}_4 \), with this change included.

After describing sorting, we realize that it takes time and that our requirement has to be more realistic, since a block can not be sorted in zero time.

The new, weakened requirement is: **When a yellow or blue brick is in the queue \( R_2 \), it eventually ends up on the appropriate side of the plant \( R_{12} \).**

Whenever we change the description of one of the components in the verification requirement, we can describe this by writing down a new instance of the verification requirement.

At this point we have the descriptions of the sorting process, the angle sensor, some of the assumptions that refer to the whole system, and some initial decisions about the control. The requirement is weakened to what is possible to achieve with the existing plant. This is not enough to prove a theorem, so we continue describing other components one by one, the control as well as refining the requirement. The end result is a theorem that states that each offered brick of yellow or blue colour will eventually be sorted. This theorem has to be proved in a theorem prover.

On this example we can see that the details we are adding while describing domains are:

- Details about each component (i.e., domain) that are a result of increase of our knowledge about the domain.
- Details about the interfaces of this component with other components
- Descriptions related to the tool or technique we are using If we use a theorem proving technique of formal verification, then we have to include some things in the model that seem obvious.
- Decisions about the control
- Assumptions
- Requirements are changed or refined
When designing the control, we may have to weaken the requirement. We add assumptions about the plant and we may also find out that there should be more details in some of the domain descriptions.

**Coffee Machine.** We are specifying a simple machine that will produce coffee. We start as if we have the machine, but don’t know anything about its parts. This exercise is based on a hypothetical example that students made, so to avoid problem shifting, we designate a person who performs the role of the domain expert and a person who plays the role of the customer. Our machine consists of water and coffee reservoirs, a mixing reservoir, warmer, filter, and motors and sensors which consist a plant, and a processor that runs the control. We are describing the behaviour of the plant and specifying the control for it.

In this process (as in the previous example), we had a mixture of monotonic and non-monotonic refinement of requirements and domains. This is expected since non-monotonic refinement does not mean that each refinement step is necessarily non-monotonic. The requirement was refined in both ways, while for the domains we mostly updated our knowledge about the system parts, and decomposed parts into sub-parts.

When designing the control, it is possible that it can be composed of parts in such a way that each has a task of controlling a component in the plant. It is also possible to have a different criteria for sub-controls. Here, we designed the control as a set of sub-controls that are controlling parts identified in the plant. The descriptions of the parts make this process straightforward but not automatic, since there could be some scheduling problems.

We also had to make the decision of which assumptions to include in the model (for example whether to assume that there is always water in the water reservoir, or this is something that will be stated informally).

**We start with the initial requirement:**

> ‘When one (two) cups of coffee is desired and a cup (two cups) is placed on a special place in the machine, the machine will produce one (two) cups of coffee’.

The first instance of the verification requirement is:

\[ A \implies (P \land C \implies R_1), \]  

where \( P \) is the description of the plant behaviour, \( C \) is the description of the controller, \( R \) is the description of the requirement, and \( A \) are the assumptions to be satisfied.

**The next step is to use expert knowledge about the plant and the control to identify a sub-process or components that somehow contribute to this goal.** We identified the following components: Water reservoir, coffee reservoir, heating reservoir with the heater, mixing reservoir with the mixer and the filter, and two buttons. Here, we decomposed the plant into physical parts. In the future, we plan to investigate which decompositions are suitable for which classes of problems.
The verification requirement at this stage can be elaborated into:

\[ A \Rightarrow Water_r \land Coffee_r \land Heating_r \land Mixing_r \land Buttons \land Control \Rightarrow R_1, \]

where \( A, Water_r, Coffee_r, Heating_r, Mixing_r \) and \( Buttons \) are formal descriptions of the assumptions and water, coffee, heating and mixing reservoirs, and the buttons.

For each component \( c_i \) that we identified, we write a formula \( a_i \Rightarrow c_i \), where \( a_i \) is the assumption about the component and \( c_i \) is its specification. The formula contributes to \( R_1 \) but does not contain any more knowledge than is necessary to contribute to the overall goal.

Here again, we describe the components one by one, which resulted in focusing only on that particular component and not modelling other unnecessary details. We also made some decisions about the control, so the control part of the verification requirement evolved with each component description.

In this case we started with the idealized requirement to produce coffee immediately after the button is pressed. After describing components, we find out the times necessary for each component to execute the desired task, so we have to change the initial requirement and define the time for which the coffee should be prepared. Our customer expressed her wishes to have a coffee ready in 10 seconds, but from the time constants provided by the domain expert, it turned out that this goal is impossible to achieve with the system we are able to build. So, we negotiated and changed the requirement.

The control that we designed first produced a single cup of coffee twice in a row when a button for two coffees was pressed. This was unacceptable for our customer and we changed the control so that the system provides two cups of coffee in one batch.

In this case the domain modelling was monotonic, with adding the knowledge about time constants, and values at the controller variables interfacing a temperature sensor and motors. The monotonicity was observed in weakening and changing the requirement, and where it was not possible to change the requirement, we redesigned the control.

5 Conclusion

As a way to obtain a formal model, NMR structures and shapes the argumentation. It reflects the natural increase of knowledge about the system during the modelling process and, at the same time, it allows putting this knowledge in a formal form and context. In this way NMR also structures the communication with domain experts and problem owners by focusing on the relevant aspects that need to be described next.

Formally writing the specifications of domains forces us to be precise. Domain experts and problem-owners usually do not understand the formal language that we (in the role of formal verification experts) use, so at every step we first have to translate from natural language what we know about the domain or the
requirement and then translate our description back into a natural language and check with the domain expert or the problem-owner to see whether what we wrote is true. This way of communication in refinement steps reveals the facts about the domains that we need for the model and the requirements for these domains.

When modelling one of the dangers is to put more detail into a model that we really need to prove the property of interest. The top-down approach in NMR reduces this danger because, at every stage, we only focus on the additional details for the next refinement step. This does not ignore the presence or importance of bottom-up thinking in design. NMR shapes the bottom-up thinking.

As a description of how a model is designed, NMR is useful to identify the rational steps and decisions while modelling. These steps are what we believe designers do, and through NMR we make their steps and implicit decisions (i.e., the tacit rationalism of model design) explicit. The advantage of explicitness is that the process can be more easily taught, is more efficient because it allows the identification of standard arguments and avoids superfluous details, and the whole modelling process is more controllable and justifiable. The latter increases our confidence in the quality of the model and the verification results.

We also hope that identifying some elements of the reasoning behind the model designers’ decisions, can contribute to teaching modelling to other people.

Even if the basic idea of NMR refinement is very intuitive, practising it requires some training and discipline of thinking. This paper elaborated small examples that demonstrate how to apply NMR.

In the future, we plan to examine how to find reusable steps and patterns for certain classes of problems, identify which decompositions are suitable for certain classes of problems, and if and how the control design can be described using the principles shown here.

References