Non-Monotonic Modelling from Initial Requirements

A Proposal and Comparison with Monotonic Modelling Methods

J. Marincic
Department of Computer Science, University of Twente
P.O.Box 217 7500 AE
Enschede, The Netherlands
j.marincic@ewi.utwente.nl

A. Mader
Department of Computer Science, University of Twente
P.O.Box 217 7500 AE
Enschede, The Netherlands
mader@ewi.utwente.nl

H. Wupper
Institute for Computing and Information Sciences,
Radboud University Nijmegen
Nijmegen, The Netherlands
Hanno.Wupper@cs.ru.nl

R. Wieringa
Department of Computer Science, University of Twente
P.O.Box 217 7500 AE
Enschede, The Netherlands
roelw@ewi.utwente.nl

ABSTRACT

Researchers make a significant effort to develop new modelling languages and tools. However, they spend less effort developing methods for constructing models using these languages and tools. We are developing a method for building an embedded system model for formal verification. Our method provides guidelines to build a model and to construct a correctness argument. We start from a high-level formula stating that a plant (a device that performs a task) and its control should satisfy requirements. As our knowledge about the system grows, we refine this formula and the model gradually, in a stepwise non-monotonic process, until we have a description that can be formally verified. In this paper we explain our method on a simple example and compare it briefly with two other methods: requirements progression and the goal-oriented KAOS approach. The requirements progression is an extension of a problem frames approach. The KAOS method is also based on problem frames, but introduces new concepts for describing a system.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.4 [**Software Engineering**]: Software/Program Verifications

General Terms

Design, Verification

IWAAPF'08, May 10, 2008, Leipzig, Germany.

Keywords

correctness argument, modelling method, nonmonotonicity

1. INTRODUCTION

Researchers make more effort to develop new modelling languages and tools than to develop methods for using them [6]. Without a method for model construction, one is left with a lot of existing formal languages and tools, but without guidelines for how to use them.

The problem is that modelling cannot be purely formal. It contains both formal and non-formal (informal, but accurate) steps. We claim that the non-formal steps are not unpredictable or irrational, but are part of educated creativity, following a systematic way of thinking. In our work on modelling we want to make the tacit rationalism of modelling explicit.

We are developing a method for building an embedded system model for formal verification. Our method provides guidelines to build a model and to build a correctness argument. The correctness argument increases our confidence in the model and makes the model and modelling decisions more understandable.

Papers on formal methods present their examples as if they have been derived monotonically. The modelling process is presented as if modellers had all the knowledge about the system before they started modelling. In that case it is possible to build a model in a strictly top-down manner. But, modellers usually do not know everything about a system that they are modelling in advance. Such a monotonic refinement sequence is the result of a long process in which an understanding of the modelling problem was built up.

Our interest is in how to build up this understanding. Therefore, we provide modelling guidelines from the very beginning, when there is a vague idea about the system requirements, and starting from a very high-level view of the system, when decomposition has not already been done.

Most of the formal specification techniques focus on the software alone. But, as Zave and Jackson pointed out in their article "Four Dark Corners of Requirements Engineering" [15], the system requirements are in the software en-

vironment and not on its interface. Their ideas were followed by a number of formal and non-formal approaches in requirements engineering, software engineering, and safety system engineering; and Jackson's problem frames [7] are the non-formal technique and framework that further extend the claims presented in "Four Dark Corners". We follow this approach, too – our models for formal verification aim to prove that the requirements are true of both plant and control.

In this paper we describe our method and compare it briefly with two other modelling methods:

- Requirements progression method [12] and
- KAOS, a goal-oriented method [13].

The first method is the extension of the problem frames framework. It starts at a later point in the process, when the requirements have been fixed. The second method uses different concepts (goals, subgoals, obstacles, and agents). It starts with the general, overall goal and decomposes it in a similar way to our method. Within the KAOS method the overall goal is as general as possible, and if it has to be weakened then all the relevant subgoals and agents have to be changed.

1.1 Terminology

As we focus on embedded control systems, we will use the following terms. Figure 1 shows an embedded system and its parts, as we view them. An *embedded system* consists of a *plant* and a *controller*. The term *plant* is common in real-time control systems, it is the physical device in which the controller is embedded; it means the same as a causal domain. The controller consists of the controller hardware, operating system, and control software (we will refer to the latter as a *control*). When modelling we do not take into account the controller hardware and the operating system – we focus on the plant and the control.

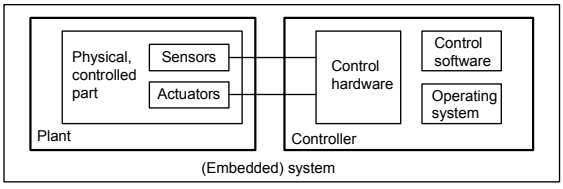

Figure 1: Embedded system

2. NON-MONOTONIC REFINEMENT

We are developing a method for verification model construction [11] [14] [8]. Modelling is a partly formal and partly non-formal activity. Also, it is a partly creative and partly rational activity. We believe that knowing the rational parts of the modelling process, being aware of what we are doing while modelling, helps to make better models.

One of the aspects of the modelling process that we focus on is *non-monotonic refinement* (NMR). This is not the same as the term used in belief revision in philosophy [5] and artificial intelligence [1]. It describes how the model grows from an initial, sketchy, general description to its final version, ready for verification.

NMR is a modelling and refinement strategy that we propose. NMR is used for building a model for formal verification. It can be used for building both verification and design models. In the first case, a modeller models an already existing system and verifies the model. In the latter case, a control is not specified yet, so the modeller designs both the control specification and the system model. Before explaining the method, we list some of the aspects of the modelling process. Our intention is not to extensively discuss these difficult issues, but to point out the modelling aspects addressed by the NMR. These aspects are described with more details in [11] and [14].

2.1 Some Aspects of the Modelling Process

2.1.1 Knowledge Increase and Relevant Information

When designing a model, a modeller learns about the system. The source of information can be technical documentation, or domain experts. Most likely, the modeller does not have a complete knowledge about the system, and does not know in advance all the questions that have to be asked about the system. The modeller starts with an initial system decomposition and identifies component interfaces. Describing the components and their interfaces gives an idea of what information is missing and what questions to ask domain experts.

While modelling, modellers increase their knowledge about the system (top-down process), but they also have to filter out the irrelevant data (bottom-up process). Not all the knowledge about the system and its parts is relevant for the current decomposition level. Also, for different system requirements, different system aspects and properties are relevant. For formal models it is important that they are not too big – a good practice is to describe only what is necessary, guided by the requirement.

For example, suppose we have a system that sorts bricks according to their colour. At the higher decomposition level the model can contain the description of brick scanning, brick transporting, and brick sorting (found in top-down reasoning). Scanning can be described as the process that determines the brick colour after it arrives in front of the scanner. At this level of decomposition, we will not mention reading and writing the register where the value corresponding to a brick colour is written, even though we already know something about it (bottom-up process). This information will be added later, when the scanning process is decomposed and described in more detail.

If the requirement states "The sorter will sort bricks according to their colour", the model will describe the behaviour of the system parts, for example conveyor belt, scanner, and sorter, and the model does not have to describe the timing aspects. The timing aspects are irrelevant for the requirement. But, if the requirement is that "The sorter sorts a brick in less than 10 seconds", then the model will contain information about the belt speed, the duration of scanning, and the sorter speed.

2.1.2 Decomposition, Abstraction, Idealization

Decomposing is a way to deal with complexity. The modeller detects a structure of the system, identifies the components, and then focuses on each component and decomposes further. On each decomposition level, the modeller decides what to put in the model, what to abstract away, and which

idealizations to make. These decisions are related to the requirement that the model is suppose to verify.

In the aforementioned example of the brick sorter, for the requirement that the system must sorts bricks, we will abstract away brick weight, belt colour, the dimensions of the queue where an operator puts bricks etc.

2.1.3 Fixing the Requirement

The requirement cannot be fixed too early. A customer has an idea of what the system should do, but this in practice is never the final version of the requirement. The requirement is likely to change as modelling and control design progress, because only then are the limitations of the plant and the system addressed and related to the requirement. This may require the modeller to weaken or change the requirement.

2.2 The NMR Method

Together with building the model, the modeller should build a correctness argument that shows that the plant and the control together satisfy the requirement. At the same time, the correctness argument is the document of the modelling decisions. It makes us more convinced that what we prove for the model, also holds true for the system. The modeller's task is to find:

- A requirement description: a formula $A \implies C$, where C is a specification of the goal and A is a set of assumptions about the environment.
- A plant and control description: a finite number of formal specifications $ai \implies ci$, one of which is the control specification, while the others are made true by well-understood components of the plant.
- A formalisation N of of the necessary domain properties, such that

$$N \wedge (\forall i.ai \implies ci) \implies (A \implies C) \qquad (1)$$

can be proved.

The formula (1) is stepwise – and not necessarily monotonically – refined. We will explain the steps in detail in the example below. We can say that we develop a first, a second, etc. approximation of the requirement (the right hand side of 1) and the correctness theorem (the whole formula 1).

Realistic model and control design seems to be a chaotic process, but its quality can be improved. Guidance by the canonical form of 1 helps to:

- understand the impact of design decisions on the overall properties.
- document all decisions in a logical place.
- choose the right abstractions for a formal model that can be used for verification.
- define the boundary between the system and its environment.
- find a meaningful decomposition.
- define interfaces when it is understood why they are necessary rather than in the beginning of the process.
- keep levels of abstraction clearly separated.
- take design decisions only when they are fully understood.

We do not propose to always write down the theorem instances while modeling, but this can be a useful exercise when practising and learning modelling. It might also be necessary during development of a safety-critical system. In other examples this could become a cumbersome task. In the latter case, it is useful to have in the back of our minds the theorem that is being built (instead of, for example, a V-model in top-down approaches). *It is knowing and understanding the theorem shape that provides modelling guidelines.*

2.2.1 An Example

Here is how the method works. We assume that the "plant" – in this case a coffee machine (Fig.2) – 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.

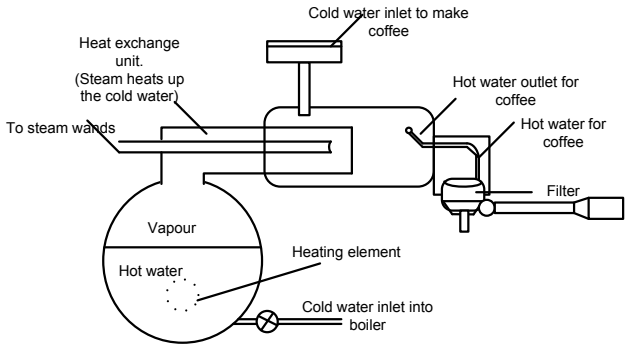

Figure 2: The coffee machine blueprint.

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. For the course of this paper, however, we will use natural language in order not to blur the explanation by language issues.

(1) Start from R_0 - a simple, general version of the requirement to be verified.
We first have to translate a customer's wish into a system requirement. If there is a vague idea of what the system is supposed to do, such as "This machine should make espresso coffee", we will start with a simple general version of the 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. The first requirement approximation is: "When the customer requests coffee, a cup of coffee is poured out in less then 10 s." (R_0).

(2) Use expert knowledge about the system to identify a sub-process that somehow contributes to this requirement.
We are now working on the refinement of the left side of formula 1. Before designing the control, we learn about the

plant. It is a machine that has a lot of parts (we can decompose it to physical parts), executes a number of processes (we can decompose it to processes), and has some functionalities (possible function decomposition).

A coffee machine expert explained to us that the machine makes coffee according to the following recipe: Some water is heated, pushed through the filter with ground espresso beans, and poured out. Just to start somewhere, we decide arbitrarily to focus on the process of heating water and investigate and decompose it further.

(3) For that process, find components in the system that are used to perform them.
For the water heating process we identify the heat exchange unit (*comp*1).

(4) For this component, write a formula $a_1 \implies c_1$ that contributes to R_0 but does not contain more knowledge than necessary to contribute to the overall goal.
Water heats up and is transported to the filter. From the domain expert we found out the following: When the water is inside the unit, steam will be inserted to heat up the water. After that, water is pushed towards the filter that contains the ground coffee. (Although the domain expert mentioned steam and the filter with the coffee, now our focus is not on it, so we will leave describing it for later.) In the $a_1 \implies c_1$ we put only the knowledge that we have about the component *comp*1.

(5) Obtain knowledge about the quantities the component requires and can handle; incorporate these in $a_1 \implies c_1$.
We learned from the domain expert that the amount of the water should be 50-60 ml. Water has to be heated to 92 - 96^0C. Heat is needed until the desired water temperature is reached. (Most probably we have some "bottom-up" knowledge, or idea about a temperature sensor, but at this composition level, we are describing only the behaviour of a unit as a whole.) So now we have a refined $a_1 \implies c_1$: "If the 50-60 ml of water is inside the unit and sufficient heat is provided, after 11 s hot water of 92 - 96^0C is pressed towards the filter that contains the ground coffee."

(6) Find out which parts of the a_1 will be ensured by another component of the plant and which have to be provided by the environment of the system.
The water will be provided from the water reservoir inside the "plant". Heat also comes from another component.

(7) Decide which parts of the c_1 will have to be dealt with by another component.
The component *comp*1 is pumping the water out to the filter that contains the ground coffee. (Steps (2) and (7) are related - if the domain expert provided a good description of sub-processes, every c_i will be matched by some a_j. But, it can happen that the domain experts or the technical documentation does not describe the system in a structured way or giving the information on the right abstraction level and relevant to the requirement.)

(8) Replace R_0 by a (possibly weaker or slightly different) $A_1 \implies C_1$ that reflects the new state of knowledge.
In our example the requirement becomes: "When the customer expresses a wish for coffee, 50-60 ml of coffee is poured out in at least 11 s" (R_1). At this point we know that the coffee preparation will last minumum 11s. So our system, as it is at the moment, cannot satisfy the requirement (R_1). We either have to change some part of the plant or we have to weaken the requirement. Suppose we did the latter.

(9)Decrease "difference" between $a_1 \implies c_1$ and $A_1 \implies C_1$
At this moment we do not have a theorem that we could prove (even if we formally write down the aforementioned descriptions). We have already found out that other components are needed to ensure a_1 and to bridge the gap between c_1 and C_1. So we now focus arbitrarily on one of them and come up with some $a_2 \implies c_2$. We adapt $A_1 \implies C_1$ accordingly.

In this case we will continue by describing the heater, the filter, the cold water reservoir, and the start button in the same way we describe the heat exchange unit, in steps (4) -(6). Describing them will give us idea what components and sub-processes to describe next.

(10) Continue this method, adding new processes and their system components until arriving at a provable:
$(a_1 \implies c_1, a_2 \implies c_2, , ..., a_n \implies c_n) \models A_n \implies C_n$
As explained in the previous step, we are adding the "missing parts" in the formula (1). The steps we describe are not prescriptive, they are arbitrary. They support refining the theorem that contains descriptions of all the plant components and their interfaces and, the requirement description. It is important to stay on the current decomposition level (e.g. we do not decompose further the heat exchange to temperature sensor, pump etc.), while we are describing the missing parts of the theorem (other components and/or subprocesses at the current decomposition level).

(11) Adapt the ai of all processes so that they include the settings of the necessary sensors and actuators (we denote them as a^*_i).
After we have the description on a certain decomposition level, we decompose all the parts, in the same way we did it in steps (3)-(6). This way we arrive to the description at the lower decomposition level. We stop decomposing until we arrive at the description of signals at the actuators and sensors.

(12) Find a specification X such that
$a^*_1 \implies c_1, a^*_2 \implies c_2, ..., a^*_n \implies c_n, X \models A_n \implies C_n$
Finally, we design the control specification. In this example, the control will sense whether the start button is pressed. After the start button is pressed, the control will turn on the valve for a certain time to add 50-60 ml of water in the heat exchange unit and then it will turn off the valve. At the same time, it will turn on the heater and it will turn it off, after the water temperature reaches 92 °. Then, the pump is turned on, the valve at the tube for pouring coffee is opened and then closed after certain time.

The NMR method follows the process of building up knowledge about the plant. It does not give the recipe for how to design the control. So, the control description is part of the correctness theorem, but the way it is structured is left to the designer. The solution might require further weakening of $A_n \implies C_n$, if, for example, the overall process takes more time because sub-processes cannot be executed in parallel.

3. REQUIREMENT PROGRESSION

The requirement progression [12] is the technique for obtaining software specifications from the requirements. It extends the problem frames (PF) technique [7] 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.

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.

Problem diagrams represent the requirements, the domains in the problem world, and the machine domain. A problem diagram shows how the domains are connected to each other and to the machine, and which domain is constrained by the requirements. The correctness argument is expressed non-formally: "The machine and the plant fit together properly". In a concrete problem, this argument is refined into the description of the assumptions regarding the domain behaviour. The developer has to argue non-formally that the specification of the machine and the assumptions imply the requirements.

The requirement progression technique provides a framework for building a formal correctness argument. The idea is to transform the requirement in steps so that, in each step, the requirement moves closer to the machine. The requirement referring to the machine only is the machine specification, so once we have it, we stop with transformations.

The technique includes three steps: writing down the assumption, rephrasing it and moving the requirement closer to the machine.

Starting from the initial requirement, the analyst examines the domain constrained (touched) by the requirement. The analyst writes down the assumption ("breadcrumb") regarding the domain behaviour. The breadcrumb can mention only the phenomena from that domain. A domain is connected through its interfaces to other domains. When the analyst starts writing a breadcrumb, he also has in mind the domain that will be mentioned by the next version of the requirement. So, the breadcrumb should be phrased in such a way as to enable a useful rephrasing of the requirement.

In the next step the requirement is rephrased. The breadcrumb has to be sufficiently strong to permit requirement rephrasing. So the following implication should be satisfied: $(breadcrumb \wedge newrequirement \implies priorrequirement)$. The rephrasing is chosen to enable pushing the requirement to another domain, closer to the machine.

Finally, it is possible to push the requirement so that it constraints another domain closer to the machine.

3.1 Comparison with NMR

3.1.1 Differences

(1) The requirement progression starts with already decomposed plant and fixed interfaces. A requirement is, preferably, already fixed and formalized. Once a problem diagram exists, and a requirement is fixed, the process of modelling and then building up the correctness argument starts.

The NMR method guides a modeller in coming to the decomposition stepwise. As a consequence, the interfaces between domains are also refined in a stepwise manner.

(2) In the RP method, because the initial requirement is fixed, it does not weaken or change. Every time the analyst finds a "breadcrumb," it validates it with the domain expert and continues further. The description progresses in a monotonic way.

The NMR method takes into account that the properties of a realistic system usually are a compromise between what one wanted ideally and what one was able to realise. Often, one did not initially understand what one wanted or what was possible. Specifications and blueprints often did not accompany the development process. Unlike in monotonic refinement, the decisions taken in such a chaotic development process are not limited to design decisions. Many decisions are adaptations of the original requirement.

(3) The machine specification and the description of the plant are separate. The plant is described with the breadcrumbs and arguments: $breadcrumb \wedge new\_requirement \implies old\_requirement$. The machine specification can be verified separately.

The NMR describes the whole system, both plant and the machine in one model. Describing both the plant and the control in one model results in complex models. So, describing them separately helps avoiding the problem of too complex models.

3.1.2 Similarities

(1) Both RP and NMR explicitly address the following modelling steps within the method (although at a different stages): decomposition into parts, identifying interfaces, focusing on one domain at a time, and describing its behaviour in terms of its interfaces.

(2) Both methods include the plant descriptions as part of the model or correctness argument.

(3) The requirement progression (RP) is presented using problem diagrams, but in problem frames based methods, the analyst can choose what formalism to use to specify the requirements and the behaviour of domains.

The NMR method also does not prescribe any modelling language or technique to be used. If we choose to use a theorem prover to verify the system, it would be handy to use propositional logic. If we use a model checker to verify the system model, we could describe the correctness theorem with process algebra or automata.

4. GOAL-ORIENTED APPROACH

The goal-oriented, KAOS (Knowledge Acquisition in autOmated Specification) method [13] is the method for designing software specifications. The software specification is derived gradually. The method takes into account both functional and non-functional requirements, because the latter play an important role in designing specifications, too.

The first step is to elicit the system requirements and phrase them as goals that the system has to achieve. Also, the obstacles that prevent the achievement of the goal are identified. The system parts that will contribute to achieving the goal are described. Those parts either achieve subgoals or prevent obstacles. These parts are both software and hardware (plant) parts.

The terminology used by the method is as follows. The *goals* are defined as "declarative statements of intent to be achieved by the system under consideration" [13], where the *system* consists of software and its environment. The goals range from goals to be achieved by the system as a whole,

to goals achieved by the system components. The system components are called *agents*. They can be software components, devices within the system, or humans playing specific roles (e.g. operator). A *requirement* is a goal under responsibility of a single *software agent*. An *expectation* is a goal achieved under responsibility of a single environment agent. Goal models manage interactions among the system goals.

These models also help in identifying exceptional conditions in the environment that might prevent some of the goals. The idea of the method is to specify the goals precisely and to refine them incrementally into operational software specifications, so that they provably assure high-level goals [2].

The steps proposed by the method are [13]:

(1) Elaborate the goal refinement graph - preliminary goals are collected. From the documentation, the preliminary goal is identified. Next, the obstacle to achieving the goal is identified. The goal to mitigate the obstacle is then introduced. This goal is refined into the property of the system so that, when satisfied, the obstacle is overcome, and the subgoal of preventing this property can be achieved. The descriptions of goals are formalized in temporal logic.

(2) Object modelling – conceptual classes, attributes, and associations are derived (so they decompose the system using an object-oriented paradigm). Classes, attributes, and associations describe the environment. The decomposition guideline is the following: an element is modelled if and only if it is mentioned in the declarative assertions about goals and requirements.

(3) Detect and resolve goal conflicts: If a conflicting goal to the current goal is detected, then one of the two goals has to be changed or weakened.

(4) Identify agents together with their control/monitoring capabilities: Goals have to be refined until they can be assigned as responsibilities of single agents. A goal can be assigned to an agent if the agent has sufficient monitoring and control capabilities to realize this goal. A catalogue of agent-based refinement tactics has been defined to guide the process of goal refinement.

(5) Derive agent interfaces.

(6) Generate and resolve obstacles to goal achievement. Obstacles are exceptional behaviours. Obstacle analysis means identifying as many ways as possible to break desired properties. The obstacle analysis is an iterative process. It can generate new goals for which new obstacles may need to be defined.

(7) Operationalization - deriving operational software specifications from the goals assigned to agents.

In parallel, two other steps are performed:

(8) Handle conflicting goals, and

(9) Handle obstacles that might obstruct the goal.

The KAOS method also provides obstacle analysis and goal refinement strategies. For every stage of the specification, there are guidelines or heuristics or patterns for deriving subgoals and agents. For example, agent-based refinement tactics provide guidelines to refine goals and construct agents that will implement these goals [10].

4.1 Comparison with NMR

4.1.1 Differences

(1) The KAOS method describes both functional and non-functional requirements, while so far, in the NMR method, we have focused on functional requirements that are formally verified.

(2) Within the KAOS method, it is necessary to formalize the descriptions early. We would like to leave the formalization step to a later point, when we know more about the system.

(3) In the KAOS method, the decomposition is guided by goals. The system is described through the goals and subgoals that it has to achieve. With NMR we have the possibility of different decompositions.

(4) The KAOS method proposes the list of steps in an order that corresponds to an idealized process. In reality these steps intertwine and there is backtracking between them. Unlike the monotonic refinement model, NMR describes the model development as a process that never backtracks, but constantly grows. So, when a goal has to be weakened, this requires propagating the decisions to all necessary descriptions related to this goal [9]. In NMR, by keeping all approximations of the correctness theorem, we also keep the argument as to why the requirement had to be weakened at some point, or why a component description or a component itself had to be changed.

4.1.2 Similarities

(1) Both methods start with stating the preliminary requirement, not trying to fix it immediately (whereas the RP method starts when this phase of fixing the requirement has already been done).

5. DISCUSSION AND CONCLUSION

The problem we are addressing is lack of a method to use existing tools and languages for formal verification. Modelling and verification are techniques one must learn, otherwise there is the danger that a modeller might become lost in language issues and details of tool usage.

Another potential problem comes from the fact that formal verification is used to prove some critical sub-requirement of the system. Modelling a system for proving all the requirements would result in too complex a model. Therefore, it is crucial to choose only the portion of the system and its aspects and properties that influence the chosen sub-requirement. A modelling method should provide guidance in choosing which system parts and aspects to model.

The NMR method proposes to guide the modeller with the shape of the theorem we want to prove: that the plant and the controller together satisfy the requirement. We refine each instance of the theorem at a certain decomposition level. However, we do not start with the decomposition immediately; we choose a process that contributes to the requirement and a component that performs this sub-process. Its description reveals to us which component to describe next. In this way we define interfaces when we understand why they are necessary, rather than at the beginning of the process.

On a lower decomposition level, each component is described in more detail. It can happen that, due to our increase in knowledge, the component was non-monotonically refined. A monotonic method would suggest that we go back and change previous steps, in order to fake the top-down method. With NMR, however, we refine the theorem and do not change its previous instances. In this way we document our modelling decisions in a logical place and do not mix the levels of abstraction.

The method proposed here has been challenged in a course in applied formal logic for several years. In the years before our method was taught, the students were mainly trying to model everything they could think of, in order "not to forget anything that might be relevant", until they got completely lost. Since we have been teaching the NMR method, we have been told by many people that the main insight they got during the course was to systematically leave out the irrelevant and formalise only what is necessary to prove the requirement.

The biggest problem of our method is that the actual writing down of each instance of the theorem is a cumbersome task. If we are using a theorem prover to model the system, we still have to do it. However, in some other cases it is useful to simply keep the theorem shape in the back of our minds and to be aware of what we are describing at each moment while modelling. In future work, we plan to look at certain classes of problems and see what is characteristic for them.

The idea to formally describe some modelling process aspects was proposed by Gunter et al. in [3]. 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 et al. developed a formal conceptual network based on problem-oriented perspective [4] where they described formally modelling steps. In our research, we identified these steps too, but our focus is on finding ways to systematically perform these steps.

5.1 Combining NMR with Requirement Progression and KAOS Methods

The requirements progression method starts with fixed and preferably formalized requirements. The decomposition is made at the start of the modelling process, but without guidelines how to decompose. Also, the modeller does not have guidelines what to ask domain experts. We believe that NMR method could be used in the initial steps, to come to the decomposition of the system and as a guidelines what to talk about on each decomposition level.

Within the KAOS method, the system is specified using the concepts of goals and subgoals, and hardware and software agents, so the way to decompose the system is fixed. However, it would be interesting to investigate whether NMR could guide the process of identifying these components. Also, both methods could use the form of the correctness theorem to document modelling decisions.

6. REFERENCES

[1] G. A. Antonelli. Non-monotonic logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2006.

[2] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.

[3] C. Gunter, E. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

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

[5] S. O. Hansson. Logic of belief revision. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2006.

[6] M. Heisel and J. Souquières. A method for requirements elicitation and formal specification. In J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, editors, *Proc. of the 18th International Conference on Conceptual Modeling*, volume 1728 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 1999.

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

[8] 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).

[9] A. v. Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, November 1998.

[10] E. Letier and A. van Lamsweerde. Agent-based tactics for goal-oriented requirements elaboration. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 83–93, New York, NY, USA, 2002. ACM.

[11] A. H. Mader, H. Wupper, and M. Boon. The construction of verification models for embedded systems. Technical Report TR-CTIT-07-02, January 2007.

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

[13] A. van Lamsweerde and E. 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.

[14] H. Wupper. Taxonomy of computer science and non-monotonic refinement - wiki page. https://lab.cs.ru.nl/taxonomie/Non-Monotonic_Refinement.

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