Structuring Problem Analysis for Embedded Systems Modelling

Jelena Marincic
Dept. of Computer Science
University of Twente
Enschede, The Netherlands
j.marincic@ewi.utwente.nl

Angelika Mader
Dept. of Control Engineering
University of Twente
Enschede, The Netherlands
mader@ewi.utwente.nl

Roel Wieringa
Dept. of Computer Science
University of Twente
Enschede, The Netherlands
roelw@ewi.utwente.nl

Yan Lucas
Neopost Technologies
Drachten, The Netherlands
Y.Lucas@neopost.com

ABSTRACT

Our interest is embedded systems validation as part of the model-driven approach. To design a model, the modeller needs to obtain knowledge about the system and decide what is relevant to model and how. A part of the modelling activities is inherently informal - it cannot be formalised in such a way to constitute a basis for automated model design. This does not mean that modelling has to be chaotic. We therefore propose an informal method that structures modelling activities. In this paper we will focus on one of the method ingredients - modelling guidelines. In the industrial case study we performed, we captured modelling steps and elements in a form of a modelling handbook. The goal was to make modelling more efficient by preventing next modellers re-inventing things, but also to preserve a modelling style recognized within company's context. We show in detail what these re-usable modelling elements are, and how identifying them can be generalised for designing modelling guidelines in general. Finally, we compare our work with work of researchers that formalise problem analysis.

Keywords

model-driven validation; modelling steps; embedded systems;

1. INTRODUCTION

Modelling embedded systems as part of model-driven design is intended to increase our confidence that the system will behave as required. During formal verification we design a mathematical model of the system and formalize requirements for the system's behaviour in terms of this model. It is possible that the control software and the plant (the rest of the system) already exist, or we verify the system while, at the same time, designing its control.

There are many languages and tools for formal verification, but one is left without a technique how to use them [7]. In other words, there is no technique that guides us while designing a model. This results in having models of different quality depending on modeller's experience and talent.

Modelling itself cannot be formalised in such a way to constitute a basis for automated model design. While designing the model, the modeller has to obtain knowledge about the system and about its requirements. Furthermore, she has to identify what is relevant to describe with the model and how to describe it. Also, we cannot formally prove that the model is an accurate representation of the system (and vice versa).

Our research is focused on designing an informal modelling method that improves confidence in the model and efficiency of modelling. Even though parts of modelling are informal, they need not be chaotic. By making them explicit and providing guidelines we make them accessible and controllable so that we can evaluate them. We identified three informal modeling aspects important to address and that usually stay implicit while modelling. They are: (1) modelling assumptions that are not part of the model [11], (2) steps that guide modelling and make it more uniform and structured, in other words make it an engineering process; (3) the argument that the model correctly represents the system [13]. We are building techniques to explicitly address these aspects.

In this paper we will focus on the second modelling aspect - steps and guidelines that structure modelling process. Our research question is: What can be generalised and extracted from modelling steps in one particular case and later reused in form of guidelines, checklists or useful techniques?

Through the description of one of our industrial case studies, we will show re-usable modelling steps and elements we captured in a form of a modelling handbook. The company where we performed the case study designs inserters - machines that automatically fold papers and insert them into envelopes. We analysed an inserter emulator - a digital circuit model of inserter's mechanical part (or "plant" as it is called). The specification of the emulator is the plant's

IWAAPO '10 Cape Town, South Africa

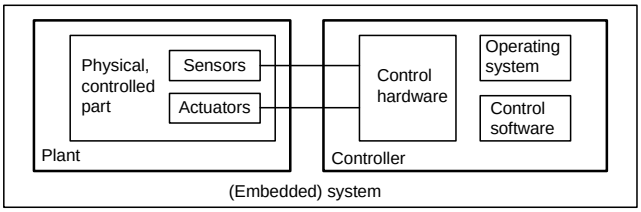

Figure 1: In embedded systems, controller enforces plant desired behaviour, via actuators, based on sensor readings.

model as well. It is designed by the modeller and then automatically transformed to emulator execution. We designed a handbook for specifying emulators of future generations of inserters. The goal of the handbook was to make modelling more efficient, but also to preserve modelling style recognized among the model stakeholders. To be able to answer our research question, we generalised the results from the case study in form of classes of reusable modelling elements and decisions.

The rest of the paper is organized as follows. Section 2.2 explains the terms and concepts we are using as well as problems we tackle and research questions we start from in our research. Section 3 describes the method we propose. In Sections 4 and 5 we describe the case study we performed. Section 6 gives the relation with other researchers work in the area of model design and conclude with Sect. 7.

2. PROBLEM DESCRIPTION

2.1 Basic Terms and Concepts

2.1.1 Embedded systems

As shown on Fig. 1, an embedded system consists of a *controller* and the rest of the system, called the *plant* by embedded systems designers. The controller consists of the control software running on a special purpose hardware or a PC computer. The controller observes its environment via sensors, and enforces the plant's required behaviour through actuators. We will abstract away the controller hardware and focus on the interaction of the control software with the plant.

For embedded system modeling, it is important to determine where the boundary of the system we are analysing is [19]. The environment of the controller, relevant for the model and the requirement, usually goes beyond the plant. There are users and operators that for example press buttons, products manipulated by the plant like paper in the printer, bottles in a bottle filling machine or chemicals mixed or extracted in a chemical plant. In wireless devices signals are transmitted through air, so the signal's frequencies and intensity can be relevant to analyse when verifying these systems. One of the problems our method deals with, is the description of the controller environment in a verification model.

2.1.2 Verification Argument

To verify the system means to give a correctness argument that the plant (P) and the control (C) together satisfy the requirement (R).

$$P \wedge C \implies R \tag{1}$$

Using formal methods to build the correctness argument increases our confidence in the system. If the control is not designed yet, verification models can be used to help the control specification design. In that case the modeller's task is to examine the problem, requirements, and the plant, and then create the solution in form of control specification. If, on the other hand, the control specification already exists, then the modeller has to additionally learn about the control, too.

2.2 Making Modelling Systematic

To design a model, it is not enough to learn a modelling language and get familiar with the tool that supports it. Before creating the solution (the model), the modeller has to analyse the modelling problem. She has to obtain knowledge about the system and the requirement, to decide what is relevant to describe with the model with respect to the requirement, and to decide how to model it.

The control software and the plant work together to satisfy the system requirements, but the requirement refers to the plant [8]. To decide what is relevant to model, the modeller needs to obtain and integrate knowledge from different domains, like for example electrical, mechanical etc.

Both analysis of the modelling problem and model construction are informal activities. This is in contrast to formal techniques that manipulate the model. Using them, we get a mathematical proof that the model has (or not) a certain property. If it does, we conclude that the system has it, too. But we do not have proof for this conclusion, we can only be confident in this. Also, we cannot automate the modelling process. This makes model quality and efficiency of modelling dependent on the modeller's talent and experience. We want to improve this situation by making modelling a stuctured, systematic activity.

Our goal is to make the modelling process less dependent on the modeller, her experience, talent and creativity. We also want to increase confidence in the model by having the modelling process more structured. We are looking for steps and principles that can be captured and reused to guide modelling.

To build our method, we need to answer the following research question: What are the elements of the modelling process that can be generalized and (1) used to explain the existing model and argue its correctness, (2) transformed into guidelines that teach others how to model and (3) exploited when maintaining or redesigning the model? In our research, we focus on the problem of the plant description (modelling).

3. TAXONOMY OF MODELLING DECISIONS

In her earlier work, one of the authors proposed a taxonomy of modelling decisions [10]. Its purpose is to provide understanding of modelling as informal activity that leads to a formal verification model. Having better understanding of modelling, being aware of its structure, already structures the way we think and work while modelling. But, the elements of the taxonomy can be also articulated as questions to answer while modelling, or as a checklist of activities to perform. At the same time, it is the list of questions to explicitly address and further refine. The elements of the tax-

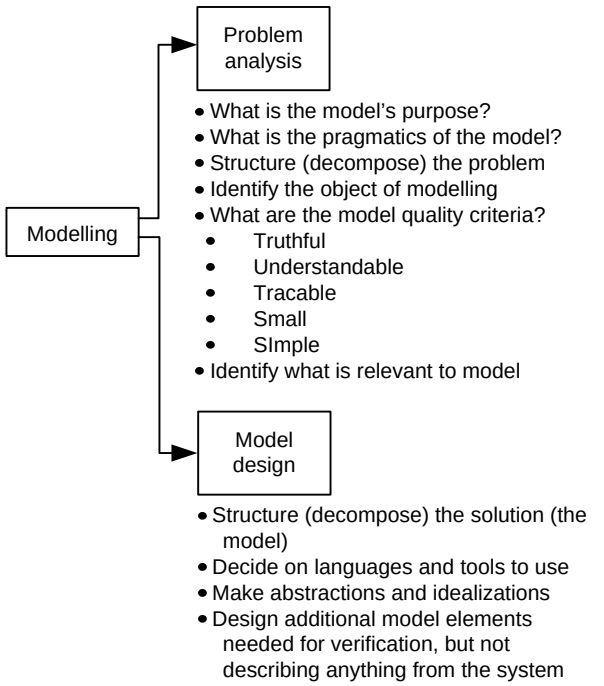

Figure 2: Modelling seen as problem analysis and solution design.

onomy are shown on Fig. 2. The high level categorization is to modelling problem analysis and model design. Each of these is further categorized into types of modelling decisions performed.

The purpose of the model is in our case to verify the requirement, but verifying the requirement is, as we will show later on the example of our case study, part of broader verification process. Also, at a different level, there are a number of requirements (quality criteria) for the model. For example, we might require that the model is traceable in a sense that it follows the existing decomposition of the system as much as possible. Additional limitations and requirements on how the system will be modelled are imposed by pragmatical issues like time, money and resources present. This is not addressed in the academical environment, but in practice it is not possible to neglect it.

Before identifying relevant parts, it is necessary to define what the object of modelling is. Is it a specification of the control, or the system requirement? Is it the algorithm that we have to verify or the control that implements it? There are differences between modelling these, and often they are not clearly stated.

While analysing the problem, the modeller also decomposes and structures it, to better understand it. Further decomposition is needed to find out what aspects of the system are relevant for the model and what parts and their properties should be represented in the model.

The modeller decides on tools and languages to use and mathematical domain in which the system is described. Also, the modeller structures (decomposes) the model which can coincide with the system decomposition he identified. Representing the system with the model means that different abstractions, simplifications and idealizations are done. They, of course, should be done in a way that the model still represents the system with respect to the requirement. For example, if we do not need to prove a timing requirement, we can model plant's actions as the actions that do not take any time.

Figure 3: The inserter folds paper sheets and inserts them in envelopes.

The model does not contain only the elements that represent the system. Often, to prove the requirement, additional elements are needed and they depend on the modelling solution, and languages and tools used. For example, in a model of a printer we possibly need an extra variable to count number of papers printed, although there is no any counter in the printer.

As we will show on the example of the work on our case study, the taxonomy can be used while modelling, but it is even more useful to apply it to the context and to refine the modelling steps given.

Applying the steps from taxonomy is part of the method we propose. As we will show on the example of our case study, these elements are helpful, but it is even better if we further refine these steps. Further refinement produces steps that are depending on the actual problem.

4. DESIGNING A MODELLING HANDBOOK - ACTION CASE

4.1 The System Description

We performed field research (action case) in a company [14][15] that develops, produces and distributes different types of mailroom equipment and document systems. The users of document systems are companies that send a lot of paper mail on a daily basis, like insurance companies, post offices, banks etc. One such machine, called *inserter*, is shown on Fig. 3. The inserter automatically folds paper sheets and inserts them in envelopes.

The inserter works as follows. The operator places documents (paper sheets) and envelopes in appropriate feeders. He also specifies recipes for manipulating documents, through the user interface. A recipe defines how many documents will be inserted in each envelope, whether the paper sheets will be folded and how (for example, a document can be folded along the half or along the third of its length). An example of a recipe is: "Select three documents from the first feeder, make a "Z" fold and insert them into the envelope." Recipes form scenarios, e.g. one scenario would be to repeat the previous recipe hundred times.

The diagram in Fig. 4 describes inserter modules where

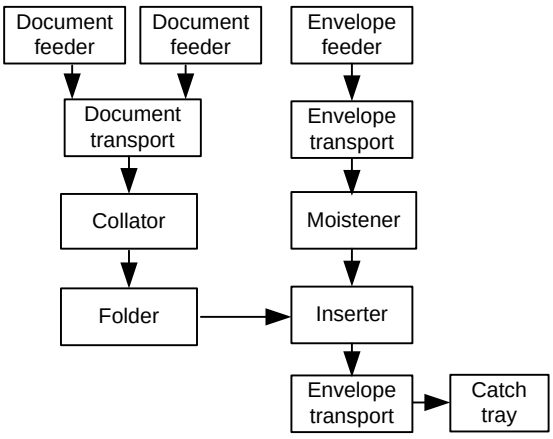

Figure 4: Diagram of the mechanical plant parts and path of documents and envelopes

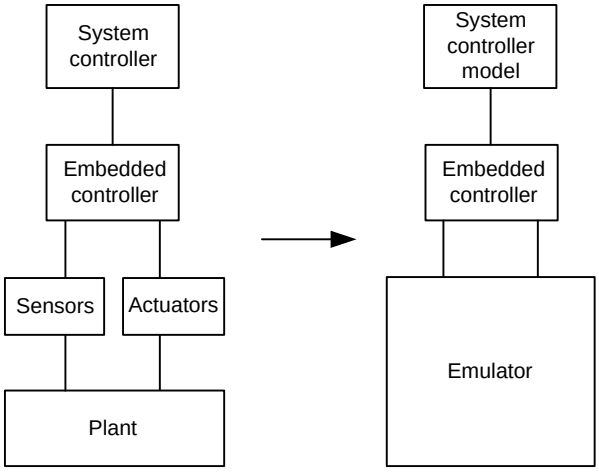

Figure 5: Inserter's high level architecture and the concept of the emulator.

different processes take place. The arrows in the diagram represent the flow of the documents and envelopes through the system. In the feeder, rollers (small plastic wheels) rotate along the surface of the document on the top, which pulls it out of the feeder. The document is then moved toward the collator, again with rotating wheels. The wheels are placed in pairs on each side of the document path. In the collator, the documents that form a batch to be inserted in one envelope are collected and straighten out (collated). The batch is then moved to the folding position. It is folded with a stroke of a long, thin, sharp-edged arm. In the meantime, the top positioned envelope in the envelope feeder is pulled out of the feeder, and brought to the flap moistener. Its flap is first turned up and then moistened with a stroke of a brush. The upper side of the envelope is lifted, so that document batch can be inserted. Folded documents are then inserted in the envelope, the flap is closed and the envelope with documents is moved toward the exit of the machine.

Every process performed on documents and envelopes is the result of controlled movements of mechatronic parts. Rollers, folding arm, brush and many other system parts are connected to actuators that move them. Along the system, sensors are placed to signal the presence or absence of material in front of them. Based on sensor readings, the control software sends signals to actuators.

The high level system architecture is shown on the left side of Fig. 5. The System Controller handles operator requests and recipes and forwards them to the Embedded Controller. Embedded Controller controls the plant.

4.1.1 Plant and Control Software Integration

When a new generation of inserters is designed, some mechanical parts, or even whole modules, are re-used from the old inserter generation. Some parts, on the other hand, are completely re-designed. Radical design results in significant improvements of functionality or performance which is important for competitiveness on the market. At the same time, it brings unanticipated problems and issues that did not exist before [18]. Some of these problems are related to the integration of the control software and the plant. If the integration comes in a later stage of the project, solving these problems might need rework of parts, whole modules or concepts which causes delays.

The plant and the controller together deliver the inserter's desired behaviour. It is therefore important to enable their concurrent development. When the control software and plant are designed at the same time, mechanical and software engineers communicate from early development stages, and problems related to plant and control software integration arise early enough to be resolved on time.

The problem is that in the early development stages the plant exists only in sketches and CAD drawings. The interaction with the plant via sensors and actuators is in the essence of the control software, therefore without the plant control software testing is not possible.

4.2 The Plant Model

To enable concurrent engineering of the plant and the control software from early development phases, the company designed and built the plant emulator for the purpose of control software testing. Figure 5 shows the comparison of the inserter architecture and the architecture of the testing setup. The borders of the control hardware are the interfaces that exchange signals with sensors and actuators. The emulator is connected to these interfaces. On the emulator side, the interface connected to the controller mimics sensors and actuators behaviour.

The emulator is a programmable hardware device, which means that it consists of digital circuits that a user can configure and combine. These digital elements are connected and they together mimic the intended future plant behaviour - the flow of documents and envelopes, rolling of rollers etc. We can say that the emulator ($M1$) is a digital signal model of the plant (P).

There are two more models that play a role here - the specification of the emulator (Φ) and the run-time visualisation of emulator's signals ($M2$).

The specification of the emulator's digital signals behaviour is at the same time the specification of the plant behaviour. The specification is written in a graphical language, called G-language, using the LabView tool. LabView diagrams describe the flow of material, the sensor reaction to the material and movement of different physical parts. The LabView compiles the diagram-based specification to the program for

the hardware. We consider this transformation correct, and deal with the graphical model of the system.

For visualisation purposes, the diagram that shows the states of emulator's digital elements is designed (part of the LabView package are also run-time diagrams for hardware monitoring). To be more precise, only those digital elements representing sensors are collected. Their order in the diagram corresponds to the spatial order of the sensors in the inserter. This is convenient for the testing engineer who monitors signal flow in the emulator as if it were document flow in the plant.

We can therefore argue that we have three models. One is the physical, digital signal model of the plant. The other is the graphical model, the diagram that is designed to represent the plant. The third is the run-time visualisation model. We can say that: $M1 \models \Phi$, $P \models \Phi$ and $M2 \models \Phi$.

The model we analysed is the second one - the graphical specification of the emulator (Φ). The reason for this is that we are interested in designing the model that represents the plant for the purpose of control software testing. Our questions are related to gathering knowledge about the plant, and deciding how to describe it. In this case, the model designed to represent the plant is the emulator's specification diagram. The programmable hardware (emulator) is just the execution of the specification, in a way similar to the computer hardware's execution of a computer program. The model could have been just as well implemented as a simulator, describing the plant with software; for our research question this would not make any difference.

4.3 Design Task

As we already explained, the plant model already exists. Designing it for the first time took significant time and effort. The modeller spent a lot of time learning about the plant, trying out different modelling solutions and finding the optimal one. He did not document his insights and knowledge progression, due to time-pressure. The model-based testing improved communication between departments and shortened the integration time, so it will be used in the future.

How the new, future generation of the system will look like, when it will be developed, which parts will be incrementally and which radically designed, is not known yet. The modeller will most probably be another software architect as the original modeller has already left the company. In fact, the modeller may be someone from outside the company. Finally, the tools and languages used now, may not be the choice next time.

To make modelling efficient next time the model is designed, and to preserve existing modelling style, we designed a modelling handbook.

5. THE HANDBOOK DESIGN

5.1 Solution Approach

We designed the modelling handbook, or in other words, we designed a modelling method suitable for the concrete modelling problem. The left hand column in the table shown in Fig. 6 shows the modelling steps we propose in the handbook. The right column in the same table shows our solution approach for this case or how we designed this method.

For modelling handbook design, we had to find what elements of the model and modelling process can be extracted and turned into clear, relatively short modelling guidelines.

Modelling method suited for the problem	**Steps to design this method**
Perform refined taxonomy steps to understand the problem	Apply the taxonomy on the existing models
Apply re-usable modelling solutions	Look for categories of modelling solutions in the existing models
Re-use previous model parts	Identify what parts of the system will not change

Figure 6: Modelling method and steps to design it

- We applied the taxonomy of modelling decisions [10] developed in our earlier work.
- We interviewed the modellers about their modelling decisions. More precisely, we asked for the rationale of their modelling choices and decisions.
- We interviewed model stakeholders about requirements for the model and about the system.
- We analysed the model and asked model stakeholders to estimate which parts will not change in the future.

Due to lack of space, we will show only a couple of elements of the handbook elements as illustration of our points. The complete handbook (without the confidential parts) can be found in the appendix of our case study report [12].

5.2 Applying the Taxonomy

We performed the analysis of the existing model using our taxonomy. When applied to concrete case, we were able to

- refine modelling decisions listed in the taxonomy
- examine if and how these steps showed while designing the model and
- examine if and how these steps could be seen in the model.

Some of our findings are the things that are not changing, they are the part of the whole modelling context in the company, and by context we mean the way the things are viewed, designed, documented and organised. Most likely they will stay the same next time the model is designed. So, documenting them may be useful to start from next time the modelling task for emulator is defined.

5.2.1 Identify the Purpose of the Model

The high-level purpose of the emulator is to facilitate testing of the control. The initial idea was to use it until the plant is made, but the emulator turned out to be useful for continuing some of the tests even when the plant is already there.

We classified verification (testing) purposes as shown in table on Fig. 7. We also listed the faults and irregularities that should be taken into account while modelling. They are: broken sensor, paper sheet slipping (not moving) under the roller, and lengths of paper sheets different then specified. The broken sensor can be always on or always off. Finally, the last thing to address is: what kind of system limitation do we want to test with the model.

Control testing	Plant testing
Testing all possible recipes	Testing limitations of the specification
Regression testing	Testing impact of changes in the software behaviour
Endurance testing	Testing how small changes reflect in system behaviour (e.g. moving sensor)
Fault and irregularities tolerance	

Figure 7: Purpose of the model

Different purposes mean that model describe different plants - one with a certain combination of faults, or one without it. Or a plant with a sensors placed on different distances between each other. Instead of having different, but very similar models for each of these purposes, the model was parametrised. Parameters reflect different faults, different position of sensors and different irregularities in documents length.

5.2.2 Identify the System Requirements to Verify/Test

The system requirement we are interested in is the correct system behaviour - inserting of documents into envelopes, according to the recipes that operator defines. What was verified was that the emulator and control software were correct wrt to the requirement.

There are different modes of work (one recipe and optimisation of many recipes) and for these two, the control behaves differently. For this, the simulator of the system controller is designed to provide the control with different recipes and scenarios.

5.2.3 Identify Quality Criteria for the Model

We explored if the quality criteria we identified were relevant when designing the model and if there were other criteria we did not mention before.

Our basic criterion is that the model should be *truthful*. It means that it has to represent the system correctly with respect to the requirement verified or tested. In the emulator design, the modeller was aware that the model was not truthful, and that there are some things that will show up once the plant is produced. The emulator does not eliminate the need to test and refine the system in the integration phase, it just makes it shorter. It would be good to know if the model is 'truthful enough' to prevent discovering major design errors in the later development phase, but this cannot be guaranteed. Another source of deviating from truthfulness is more benign. Some parts of the model are oversimplified because of lack of hardware resources. This causes some of the tests giving wrong results about the system. However, these modelling decisions were chosen in such a way that further analysis of testing results leads to valid results.

If we want the model to be *understandable* we also have to specify for whom it has to be understandable. For example, the model might be completely clear to another modeller and unclear to a software engineer. The main users of the emulator are testing engineers. They used not the emulator directly, but the LabView model. As we explained earlier, the LabView model is at the same time specification of the emulator hardware and the representation of the emulator during run-time. The model consists of a number of diagrams, each representing one part or one process of the system (emulator). The modeller designed a visualisation model that shows only the main signals in the emulator. This model is at the same time a simulation model. Testing engineers use this model, and in case of problems they examine further a diagram where the source of the problem is. The modeller was always present and available to assist analysis and changes of the model. This may not be the case in the future. The role of the modeller and testing engineer might be separated. So, the more of the modelling style is used in the next model, the less time testing engineers have to spend analysing what is what in the model.

A model is *traceable* if the structure of the system can be found in the structure of the model. It is traceable in the other direction if the fault in the model can be traced to the specific part of the system. The emulator specification followed the decomposition of the system into modules. This way, it was also possible to design a visualisation model, to be able to understand the testing results. Everything we said about testing, when talking about understandability holds here, too. The traceability was in conflict with the model simplicity in the part of the model describing document merging together after they left different feeders. When looking at the model, we may think of different layout of feeders. At the same time, the document flow still describes the flow in the existing plant.

When specifying that the model has to be *small* in case of model-checking, this usually means, simple enough to be manipulated with computer-aided tools and avoiding state explosion. In the emulator case, it was not processing time that had to be considered, but the number of integrated circuits that constitute the emulator. They are placed on integrated circuits boards, and they are expensive. Unnecessary complexity increases modelling time, maintenance time and also makes the model less understandable. For example, the calculation that involve multiplications need a lot of digital elements. That was one of the reason why the motor in the system was not modelled, but kept as part of the system. It was cheaper to have it there then to model it.

Having the model *simple* is closely related to its understandability. Simplicity was not quantified here in form of number of maximal elements in a diagram or maximal number of states. It was the subjective decision of the modeller to dismiss modelling solutions that might have a smaller model, but it would make the diagram difficult to read.

Finally, the new criteria we found while analysing the model was *the degree of software independence*. Even though the plant model should be as much as possible independent from the software specification, this in practice would mean that the model would be too complex, too big and incomprehensible. Certain mechanical parts were designed knowing that, for example, that the control will not make the full rotation of the folding arms, which would cause it crashing into another part. The software dependence might be seen as another source of model's untruthfulness. But, where the modules are reused, and where also the control is reused, it makes sense to have this dependence. For example, if the same feeders are used and the same control that takes two documents from the feeders is used, and if the control and feeders behave in such a way that documents either overlap at least on one third of their length, or do not overlap at all - if this was already tested in previous systems, and if it

would be too complicated to model all possible overlaps, it is reasonable to make the assumption that the control will behave in the way we explained.

5.2.4 Find Out Pragmatic Aspects of the Model

The most obvious pragmatic aspect is that the model has to be as cheap as possible, and be designed in the shortest possible time. This was reflected in the previous criteria we talked about, like not modelling the motor but having it as part of the testing setup, oversimplifying some model elements etc.

Other pragmatic aspects were the knowledge and experience of software engineers. This influenced the choice of the modelling technique and tool. The software engineers were not experienced in using hardware programming language like VHDL, but had experience in LabView programming. This was one of the reasons to choose the second tool.

Another reason to choose LabView was that it was a diagram based. The goal was to avoid a language that looked like programming, because software engineers would start to think of events in the model as the events in the control execution, rather then events in the plant.

5.2.5 Decide What the Object of Modelling Is

The object of modelling are physical parts of the plant, processes performed on documents and envelopes, and the flow of documents. At the first glance, it may look like the emulator is the model of sensors and actuators behaviour only. However, the emulator is more than that. Its signals represent the flow of documents and envelopes, movement of different physical parts and faults in them.

Where do we stop decomposing elements, what is the granularity of the model? In the emulator model, the actuators and physical parts performing functions on the paper sheet were described. Where no faults of physical parts were reflected, the model stopped decomposing on a previously defined functional decomposition.

Another object of modelling that we do not analyse here are user recipes and scenarios.

5.2.6 Decompose the System to Model it

One of the model purposes, besides representing the plant for control testing, is the communication between domain experts. The purpose of the testing is to integrate control and mechanical plant, and modelling is the way to give feedback to plant and control designers.

While modelling, it is sometimes necessary for the modeller to check with different domain experts if the model is correct. If the model follows commonly accepted decomposition of the system, this makes the task easier.

One of the wide-spread decompositions is the one to physical modules and the processes they perform on documents. This decomposition is reflected in how the design, development and testing tasks and responsibilities are assigned, how the teams are organized and how communicate with each other. This decomposition is also reflected in project documentation available to everyone.

The model follows this decomposition whenever possible. The fact that the system is highly modularised helps here. The model is decomposed to diagrams that describe these modules. There are parts that perform different functions and belong to two modules, in this case it was up to the modeller to place them in one of the model components.

Path element	**Description**
Papers	Length, thickness and deviation of the defined length are relevant
Segment	The path of the paper sheets in the inserter is divided in segments. The segment begins at the sensor position. The segment ends before the next sensor. There are one or more rollers on each segment.
Merging point	This is the place where documents coming from different feeders cross each other and merge.
Selector	From this point, two different paths are possible.
Sensor	A sensor is on when there is a material in front of it. It is off when there is nothing in front of it.

Figure 8: Paper sheet path elements described in the model

5.2.7 Decide on Mathematical Domain, Language and Tools

We addressed this issue when talking about the model's pragmatical aspects. We only did not mention that the plant behaviour is described with hierarchical state machines. This is a common way of describing reactive, embedded systems and is natural way of looking at systems and describing them.

5.3 Re-usable Model Components

As we shown in Sect. 5.2, we identified context dependent elements relevant when analysing the problem and that are less likely to change in the future.

We analysed the model to find re-usable modelling components. We did not document them in form of a pattern in LabView, because it is not sure that LabView will be used next time. Instead, we used state charts that both mechanical and software engineers understand.

The model has two very different components, one describes the flow of documents and envelopes through the inserter. It may happen that something change in the future, maybe different sensors will be used, or rollers that have to be described in more details. But certain aspects of the material movement will not change. Due to space limitation we only list some of these elements in the table on Fig. 9; all of them can found in the Appendix of our technical report [12]. We documented them in a form of short explanation, diagram that helps understanding their position in the real system (for this we used the diagrams that the modeller used) and as state machines. The state machines are accompanied with the vocabulary explaining what events and states mean in the system. As these things were written informally, they are not the absolute source of information. For sure, the modeller will have to talk about them with domain experts in the future. They are there to structure issues and elements that the modeller will have to address and as an instruction how to model them, rather then a ready modelling solution, or a pattern to initialise.

Besides these elements we documented possible faults. For example, it is assumed that the broken sensor will either be always on or off (it will not for example be off when it should be on and vice versa.)

The other model components describe the rest of the plant - mechanical modules performing different processes on pa-

Questions to answer when modelling the rest of the plant
For a given function, is there more than one physical element that performs it?
Does examined physical element perform more than one function?
Does the physical component work in synchronisation with another physical component?
Is the duration of performing the function relevant?
What is the interface of a component with other components?
Is the material brought to a component by another part of the system or the component takes it?
Is the initial position of a component relevant?
Is the ending position of a component relevant?

Figure 9: Purpose of the model

per sheets and envelopes, which are: folding and inserting documents, opening and sealing envelopes and wetting envelope flaps. Here, radical changes are possible, so we cannot assume for any of the design solution that it will stay the same. This is the part of the system where new design ideas are allowed and welcomed, because they make the system performing much better and give the inserter advantage on the market.

However, the processes performed on paper sheets will not change. They will stay the same, as well as their order. If we look at the rest of the model, not as the description of physical parts, but the processes they perform, we can extract model parts that do not depend on the physical characteristics and the position of mechanical parts.

One such thing is the algorithm of folding, which is already documented It specifies the order of actions to perform on the paper sheet and the positions of folding movement across the paper sheet. To decide further if there are relevant physical characteristics to put in the model or to take into account, the modeller will have to know more about the concrete solution for folding module and the parts surrounding it.

Analysing the existing model, we generalised from modelling decisions and document questions that could steer modelling in the future. These questions are the result of the analysis what aspects of physical parts and their position were relevant in the current model.

5.4 Rationale of Modelling Decisions

We interviewed the modeller about the rationale of his modelling decisions. Some of them we already described when talking about the modelling decisions in the light of our taxonomy. It is more focused on problem analysis than on designing the solution, although sometimes it is difficult to draw the line between these two. For example, decomposition of the system can be seen as investigation of possible system decompositions or deciding about the model structure.

The taxonomy-based refinement of modelling elements, discovered us the things that do not change. Some of the modelling decisions that belong to the classes we list below will probably change. But, those classes give some directions how to model.

5.4.1 Following the System Decomposition

As we explained earlier, the model followed the existing system decomposition, wherever that was possible. We explained this in more detail in Sect. 5.2.6.

5.4.2 Exemption from Following the System Structure

Some parts of the model did not follow the system structure. More precisely, the part that describes movement of the material from the feeder is implemented differently. The feeders will not change in the future, so we documented this solution as relevant for the future, too.

5.4.3 Model Elements That Do not Exist in the System

A models does not consist only of the elements that describe parts of the system. There are additional elements that are, for example, needed for initialisation of the model or they are tool-related modelling tricks needed to draw conclusions about the model. Some of these elements may look like the description of the system part. In the model we analysed, we observed that there were 'sensors' in the model where they do not exist in the plant. The control software extrapolates position of the paper without a sensor in this particular case, but here the modeller needed the plant to be independent of the control software. The non-existing sensor shows the testing engineer the position of the paper, even though the control only calculates it.

5.4.4 Alternative Modelling Decisions

Some of the alternative modelling decisions were useful to document, because thinking of them does not reveal modelling difficulties. Only when trying these ideas out, it turns out that that model component becomes too complex , or not easy to maintain or not easy to follow the signals in the run-time. To prevent re-invention of modelling solutions, we sketched these alternative solutions and explained the difficulties with them. These were all decisions about the material path, namely they were about the length of the model segment and of the positions of sensors and rollers on them.

5.4.5 Knowledge and Insights of the Modeller

Finding out what plant properties are relevant for the model is a process of discovery taking place while the model is designed. We documented some of the modeller's insights. We also documented the properties that were not relevant for the current model, but were recognized by the modeller as subtle details that might be important in the future. For example, the rollers that move the paper flatten on the places that touch a paper, and this influences the peripheral speed of the roller, and therefore the paper.

5.4.6 Assumption on the control

Ideally, the emulator would describe all the possible plant behaviours. However, this would lead to a model that is too complex to understand, maintain and that will require a lot of hardware sources and modelling time. So, the task of the modeller was to find out those control solutions that will never change. For example, the folding knife will never rotate if the paper is not ready for folding. This is quite a strong assumption on the control, but it was good enough for the modelling purposes. We collected these assumptions,

even though some of them might be different in the future.

5.5 Solution Validation

5.5.1 Internal Validation

We designed the handbook in close collaboration with the modeller. We adopted industry as laboratory' approach [16] which means that we do not invent a method and then look for the problems solved by our method. Instead, we perform action research and adopt our method to the problems we face. Therefore, we were performing shorter, iterative cycles of problem investigation, solution design, design validation, design implementation and implementation evaluation [20].

After the modelling handbook was designed we presented it to model stakeholders - heads of software and system departments, testing engineers and software architects. We interviewed them and asked them to validate the handbook. Software architects found it useful, the head of software department said they would use it for the future emulator. The testing engineer who was using the existing emulator said that for understanding the current model using the actual emulator was much more useful than reading the handbook. This is related to another approach of guiding the modelling, which would be using the existing model and maintaining it. This was possible at the moment, but in the future it might not be.

5.5.2 External Validation

How is our handbook useful for modelling in general? It showed that the taxonomy we developed earlier was a good starting point to analyse an existing model. The classification of concrete modelling decisions is classification of solutions that possibly can be applied in practice.

With this example we argue that it is possible to extract guidelines from the existing plant model of mechatronic systems. To give an answer to the question of how much the guidelines would be useful when starting modelling from scratch - this requires further research.

Another threat to validity is that the emulator is not a formal verification model in a sense we define a formal verification model. But, to build it, it was necessary to be precise and accurate the same way we need to be when designing, for example, an automata based model for model checking.

6. RELATED WORK

6.1 Techniques To Reuse Design Decisions

Our work has common elements with the idea of software product lines and software design patterns.

Software product lines are a technique to reuse one model for a product that evolves or changes. Except that this is the technique to design the software and not the model of the plant or control, it also pays a great deal to maintenance and evolvability. For us this was not an issue, although it can be seen as a quality criteria for the model.

Software design patterns [3] [17] capture the solutions to common software design problems. A pattern is not a solution, it is a template, a rough-structure that has to be shaped to the concrete design problem. Software design patterns are invented by object-oriented community. A pattern consists of the problem description, context of a problem, specification of the behaviour (software requirement), solution description, resulting context and behaviour and what it does not do, and the rationale of the design. The reusable model elements we identified have a lot in common with modelling patterns, but we did not design a LabView pattern that has to be initialised with parameters that describe a future problem.

6.2 Problem Orientation

As we already mentioned earlier, our work has a lot in common with problem frames technique [8]. Starting from its views and principles we continued further in designing an informal modelling method. Other authors combined problem frames and UML to design control [2], combined patterns and problem frames technique for architectural decisions [1], or developed guidelines for designing a formal control model [6].

Another group of researches, including Jackson himself worked on formalisation of the technique [4] or on formalisation of recomposition of the sub-problems found after problem decimposition [9]. A community of researchers are working on formalising problem analysis and combining problem frames technique with formal techniques [5]. Formalising modelling steps give valuable insights on modelling and relationships between different modelling elements. Our interest is more on identifying these steps before formalising them. These steps are an engineer's own way of finding a solution, but they usually stay implicit. They are usually not recognised by academia, because they are not (or cannot be) formalised. Our goal is to find these engineering steps and bring them up in order to cope with them in adequate way.

7. CONCLUSION

To summarise, we are developing a modelling method in order to improve efficiency of modelling and the quality of models. Our method is informal, it structures modelling activity and makes implicit modelling decisions explicit. In fact, in this paper we described modelling method on two levels. This is illustrated in table on Fig. 6. One is the modelling method for designing a model, suited for the problem in hand (shown in the left column of the table). In the case study we performed, we designed guidelines to model future inserters, by reflecting from the existing model.

On another level, we identified steps to design the first modelling method. They are

- Apply the taxonomy to analyse the existing model
- Identify re-usable model components
- For the system parts that will change, write down questions to which current modelling decisions were the answer
- Find out what model components do not follow the system structure
- Identify model elements that do not exist in the system
- Document alternative modelling decisions that might be relevant in the future
- Write down insights that were not relevent for to model, but are hidden in the domain expertise, and might become relevant

- Write down the assumptions on the control

It is not possible to separate these two, as they together form our approach for dealing with modelling decisions.

In comparison to the techniques to formalise problem analysis, we might be not precise as they are, or we might be missing a relationship between elements of our taxonomy, but this does not make our informal method wrong. For future work, we plan to compare the results of formalisation of problem analysis and if the two can complement each other.

7.1 Acknowledgements

We would like to thank Dusko Jovanovic from Neopost for useful comments on this paper and Peter Tjerdsma for insights about modelling the inserter plant.

8. REFERENCES

[1] C. Choppy, D. Hatebur, and M. Heisel. Architectural patterns for problem frames. *IEE Proceedings - Software, Special Issue on Relating Software Requirements and Architectures*, 152(4):198–208, 2005.

[2] C. Choppy and G. Reggio. A uml-based approach for problem frame oriented software development. *Inf. Softw. Technol.*, 47(14):929–954, 2005.

[3] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP '93: Proceedings of the 7th European Conference on Object-Oriented Programming*, pages 406–431, London, UK, 1993. Springer-Verlag.

[4] 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.

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

[6] M. Heisel. Agendas – a concept to guide software development activites. In R. N. Horspool, editor, *Proc. Systems Implementation 2000*, pages 19–32. Chapman & Hall London, 1998.

[7] 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.

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

[9] Z. Li, J. G. Hall, and L. Rapanotti. Reasoning about decomposing and recomposing problem frames: a case study. In *Proceedings of 1st International Workshop on Applications and Advances of Problem Frames*, pages 49–53. The Institution of Electrical Engineers, 2004.

[10] A. H. Mader, H. Wupper, M. Boon, and J. Marincic. A taxonomy of modelling decisions for embedded systems verification. Technical Report TR-CTIT-08-37, Enschede, May 2008.

[11] J. Marincic, A. H. Mader, and R. J. Wieringa. Classifying assumptions made during requirements verification of embedded systems. In *Requirements Engineering: Foundation for Software Quality, 14th International Working Conference, REFSQ 2008, Montpellier, France*, volume 5025, pages 141–146, London, 2008. Springer Verlag.

[12] J. Marincic, A. H. Mader, and R. J. Wieringa. A handbook supporting model-driven software development - a case study. Technical Report TR-CTIT-09-11, Enschede, January 2009.

[13] J. Marincic, A. H. Mader, H. Wupper, and R. J. Wieringa. Non-monotonic modelling from initial requirements: a proposal and comparison with monotonic modelling methods. In *IWAAPF '08: Proceedings of the 3rd international workshop on Applications and advances of problem frames, Leipzig, Germany*, pages 67–73, New York, NY, USA, 2008. ACM.

[14] Neopost. http://www.neopost.com.

[15] NeopostTechnologies. http://www.neopost-technologies.nl.

[16] C. Potts. Software-engineering research revisited. *IEEE Softw.*, 10(5):19–28, 1993.

[17] W. F. Tichy. A catalogue of general-purpose software design patterns. In *TOOLS (23)*, pages 330–339, 1997.

[18] W. G. Vincenti. *What engineers know and how they know it: Analytical studies from aeronautical history*. Johns Hopkins University Press, Baltimore, 1990.

[19] R. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate and the UML*. Morgan Kaufmann, 2003.

[20] R. J. Wieringa. Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, Philadelphia*, pages 1–12, New York, 2009. ACM.