Validation of Embedded System Verification Models

Jelena Marincic*, Angelika Mader†, and Roel Wieringa*
*Faculty of Computer Science
†Faculty of Control Engineering
University of Twente, The Netherlands
Email: {j.marincic, mader, roelw}@ewi.utwente.nl

***Abstract*—The result of a model-based requirements verification shows that the model of a system satisfies (or not) formalised system requirements. The verification result is correct only if the model represents the system adequately. No matter what modelling technique we use, what precedes the model construction are non-formal activities. During these activities the modeller has to learn how the system works, what the requirements are, and to decide what is relevant to model and how to do it. Due to a partly non-formal nature of modelling steps, we do not have a formal proof that the model represents the system adequately. The most we can do is to increase the confidence in the model. In this paper we explore non-formal model validation steps while designing a formal model. On the example of a Uppaal performance model we designed in a company that produces printers, we will show what validation steps were necessary to increase the stakeholders' confidence in the model. Based on this case study, we propose more general, but non-formal model validation steps, that can structure model validation. The steps we propose deal with the same design elements and issues present in other model-based verification activities, therefore can accompany them as well.**

***Keywords*-conceptual analysis of modelling; case study; formal models; non-formal, design steps;**

I. Introduction

There are many techniques, languages, and methods for modelling software-intensive systems. Our interest is modelling as a design activity. From this perspective, we see modelling as a cycle of mutually intertwined (1) problem analysis, (2) model design and (3) validation. Within the problem analysis a modeller learns about the system and its requirements, through a discovery process; the solution (model) design subsumes system and problem decomposition, decisions what exactly to model and how; finally, as in any other design process, intermediate design steps are validated.

These modelling decisions are partly non-formal, they are creative, and non-predictable given that there are many models that suit the purpose. Given their non-formal nature, we cannot make these steps automated. Instead, to ensure model quality, different heuristics, standards, reusable solutions are used to steer the modeller's creativity and personal choices during the design process.

We focus on modelling for formal verification of embedded systems. More concretely, we are looking at systems in which embedded software controls the movement and positions of mechanical parts; the mechanical parts can manipulate some external products. For example, a system that conveys bottles and labels them is a system that has mechanical parts whose position and precision is important, and it also manipulates bottles that are not part of the system.

To prove that the requirement for the whole system holds we model both the plant (mechanical parts and processes controlled by the software) and the software. The existing techniques to structure and guide non-formal modelling decisions are helpful but they also have limitations. Some of the already existing reusable solutions cannot apply for the plant modelling because they are more focused on the software modelling. Other techniques are useful to roughly structure modelling, but they are too general and leave the modeller still quite large solution space to explore. Finally some techniques are closely related to the language chosen and cannot be used with another formalism. Many of the non-formal decisions stay implicit, so the reasoning that led to a model cannot be traced and reviewed.

The goal of our work is to characterise non-formal modelling decisions for the systems of our interest, and make them explicit. Our analysis aims at better understanding of the non-formal aspects of modelling decisions, but also it can serve as a framework that guides the modeller. In this paper we are looking at validation steps. The goal of validation is to increase confidence in the model, and consequently into the verification result.

The question we want to answer in this paper is: How can we validate the model in order to have confidence in the verification result? What are the modelling steps that have to be made explicit, and then examined, in order to increase confidence in the verification result based on the resulting model? We are focusing on the industrial systems where resources (time and people) are most often under tight constraints. There, not too much time can be spent validating the model; however, if a rigorous system verification based on models is chosen, some validation of models should be performed.

In the next section we will present a high-level conceptual analysis of non-formal modelling decisions with the special focus on the validation part. We will then describe the industrial case study of modelling a printer's performance.

978-1-4577-0959-3/11/$26.00 ©2011 IEEE

We will discuss related work and finally conclude with remaining open questions and future work.

II. Model validation

Formal verification is the way to prove mathematically that a model of a system satisfies (or not) some properties. In order to draw meaningful and correct conclusion about the system itself, the model should represent the system adequately. This means that all the system aspects and components relevant to the property should be described in the model. Also the model's properties that are subject of verification queries should encode the requirement we want to verify.

The system that is modelled is non-formal, it belongs to the physical world, and a model belongs to the mathematical, formal domain. Modelling steps are partly formal, partly non-formal, which results in lack of a formal proof that the model is correct. Instead, we have to rely on insight. The modelling steps must be explicit and structured so that we can understand them and be able to confirm or discard them. If we have broken down the modelling in such easy steps that can be understood, then we also have more confidence in the resulting model.

In some approaches only the software is modelled. As software is already an object in a formal language, its modelling can be formalized to a greater extent. But, to argue that the overall system behaviour is satisfied, an extra argument that makes a statement about the plant behaviour, is needed. This argument, again, cannot be a formal one, and we have again that a non-formal argument has to be made explicit and examined.

In the approaches in which we model both the plant and the software, we have the argument that 'connects' the software interface with the observable, end-user behaviour, formal as well. Still, there will always be assumptions and additional domain properties that have to be taken into account in the correctness argument. Altogether, part of the verification process consists of building the correctness argument of the model, and this can never be completely formal.

A. Roles and stakeholders in model design

The *modeller* is not necessarily someone who knows the system or the requirements, but it can merely be an expert in using formal languages and tools. So she needs other stakeholders to provide her knowledge about the requirements, the problem and the system.

Another role involved in modelling are *domain experts* – engineers specialised in different areas (e.g. mechanical, chemical engineers) whose responsibility is a certain aspect of the system (mechanical components, chemical processes etc.). For highly modularised systems, every component has a system engineer who is in charge of the component. For the modeller, the *component expert* is a source of knowledge about how the component works internally, and how it communicates with the outside world.

A *system architect* provides the modeller an overview how the system as a whole works, whereas a *requirements engineer* has an overview of all high-level and low-level requirements, and also can explain their rationale. Finally a *software architect* provides knowledge about software specification.

As users of the model, stakeholders have to have confidence in the results of modelling and sometimes they are using the model themselves to simulate, visualise and/or analyse the system.

B. Categorisation (Taxonomy) of Modelling Decisions

Figures 1 and 2 show our classification of non-formal modelling decisions.

As a design process, modelling comprises of problem analysis, solution design, and solution validation. However, as Hall and Rapanotti suggested [1], we are not validating only the design of our solution, but also our understanding of the problem. The steps of validation therefore intertwine with all the other steps. At the same time we are not claiming that the modeller will validate each and every step explicitly, it is a subjective decision of both the modeller and its users what modelling steps will be addressed, explained, questioned and validated.

Another categorisation of modelling distinguishes knowledge gathering and knowledge representation. Part of the modelling problem is deciding what to model, this is not given in advance. In order to decide what is relevant to model, the modeller needs to understand the modelling problem and how the system works. This knowledge is obtained by accessing documentation, talking to different roles and possibly using the system or its prototype.

During this process the modeller learns how the system is structured by different domain experts, and also makes her own structure of the system. Later this may be mapped into the structure of the model, but the model can be designed with a different structure as well.

Deciding what to model means that the system properties not relevant for the requirement are abstracted away. The relevant properties are represented in the model, by using different idealisations, abstractions and transformations.

C. Validation Aspects and Steps

1) Model Validity and Credibility: The modeller needs to validate her understanding of the problem and the solution steps. On the other hand in order to have the model used, the stakeholders should trust the model. Is it enough to only trust the modeller or more can be done? To establish the credibility of the model, the modeller can explain the model and build an argument that the model represents the system correctly. These are two different types of validation steps.

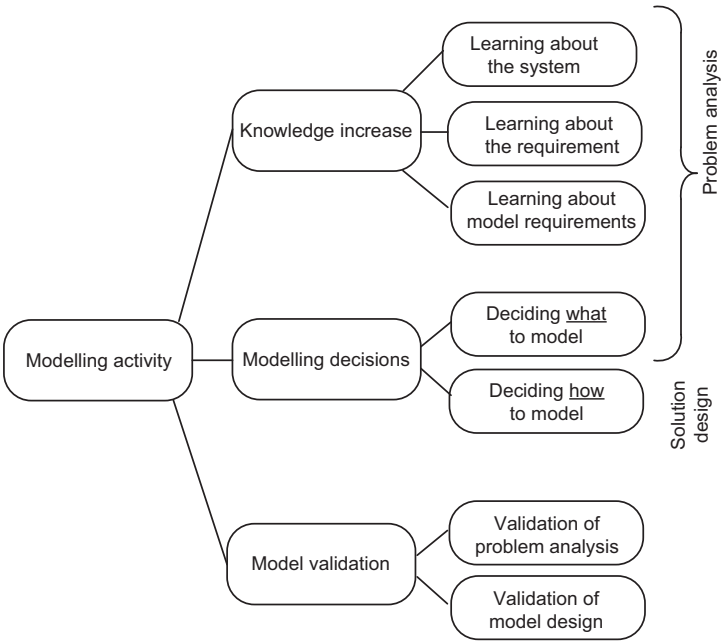

Figure 1. Taxonomy of modelling decisions.

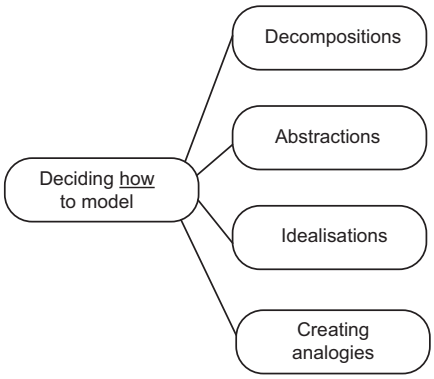

Figure 2. Taxonomy of modelling decisions (ctd).

III. Modelling Printer Performance - Case Study

In this section we will reflect on the case study we performed in a company. Our task was to design a performance model of a printer. What follows is not the explanation of the model itself, that would lead us to the topic of modelling scheduling processes and performance. What we want to present here is the validation aspect of modelling in industrial context. From the concrete validation steps we will generalize to lessons learnt for validation in general.

A. Introduction to the Case

We performed a case study in a company that produces printers for office use. Printers are produced in product lines to suit different purposes, such as printing on large formats and printing high-quality books and brochures. It is a large company, with a few hundred people employed in research and development department. As part of their business strategy the company has innovativeness as one of their core values. This results in encouraging innovativeness within the company and collaborating with academia and researcher institutes.

At the time we performed the case study, the printer that we modelled was in the one of the last development phases, with the plant and software already designed and tested. Some fine tuning of the control was discussed in order to improve performance requirements. Our work was related to performance estimation, therefore in the further text we will focus on this problem and the relevant printer aspects.

To execute printing, multiple printer components work synchronously, performing different mechanical, electrical and chemical processes with high precision and speed. The control software ensures components' correct behaviour in time and space. The control software also stores and transforms data to be printed.

We can decompose printing of documents into four main processes. They are as follows.

- Transporting paper sheets.
- Processing printing data.
- Forming toner image.
- Printing.

Paper sheets are moved along the paper path, from the trays to the module where the actual printing takes place. In case of double-sided printing, the sheet is turned over and brought back to the printing module, otherwise it proceeds to the exit tray.

After a print request, printing data is stored into the local printer memory. It is processed and transformed into a corresponding matrix of electromagnetic signals. The matrix further serves for forming the toner image.

Forming the toner image and printing are performed by two different, mutually connected, modules. The toner image is formed in the **Cold Process (CP) component**, and printing takes place in the **Warm Process (WP) component**. Obviously, the paper path passes through the WP component. The block diagram of the printer's main components is shown on Fig. 3.

Besides performing their main functions, the CP and WP component perform a number of supporting processes which are not directly part of the printing process. These processes prevent premature wear-out of the parts and keep them in a state that ensures maximal printout quality. For example, the belt along which a sheet of paper moves while the toner is applied to it, shifts its position occasionally, in order to prevent sharp edges of papers carving the belt and thus wearing it out prematurely.

Each supporting process starts under different conditions, at different points in time. Among conditions that trigger a supporting process are the following events: a certain number of pages is printed since the process is completed the

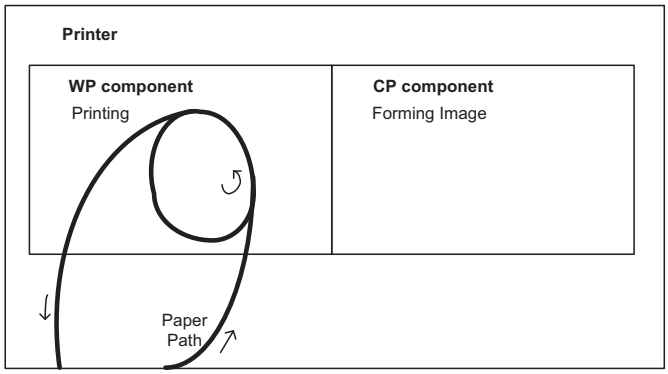

Figure 3. A diagram of the printer's main modules.

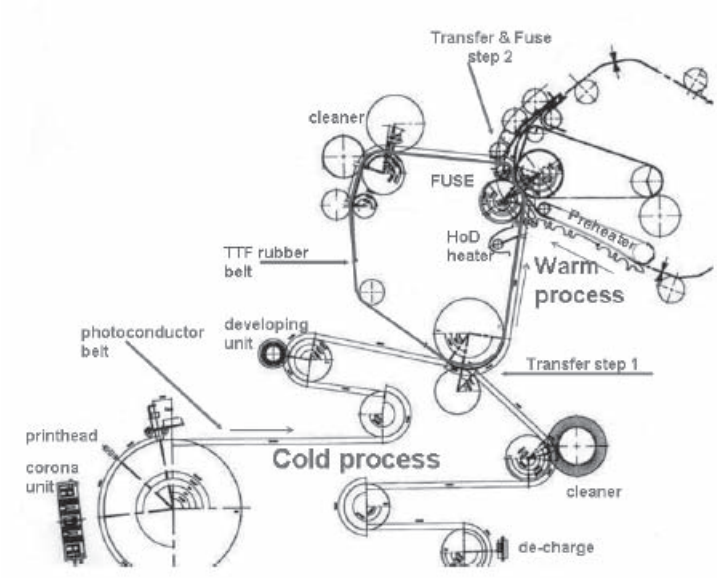

Figure 4. Processes and physical components performing them of a similar printer, taken from [2].

last time, a certain button is pressed by a user, a temperature of a part reached a critical value. Some of the processes need to be executed as soon as such an event occurs, some of them can be postponed. Furthermore, some of the processes can be interrupted if a print request arrives during their execution, some cannot. The execution times of the processes differ as well, from a couple of seconds to tens of minutes. The components performing these processes are shown on Fig. 4.

The printer states relevant for the performance are the Standby and Run states. The printer is in the Run state while it is printing. If for a certain period of time the printer is not printing and there are now new print requests the printer control puts the printer into standby state. Some of the supporting processes are done in the run state and some in standby state. There are also in between states, during which there are also standby processes that can be performed.

Apart from controlling printing, the printer control starts the supporting processes. They are executed in two parallel sequences of processes. This is because the control modules separately control CP and WP component so the processes done by them are independent, with exception of those processes who need to be performed by both modules at the same time. The CP and WP controls have to be in the same states, that is, it cannot happen that the CP control brings the CP component to the standby state, and WP control leaves the WP component in the run state.

The scheduling of the processes affects different printer's performance properties, so a trade off has to be made between the different features. For example, we could optimise printing so that the waiting time for printouts is minimal starting from the moment the first sheet in a batch is printed, or we could optimise the waiting time for the printing to start. The latter is more convenient in cases we need to print one or two pages.

B. The modelling task

The modelling task was to describe the printing process, supporting processes and their different scheduling strategies. The modelling language and tool used was Uppaal [3]. Uppaal is used to describe concurrent processes, it has a simulation tool, which was an important to the stakeholders.

Verification should enable comparison of different strategies and trade-offs between different performance requirements, like for example smallest number of interruption versus minimal total printing time, or minimal delay before starting a printing job.

The two scenarios in which the performance is measured are as follows. In the first scenario a user is printing a couple of pages, checks whether the printouts are good, and then prints a large number of pages. In the second scenario a large number of pages is printed.

C. The solution

Uppaal is a tool that integrates a description language, a simulator and a model-checker. "It is appropriate for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables." [3]

The high-level structure of the model we designed is shown on the Fig. 5. The squares on the diagram stand for an automaton or a set of automata describing relevant printer behaviours. The arrows informally describe which processes share variables or synchronised channels.

The processes and printing all have the same simple, two-state structure that describes beginning and finishing the process. They describe the plant, and therefore cannot interrupt, start or stop themselves, this is the task of the control. On the other 'end' of the control, the user sends printing request.

The task of the control is to start and stop processes via CP control and WP control components, and to directly change printer states. It implements the following strategies:

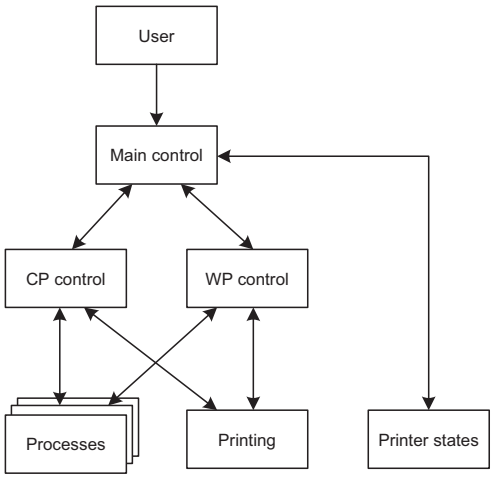

Figure 5. The high-level structure of the model describing the printer under different scheduling strategies.

S1 If a run is not finished, but interrupted for an urgent process, do only the urgent processes and continue printing.

S2 If a run is not finished, but interrupted for an urgent process, do the urgent processes. If there is an indication that during the current run a condition for other urgent processes will occur, perform them as well.

S3 If one of the CP or WP control is idle while waiting for the other to finish, check if there are other non-urgent processes that the idle sub-control can finish in the meantime and finish them.

S4 If the run is finished, do all the urgent and non-urgent support processes without checking if there was a print request issued in the meantime.

S5 If the run is finished, do all the urgent processes and check if there is a print request.

S6 If the run is finished, do processes by process, no matter whether it is urgent or no, and after each process check whether there is a print request. If there is, finish only the urgent processes and start printing.

D. Validation Steps

We performed two kinds of validations – testing the model itself by review, simulation and model-checking; and explaining the model to the model stakeholders. What follows are validation steps elaborated in the light of the taxonomy of modelling decisions.

1) Knowledge increase: The validation of the modeller's steps during knowledge increase goes into two directions. First, it ensures that the modeller understood the problem correctly and that the stakeholders gave correct information about the system and formulated the problem correctly. By understanding the problem we mean the following: (1) understanding how the system works, (2) understanding the system requirements, and (3) understanding the requirements for the model itself.

To discover how the system works, we used the existing documentation and talked to domain experts. We explored (1) how the control works, (2) how individual components shift from one state to another and (3) how processes and plant parts work. Our discovery was related to finding the right abstraction level to represent these components, for both the problem-owner and the modeller.

The problem owner gave too detailed descriptions of some components and simplified the others. The sources of model errors in the first versions of the model were: (1) the modeller did not understood correctly the documentation or the sketches of the domain experts; and (2) the problem-owner gave too detailed or too simplified explanations of some components – only after we explained, component by component how the model behaves, it became clear that changes in the model were necessary. For example, the documentation describes active, standby and in between states of the printer. It turned out that the printer as a whole, its modules, the control as a whole and control components are assigned states with the same names. Most of the time they mean the same thing, but not always. For example, the sub-control module that controls CP component will wait until all the parameters show that the CP component reached the standby state. It will then assign itself the same state to reflect and the situation in the plant and will let know the high-level control about this. Only when both CP and WP sub-controls report reaching the standby, the high-level control will also shift to the standby state.

An aspect of the problem validation is requirements elicitation. In this assignment, the main goal was to check how already a satisfactory performance can be improved. Modelling scenarios forced the problem-owner (the software architect) to be more precise, and brought insights and structure into their own perspective of the problem. This is inherent to requirements elicitation.

Another set of requirements were those for the model. Some of the requirements were already given us in advance, whereas some came up while explaining the model and possibilities and restrictions of the Uppaal tool and model-checking in principle. Also, there were the requirements to improve model's understandability by leaving out some simplifications. For example, we simplified the description of user behaviour, with an equivalent model that led to a valid verification result. However, the users wanted to simulate the scenario they were given, because it was easier to follow the simulation.

2) Modelling decisions: In the classification of the solution design steps we distinguish decomposition, abstractions, idealizations, and creating analogies.

Finding the right abstractions and idealizations is also part of the knowledge discovery and we identified what were three groups of components that we needed to validate and understand.

When it comes to decomposition, it was necessary to validate those components whose structure did not match the structure of the existing control. We decomposed the control into a number of automata with a smaller number of states and needed to come up with an argument that these two models (the existing control software specification) and the control in our model are equivalent with respect to the problem we modelled.

Creating analogies are 'modelling tricks' that simplify the model, by choosing a different decomposition or by designing a structure or an element in the model that does not exist in the system, but that results in modelling sharing the relevant behaviour or properties with the system.

To address the last two elements, we validated the model by designing queries that explored different safety and liveness model properties. In our model, the most important thing was to check whether all components are in a correct states, given that this was not obvious for some border conditions. We also had to model-check whether the order of processes execution was correct.

E. Model's Credibility

The steps we performed to validate the model for ourselves in the role of the modeller overlapped with the steps we need to perform to establish the credibility of the model for the problem-owner. Establishing model credibility ensures that the model-users have confidence in the model and the results they get from model-checking.

As we said, the model did not follow the system decomposition. One reason was simplification of the model and the other were Uppaal limitations, namely lack of hierarchies in Uppaal. So, we designed queries for the problem-owner to validate the model.

Explanations the model to establish credibility is another non-formal step, especially in the case in which the problem-owners do not 'speak' the modelling language. On the one hand, Uppaal language has almost the same syntax as the state charts used for software specification. Still, there are differences, and even though we explained what means what in the model, it is not really easy for someone who has not been using formal methods earlier to grasp the concept of urgent channels and other elements of Uppaal, that are not present in state charts.

To minimize the risk, we also developed additional test queries that addressed phenomena described with urgent channels and locations.

Another set of test were those that checked performance for initial conditions and parameters for which already the results were known or could have been easily calculated.

F. Verification Validity

We already mentioned that verification results hold under additional assumptions. We collected these assumptions and checked if they are fulfilled when printing is in the states we modelled. We delivered the list of assumptions together with the model. This list is never complete, but some of the assumptions are important to document, otherwise they can be easily overlooked.

We presented them to the domain experts in order to check their validity. We also classified them in order to check with the domain experts if there are more assumptions that we overlooked. Some of these assumptions always hold, and some are the idealisations that we introduced. Their role is to (1) document conditions under the modelling results are correct and (2) to test with the model-stakeholders if the idealisations we made still result in an adequate model.

Following assumptions and classes are just some of those we collected, given as illustration.

- Decomposition to: User, Environment, Plant, Paper, Control. Here for example, we documented an assumptions that during printing, the user will not try to open the printer door (this refers to the printer behaviour). Also, we documented that the time to forward the print request through the network is negligible. The first assumption is the condition under which the user can expect the performance that is promised by the producer. The latter assumptions is more an idealisation, but it should be explored what happens if there is a delay.
- Decomposition to processes performed on paper: printing, transporting. Here, one of the assumptions recorded was that printing takes 0.5 seconds. This is the assumption under which the result of the verification hold. It can happen that under some circumstances we need to take into consideration longer page printing times.
- Functional decomposition - here we focused on the assumptions we took about individual processes like:"Component X will never become too hot."
- Errors and faults - one of the assumption here is "Paper jam will not happen".
- Initialization and finalization concerns - here we addressed initial values of different counters.

IV. Related Work

The conceptual analysis we perform coincides partly with the work of Hall and Rapanotti. We adopted their model of designing of a solution to a problem with validation of both problem analysis and solution validation [1]. Also, the conceptual analysis of requirements engineering present in the work of goal-oriented community [4].

There are many standards that recommend the verification and modelling process, but as case studies show[5] it is too expensive to use them for the systems that are no safety-critical.

In social sciences, experiments are performed to validate simulation models and there is also a concept of a model credibility. An example how this is done is shown in [6] and

[7]. Also, for the simulation models in general, establishing validity is a known issue.

Finally, the tests that we performed on the model using the model-checker to establish the model's validity are part of the modeling practice in model-checking.

V. Discussion and Conclusion

The goal of this paper is to explore what non-formal modelling and verification steps should be validated in order to trust verification results. As we shown in our case study, there are different roles and stakeholders involved and there should be trust from all sides in the verification results.

For modelling the plant as a discrete system, we do no have an established theory to use like for example in physics or dynamic (control) systems where models and theories are well established and validated. Moreover, a lot of domain-specific knowledge that has to go into the model makes it difficult to identify a generic modelling pattern, or even impossible. It is always necessary to go into technical details of the system while modelling. As a result, most of the models are build from scratch. It is what Vincenti [8] called a radical design. Accordingly, the validation argument needs to be build from scratch as well.

As a 'way out', we propose to generalize within narrow classes of systems. In domain-specific cases, it is possible to identify modelling decisions and elements that can serve as guidelines and heuristics. In the case we presented, we started from a more general taxonomy and recorded the steps that will possibly useful for modelling the nexy system generation.

Another lesson learned from our case study is, that when it comes to increasing the stakeholders' confidence in the model, it can be a negotiation process. For complex industrial systems, one model does not cover all the aspects and the requirements, and for some analysis, a model that is not 'perfect' may be good enough.

Finally, there is a danger in explaining the model to the stakeholders that are not experts in the language and the methods that the modeller used. In our example, Uppaal diagrams look very much like some of the state diagrams the stakeholders used, but there are semantic differences that can change the complete meaning of the model.

References

[1] J. G. Hall and L. Rapanotti, "The discipline of natural design," in *Proceedings of the Design Research Society Conference 2008, 16-19 July, 2008, Sheffield, UK*. Sheffield Hallam University Research Archive, 2008, 16-19 July 2008.

[2] M. Heemels and G. Muller, Eds., *Boderc: Model-based design of high-tech systems*. Embedded Systems Institute, Eindhoven The Netherlands, 2006.

[3] "UPPAAL home page," *http://www.uppaal.com*.

[4] A. Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley, 2009. [Online]. Available: http://books.google.com/books?id=AYk_AQAAIAAJ

[5] S. Robinson and R. J. Brooks, "Independent verification and validation of an industrial simulation model," *Simulation*, vol. 86, pp. 405–416, July 2010. [Online]. Available: http://dx.doi.org/10.1177/0037549709341582

[6] K. M. Carley, "Validating computational models," 1996.

[7] J. Thomsen, R. E. Levitt, J. C. Kunz, C. I. Nass, and D. B. Fridsma, "A trajectory for validating computational emulation models of organizations," *Computational & Mathematical Organization Theory*, vol. 5, pp. 385–401, 1999, 10.1023/A:1009624719571. [Online]. Available: http://dx.doi.org/10.1023/A:1009624719571

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