Formalizing the Concept Phase of Product Development *

Mathijs Schuts[1] and Jozef Hooman[2,3]

[1] Philips Healthcare, Best, The Netherlands
[2] Embedded Systems Innovation by TNO, Eindhoven, The Netherlands
[3] Radboud University, Nijmegen, The Netherlands

Abstract. We discuss the use of formal techniques to improve the concept phase of product realisation. As an industrial application, a new concept of interventional X-ray systems has been formalized, using model checking techniques and the simulation of formal models.

1 Introduction

Traditionally, during the concept phase an informal document is being created with a high level description of the concept. This document consists of a decomposition of the product to be developed, the different hardware and software components it consists of, the responsibilities per component, and the interaction between the components. From the concept description, different development groups concurrently start developing the component they are responsible for.

A frequently occurring problem in industry is that the integration and validation phase takes a large amount of time and is rather uncontrollable because many problems are detected in this phase. An important reason for these problems is the informal nature of the concept phase. This leads to ambiguities, inconsistencies, and omissions. Typically, a large part of system behaviour is implicitly defined during the implementation phase. If multiple development groups work in parallel in realizing the concept, the integration phase can take a lot of time because the independently developed components do not work together seamlessly. Moreover, during the integration phase sometimes issues are found in which hardware is involved. Then it is usually too late to change the hardware and a workaround in software has to be found and implemented.

To prevent these types of problems, we investigate the use of formal modelling techniques in the concepts phase, because all consecutive phases can benefit from an improved concept description. We report about our experiences with model checking and simulation of formal models in a real development project concerning the start-up and shut-down of interventional X-ray systems of Philips.

2 Industrial Application

The interventional X-ray systems of Philips are intended for minimally invasive treatment of mainly cardiac and vascular diseases. For a new product release,

* This research was supported by the Dutch national program COMMIT.

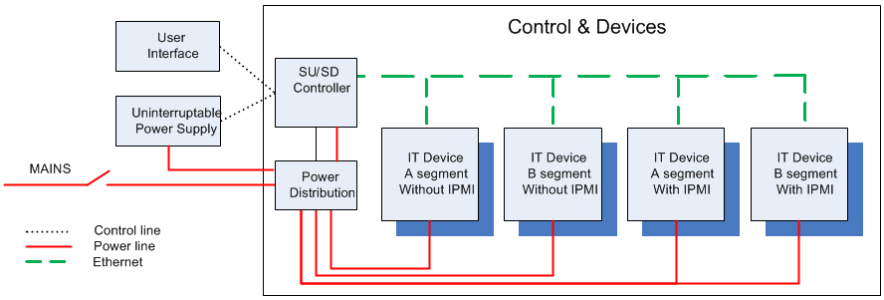

Fig. 1. System overview

we had to create a new concept for starting up and shutting down the system. This new start-up/shut-down (SU/SD) behaviour includes power failure scenarios where graceful degradation mechanisms should ensure that crucial functionality remains operational.

An interventional X-ray system contains a number of IT devices such as computers and touch screen modules. All IT devices can communicate with each other via an internal Ethernet control network. There is a central SU/SD controller which coordinates SU/SD scenarios. A user of the system can initiate a SU/SD scenario by pressing a button on the User Interface (UI). Another scenario can be initiated by the Uninterruptable Power Supply (UPS), for instance, when mains power source fails or when mains power recovers.

The system is partitioned into two segments: A and B[4]. This partitioning is mainly used in the case of a power failure. When all segments are powered and the mains power is lost, the UPS takes over. Once this happens, the A segment is shut down in a controlled way, leaving the B segment powered by the battery of the UPS. If the battery energy level of the UPS becomes critical, also the B segment is shut down in a controlled way.

The new SU/SD concept uses the Intelligent Platform Management Interface (IPMI), a standard interface to manage and monitor IT devices in a network. The IT devices in our system are either IPMI enabled or IPMI disabled. Combined with the two types of segments, this leads to four types of IT devices, as depicted in Figure 1. This figure also shows that there are several communication mechanisms; the internal Ethernet network, power lines for turning the power on and off, and control lines to connect the controller to the UI and the UPS.

3 Formal Techniques Applied

3.1 Model checking using mCRL2

We made an abstract model of the SU/SD concept using the mCRL2 model checker[5]. Making the model was very useful to clarify the main concepts and

[4] For reasons of confidentiality, some aspects have been renamed

[5] www.mcrl2.org

remove ambiguities. To allow model checking, this model does not include IPMI and the segments. Also timing aspects and error scenarios are ignored. Nevertheless, model checking such a model (78,088,550 states and 122,354,296 transitions) easily takes hours. The full concept is far more complex because of the many IT devices that all exhibit different behaviour and might fail to start-up or shut down. Moreover, these components are loosely coupled using asynchronous communication mechanisms, leading to a large number of message queues. Hence, we did not see a possibility to check the full model.

3.2 Simulation of POOSL models

As an alternative to increase the confidence in the concept, we used simulation using formal models expressed in the Parallel Object Oriented Specification Language (POOSL) [3]. POOSL is a modelling language for systems that include both software and digital hardware. It is an object-oriented language with concurrent parallel processes. Processes communicate by synchronous message passing along ports, similar to CSP and CCS. Progress of time can be represented and also stochastic distribution functions are supported.

The formal semantics of POOSL has been defined in [4] by means of a probabilistic structural operational semantics for the process layer and a probabilistic denotational semantics for the data layer. This semantics has been implemented in a high-speed simulation engine called Rotalumis. Recently, a modern Eclipse IDE has been developed on top of an improved Rotalumis simulation engine. The Eclipse IDE is free available[6] and supports advanced textual editing with early validation and extensive model debugging possibilities.

Application of POOSL

The aim was to model the Control & Devices part of Figure 1 in POOSL. Besides the SU/SD Controller and the Power Distribution, the model should contain all four types of IT devices, i.e., all combinations of segments (A and B) and IPMI support. Moreover, to capture as much as possible of the timing and ordering behaviour, we decided to include two instances of each type.

To be able to discuss the main concepts with stakeholders, we connect the POOSL model by means of a socket to a simulation environment of the Control & Devices part. The simulator allows sending commands from the User Interface and power components to the model and displaying information received from the model. Additionally, one can observe the status of IT devices and even influence the behaviour of these devices, e.g., to validate scenarios in which one or more IT devices do not start-up or shut down properly.

The simulator has been used to align the behaviour with stakeholders and to get confidence in the correctness of the behaviour. To increase the confidence without the need of many manual mouse clicks, we created a separate test environment in POOSL. It contains stubs which randomly decide if a device fails

[6] poosl.esi.nl

to start-up or shut-down. Moreover, observers are added to check conformance to component interfaces. The test environment leads to a deadlock when the SU/SD controller or the IT devices do not behave as intended. Already during the first simulation run we experienced such a deadlock. The cause of the problem was easily found using the debug possibilities of the new POOSL IDE.

4 Concluding remarks

Our experiences with the use of model-checking and formal simulation are similar to the observations of [1] on the application of formal methods early in the development process. They propose a rapid prototyping approach, where prototypes are tested against high level objectives. The difficulty to use formal methods early in the development process, when there are many uncertainties and information changes rapidly, is also observed in [2]. They investigated the use of formal simulations based on rewriting logic.

In our case, we successfully used a formal system description in POOSL in combination with a graphical user interface to align stakeholders and get confidence in the behaviour of the system. To increase the confidence in the concept, we created an automated test environment for the system with stubs that exhibit random behaviour and random timing.

While modelling, we found several issues that were not foreseen in the draft concept. We had to address issues that would otherwise have been postponed to the implementation phase and which might easily lead to integration problems. We observed that the definition of a formal executable model of the SU/SD system required a number of design choices.

In addition, the model triggered many discussions about the combined behaviour of the hardware and software involved in start-up and shut-down. This resulted in a clear description of responsibilities in the final concept. Also the exceptional system behaviour when errors occur has been elaborated much more compared to the traditional approach. Note that the modelling approach required a relatively small investment. The main POOSL model and the simulator were made in 40 hours by the first author, starting with very limited POOSL experience; the tester and the stubs required another 10 hours.

References

1. S. M. Easterbrook, R. R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Trans. Software Eng.*, 24(1):4–14, 1998.
2. A. Goodloe, C. A. Gunter, and M.-O. Stehr. Formal prototyping in early stages of protocol design. In *WITS'05*, pages 67–80. ACM, 2005.
3. B. D. Theelen, O. Florescu, M. Geilen, J. Huang, P.H.A. van der Putten, and J. Voeten. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *Proc. of MEMOCODE'07*, pages 139–148. IEEE, 2007.
4. L.J. van Bokhoven. Constructive tool design for formal languages; from semantics to executing models. Phd thesis, Eindhoven Univ. of Tech., The Netherlands, 2004.