Towards Failure Models and Error Propagation in Product Lines

Sara Bessling

Towards Failure Models and Error Propagation in Product Lines

Sara Bessling

Department of Informatics, Clausthal University of Technology
Clausthal-Zellerfeld, Germany
sara.bessling@tu-clausthal.de

Abstract. Safety-critical systems and especially their software components need a thorough verification for failures and potential error propagation. Reliability has to be guaranteed for medical devices in particular. These devices exhibit a broad variability, as well. They have to be suitable for a diverse variety of individual requirements leading to product lines which share a common base functionality, but each product is adapted to different requirements.
We present an approach in which failure models are assigned to features which are combined into different product models. Starting with a base model, further product models are derived from it by model transformations. We investigate the structure of the failure models and a possible error propagation. We demonstrate our method using SCADE Suite for the model-based product line design of cardiac pacemakers. Formal safety analysis is performed by using the SCADE Design Verifier.

1 Introduction

Software product lines are characterized by sharing functionalities grouped in features resulting in similar products. Regarding product lines of dependable systems, the application of formal approaches is complicated by restrictive formal product building mechanisms. Moreover, an integrated specification of product-specific failure models is missing during the design methodology. This restricts the usability of formal techniques to dependable systems product lines.

Currently, products of software product lines, especially for safety-critical systems, are verified individually resulting in very time consuming processes. If failure models are appended to a product model at a late design stage, the product model may need to be reinspected and possibly redesigned. An investigation of the product structure and its features, leading to adding failure models to a certain feature, is more profitable. The reduced functional range of a single feature simplifies the definition of safety constraints and failure models. We can revert to them when creating a new product out of single features. As features are already analyzed, we only have to concentrate on possible feature interaction.

We already investigated how software product lines and their features can be expanded by individual safety constraints in [8]. Moreover, we showed how individual products containing safety constraints can be derived from a single base

model by graph transformation. Following, we verified the individual product models with the SCADE Design Verifier.

We present an approach in which we append failure models to specific features. In order to accomplish this, we perform a model transformation in which not only features are combined into a new product but also failure models are transformed and attached to the new product at the same time. Architectural decomposition and functional structuring is not only applied on features, as it is well known for product lines, but on the safety constraints derived from them and their verification as well.

For demonstrating our approach we use the SCADE development framework for the phases of architectural and functional design. Our transformational approach shows that we have a seamless integration of failure models into product models within our feature-oriented transformational approach.

2 Basics

2.1 Dependable System Modeling Using SCADE

The acronym SCADE stands for Safety-Critical Application Development Environment. The main objectives of the SCADE Suite are (1) to support systematic, model-based development of correct software based on formal methods and (2) to cover the whole development process [5]. Its formal semantics is based on a synchronous model of computation.

The SCADE Suite is an integrated development environment that covers many development activities of a typical process for safety-critical software: modeling, formal verification using the SAT-based SCADE Design Verifier [2], certified automatic code generation producing readable C-code, requirements tracing down to model elements and code, simulation and testing on the code level.

2.2 AADL

The Architecture Analysis & Design Language (AADL) [7] is a formal declarative language to model system architectures consisting of software and hardware components. It was standardized by the SAE. AADL offers predefined elements to model system components and their connections. Furthermore it differentiates between a declaration of a certain system component and its explicit implementation. AADL can be extended by several annexes. The Error Model Annex [6] offers a predefined mean to express error models and their propagation.

2.3 Related Work

Like us, Liu et al. [11,12] consider a product line of pacemakers as a case study. Their sequential composition is less flexible compared to our synchronous product, as they assume that only one feature is currently active. UPPAAL is used by Jee et al. [9,10] to formally develop and verify the software control of pacemakers. As they investigate a product-centric assurance case in [9], a thoughtful

combination of methods and results from safety analysis, design and verification is needed. But [11] and [9] introduce other formalisms than we do, but the intended seamless integration of safety analysis, development and verification is similar to our approach. Sun et al. [1] concentrate on the integration of FTA results into AADL, whereas we use AADL to describe and derive failure models.

3 Adding Failure Models to Features

3.1 Defining Failure Models

Before starting with our method, we need a step ahead. Errors and their causes have to be identified by a safety analysis in advance. The exact method for the safety analysis is not relevant for our approach as long as we receive a list of errors and their causes. Assigning these errors to features is the first step in our approach.

We concentrate on software features which are modeled in detail, the associated hardware features are modeled as a hardware abstraction layer. The identified errors are analyzed in detail to find the exact cause and the place where the errors occurs first. Starting with a hazard, we identify errors leading to it. Then we analyze the specific errors trying to find its source. This source is a certain function belonging to a feature. The error is assigned to this feature. Possible error propagation can be identified through the search for the source if further features are affected afore. In the next step we have to classify the errors.

```
failure model <name>
states
  <ErrorFree>: initial state;
  <FailState>: state;
events
  <InError>: in event;
  <ErrorProp>: out event;
transitions
  <ErrorFree> - [<Condition/
      Event>] -> <FailState>;
  <FailState> - [<Condition/
      Event>] -> <ErrorFree>;
```

Listing 1. Template failure model

We differentiate between errors which are only singular affecting only one feature and errors leading to an error propagation. This has to be done as the failure models differ regarding these both error types. Error propagation leads to further or altered failure models in at least one further feature as these models have to react to incoming or outgoing propagated errors.

After the assignment and classification, we concentrate on the single failure models for each feature. Every feature, to which an error is attributed, is amended by a description for a failure model. This description is based on the AADL Error Annex [6]. We do not provide an implementation of a failure model at this point, only a definition. Our definition enfolds the states of the failure model and the transitions between the states. Further aspects are possible events for an incoming or outgoing (propagated) error. Error propagation can be understood as a broadcast of an event, as a propagated error does not have a specific goal. Propagation is carried out by means of certain rules. Error propagation can only occur along existing connections between features or in hierarchical

structures. A further possibility is propagation by resources. Features can use shared resources and this leads to an implicit connection between features. We call this effect mutual reaction.

A template for a failure model is shown in Listing 1. Template elements in chevrons are placeholders for names of the elements. Events are used as inputs or outputs. Outputs are an equivalent for the error propagation. The second transition has to be deleted for persistent errors.

3.2 Implementation of the Approach

In order to be able to verify different product models of a product line, we need to perform different steps. We develop a base product model which consists not only of single features, but also includes the safety constraints and failure models. This base model is modeled in SysML by using the SCADE System Designer. Each feature is modeled as a part in the Internal Block Diagram (ibd) of the product model. Further product models are derived from this base model by model transformations. These transformations are described in detail in [8]. The single product models are then imported into the SCADE Suite for verification. The SCADE Suite offers a special interface for converting SysML models into SCADE models, but only if the SysML models are annotated with special SCADE types. However before we can verify our product models, we have to implement the inner logic to the automatic compiled operators. Failure models are modeled as state machines with at least two states: one for normal and one for erroneous behavior.

4 Case Study: Product Line of CardiacPacemakers

The heart is a very complex biological system. The single heart beats are triggered by electric impulses which are transmitted over the cardiac conduction system. These impulses trigger the contraction of the cardiac chambers. Failures in this system lead to a misbehavior of the heart. A pacemaker is used to handle it. As there are several different points in the system where a single failure can occur, or even several ones in combination, different pacemakers exist to fit the possible failures and the resulting misbehavior.

4.1 The Pacemaker Product Line

Industrial pacemakers are categorized by an international code, the NASPE/BPEG Code [3] which we also use to structure the pacemaker's features as shown in Fig. 1. Its definition enfolds five letters. The first three letters characterize the main functions of a pacemaker as the stimulation of the heart, the detection of natural heart paces and the response mode to detection. The letters indicate the heart chambers affected by stimulation or detection, i.e. "V" denotes *ventricular* stimul3tion.

The mandatory functionalities for stimulation, sensing and the sensing response mode are classified into feature groups on the first layer. The stimulation features can be found in the group “chambers paced”, the features for sensing in “chambers sensed” and the ones for the sensing response mode in “sensing response”. The feature ”rate modulation” in the correspondent group is optional despite the AND connection between all features groups because we can choose between the feature itself and a “none (0)” feature. Such features do not offer any functionality and serve just for compliance to the NASPE/BPEG code. The feature “rate modulation” ist not a necessary feature, but it was first introduced as a comfort functionality which has become quite common. At the second layer, a single feature is selected from each group (XOR operator). But a feature from each group cannot be chosen arbitrarily as the groups condition each other. These dependencies are shown in Fig. 1 as lines with a “D” between the groups. For example the feature group sensing response depends on the feature group sensing. If the feature “none” is chosen in sensing, a selection of any other feature as “none” in sensing response is not reasonable.

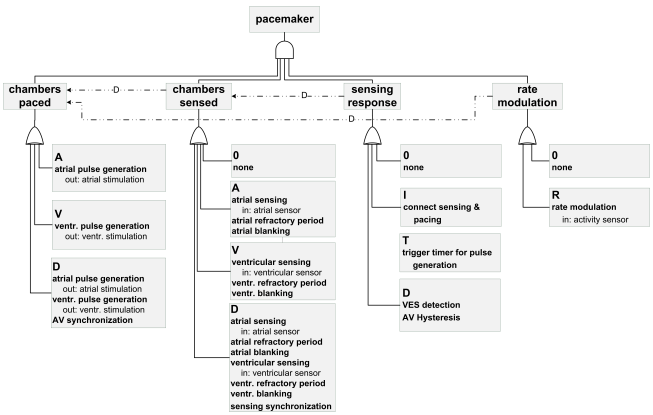

Fig. 1. Feature diagram of pacemaker product line

In the case study we investigate which software errors occur in a pacemaker. We analyze the pacemaker’s architecture with the help of an VVI pacemaker. This is a variant that stimulates the ventricle, in case no ventricular pace is detected. The pacemaker awaits a natural pace or a stimulation and then switches into the refractory period. A transformation is done into an DDDR pacemaker senses and stimulates both chambers and can adapt its pacing intervals to the patient’s physical stress. All behavioral details and an informal specification of the safety requirements originate from an industrial specification document by Boston Scientific [13].

4.2 Product Variability

Every product model is derived from a base model. The VVI pacemaker is the base model in this case because it is a central pacemaker regarding its functionality. To receive a DDDR pacemaker, several steps are needed. The ventricular

features are transformed into dual features by adding elements and altering the existing ones. The feature rate modulation has to added as a totally new feature including its connection to the residual features, as the VVI does not contain such a feature.

4.3 Architecture of the Pacemaker

The architecture of each pacemaker model is divided into two parts. We differentiate between hardware elements, respectively their abstractions, and software elements. The hardware structure of the pacemaker is reduced to four elements: a processor, a battery, memory and an electrode. The processor is modeled as system clock, whereas the battery and the memory do not offer a further functionality. Both elements only deliver values. The memory only provides the parameters of the pacemaker and the battery only a voltage value. The electrode is modeles as output. Depending on the chosen feature of the group "chambers sensed", an input is modeled as well for sensing natural cardiac paces. In case of dual features the model elements of the electrode are doubled to tend to each chamber. The sensor of the feature "rate modulation" is modeled as input if this feature is chosen providing a value for the activity measuring.

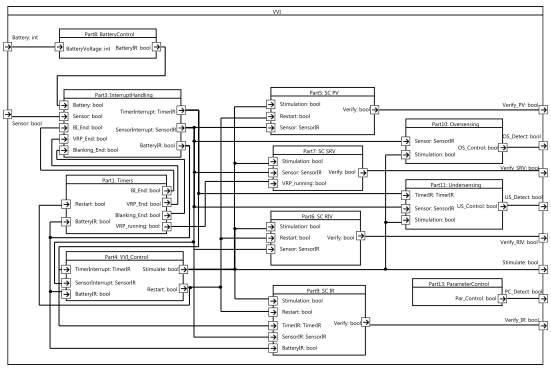

Fig. 2. Architecture of the VVI pacemaker

The software architecture of the pacemaker is divided into three parts on the first layer. The pacemaker's functionality is integrated into the blocks "Timer" and "Control". The "Control" block contains the functional logic of the pacemaker which is needed to detect natural paces at the right time for example. The "Timer" block includes counters which use the system clock. These timers control the pacemaker's time intervals. Both blocks are connected by the "InterruptHandling" which conducts the signals and events for the "Control". The "Control" itself can trigger counters in "Timer" like restarting them. An exemplary architecture of a VVI pacemaker is shown in Fig. 2. It includes further parts like observer nodes for the later verification and are attached to each feature. The observer nodes are first a representation for safety constraints regarding the pacemaker's behavior and second a control system to react to certain errors. The Safety constraints shall ensure that e.g. during a defined base interval only one stimulation occurs or during the same interval one stimulation or natural pace is detected. Further reading regarding the safety constraints is offered in [8].

4.4 Errors of the Pacemaker

We differentiate between three different groups of failures according to their origin: software errors, hardware errors and environment errors. Each of them can lead to a feature hazard or even a system hazard. The errors are grouped according an architecture scheme consisting of layers for software, hardware and environment.

In the Boston Scientific Product Performance Report [4] several errors for a pacemaker are noted. These errors are grouped into two categories: errors affecting the leads and errors affecting the pulse generation. We aggregate these errors into six errors and assign them to software or hardware errors. As lead errors we have undersensing, oversensing and a complete fallout of the electrode's pacing or sensing part. A complete fallout results in a total miss of paces to stimulate the heart respectively a total miss of all natural cardiac paces which shall inhibit the pacemaker. Undersensing, as well as oversensing, happens due to material failures of the lead or its dislocation. Undersensing means a sporadically loss of natural paces. Impulses in neighboring muscles of the heart are interpreted as cardiac paces in case of oversensing. A mechanical defect of the activity sensor for the rate modulation can form a further error. These errors belong to the group of hardware errors and do not lead to an error propagation. Missing or misinterpreted paces do not lead to an erroneous behaviour of further features. The pacemaker still works as expected, although its input is erroneous. Instead these errors lead directly to system hazards.

For the errors affecting the pulse generation we concentrate on two failures: a missing pulse generation itself and a reset of the time parameters. The first error describes a sporadically loss of a generated pulse although the electrode is not defected and the pacemaker is working correctly apart from that. This error can result from a missing interrupt signal which shall trigger the stimulation. A reset of time parameters is connected to memory errors. A pacemaker owns default parameters which are changed to fit the timing of the patient's heart. A memory error can lead to a permanent reset of the timing parameters to the default ones. Furthermore the detection algorithm for natural paces can misinterpret natural paces and thus leads to behavior similar to over- or undersensing. Another algorithm failure can arise in case of rate modulation by interpreting signals and activity sensor values falsely. These errrors are regarded as software failures. As well as the hardware errors, these errors do not lead to an error propagation.

A special case are errors concerning the battery and its charge. These errors can be software or hardware failures. If the charge drops below a certain value, the pacemaker will generate too weak stimulation pulses if at all. Furthermore, a charge measurement is integrated into the pacemaker software which can measure wrong values leading to a software failure. This error triggers an error propagation, manifesting in the consecutively shut down of each feature due to low charge. Errors regarding the environment of the pacemaker can concern the heart itself. A fibrosis can arise around the electrode and thus prevent it from sensing, pacing or even both. This triggers an error propagation as well.

4.5 Fault Injection and Error Propagation

For every error mentioned, a failure model is integrated into the single features. An exemplary failure model description is shown in List. 2. This failure model owns an error propagation and triggers further failure models. Further, observer nodes are included into the product model reacting if a certain erroneous behavior happens. In case of undersensing, oversensing and memory errors, the pacemaker cannot react to these errors. Instead the pacemaker only records its behavior and if it detects a certain pattern, a possible error is implied. A physician analyzes the pacemaker's protocol and decides if an error really occurred. Thus these observer nodes are a part of a warning system.

```
failure model battery
states
  HighVoltage: initial state;
  LowVoltage: state;
events
  DropVoltage: in event;
  VoltageDropped: out event;
transitions
  HighVoltage - [DropVoltage]
      -> LowVoltage;
```

Listing 2. Failure model battery

We added failure models to the pacemaker in combination with a simple heart model. The heart model guarantees that a natural pace does not occur in every base interval cycle. A further restriction is that not all natural paces occur during the blanking period. The aim of the following control safety constraints is a falsification during the verification if an error occurs. If no error occurs, they will be valid during the verification.

- *Undersensing Control (UC)* If a specific number of stimulated paces occur in a defined interval, under-sensing is detected.
- *Oversensing Control (OC)* If a specific number of sensed paces occur in a defined interval shortly after the end of the refractory period, oversensing is detected.
- *Parameter Control (PC)* The time parameters have not changed iff the time parameters of time n+1 are the same as at time n.

In summary all safety constraints except the control safety constraints stayed valid as the single failures were triggered. We anticipated this result as the errors have mainly only one impact on the heart. The pacemaker is still working correctly according to its safety constraints. We investigated further what happens, if the battery depletes. Only this error leads to a falsification of the regular safety constraints.

5 Conclusion

Our transformational model-based approach offers an integration of failure models into product models for product lines of safety-critical systems. Possible errors were analyzed and converted into failure models. Then we demonstrated the approach within the SCADE framework for the modeling and verification of dependable medical device software.

Medical products need a comparatively high adaption to the individual requirements of each patient. This leads to complex product lines with the demand for a high dependability. By allocating failure models to single features, we can reduce this complexity but preserve the requirement for a highly dependable system. SCADE Suite is approved in practice and provides certified code generation for dependable systems according to several safety standards and, moreover, formal verification by STA-based model checking.

Our next step will be the development of a safety analysis method for features of product lines, so that we have a formal methodology to find possible errors and hazards.

References

1. Integrating Product-Line Fault Tree Analysis into AADL Models (2007), http://dx.doi.org/10.1109/hase.2007.28
2. Abdulla, Deneux, Stålmarck, Ågren, Åkerlund: Designing safe, reliable systems using Scade. In: Intern. Symp. On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA). LNCS, vol. 4313, pp. 115–129. Springer (2004)
3. Bernstein, Daubert, Fletcher, Hayes, Lüderitz, Reynolds, Schoenfeld, Sutton: The revised NASPE/BPEG generic code for antibradycardia, adaptive-rate, and multisite pacing. Journal of Pacing and Clinical Electrophysiology 25, 260 – 264 (2002)
4. Boston: Crm product performance report 2012 - q3 edition (09 2012)
5. Esterel Technologies: SCADE Suite KCG 6.1: Safety case report of KCG 6.1.2 (July 2009)
6. Feiler, P., Rugina, A.: Dependability modeling with the architecture analysis & design language (AADL). Tech. rep., Software Engineering Institute, CMU (July 2007)
7. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 1st edn. (2012)
8. Huhn, M., Bessling, S.: Enhancing product line development by safety requirements and verification. In: 2nd International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2012). LNCS, Springer (2013)
9. Jee, Lee, Sokolsky: Assurance cases in model-driven development of the pacemaker software. In: Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part II. pp. 343–356. Springer-Verlag, Berlin, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1939345.1939383
10. Jee, Wang, Kim, Lee, Sokolsky, Lee: A safety-assured development approach for real-time software. In: Proceedings of the 2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications. pp. 133–142. IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/RTCSA.2010.42
11. Liu, J., Basu, S., Lutz, R.R.: Compositional model checking of software product lines using variation point obligations. Autom. Softw. Eng. 18(1), 39–76 (2011)
12. Liu, J., Dehlinger, J., Lutz, R.R.: Safety analysis of software product lines using state-based modeling. The Journal of Systems and Software 80, 1879–1892 (2007)
13. Scientific, B.: PACEMAKER System Specification (Jan 2007)