Formal Safety Analysis and Verification in the Model Driven Development of a Pacemaker Product Line

Sara Bessling Michaela Huhn
Department of Informatics, Clausthal University of Technology
38678 Clausthal-Zellerfeld, Germany
{Sara.Bessling|Michaela.Huhn}@tu-clausthal.de

Abstract: Cardiac pacemakers are a popular showcase for formal methods in the development of dependable, software-controlled embedded systems. We present the pacemaker as a case study on the product-line development of certifiable safety-critical software using SCADE Suite. The product line and and its products are specified by means of the *Common Variability Language* (CVL). CVL separates the variability modeling from the domain modeling of the base elements. CVL enforces a strict structuring of the product models (done in SCADE) that reflects the substitution concepts used to describe variability. From a formal safety analysis we derive a set of safety requirements which we can prove valid on the family of pacemaker product models by straightforward model checker using the built-in Design Verifier.

This positive result indicates that the strict design discipline imposed by the formal variability modeling pays off in verification with a better applicability of automated state space reduction techniques.

1 Introduction

The tight integration of computational and mechatronic devices, nowadays called *cyber-physical systems*, has lead to numerous new applications in multiple domains. Whereas in classical domains like transport, defense or industrial automation, dependability has been a key issue for decades, software safety has become a concern in the new areas like living assistance, home-based medical support or smart home applications only recently.

With the prospects of these markets also the demand grows for software development methodologies and tools that support both model-based design and the safety process. However, development conditions for these new cyberphysical systems slightly differ from the traditional large scale industrial processes performed for classical dependable systems: Since the applications aim at a mass market, variability of products as well as efficient development are essential. This leads to the request for seamless, tool-supported integration of artifacts and activities from different development phases and a higher degree of automation, not only for code generation but also for the safety analysis and resulting verification tasks.

Here we consider a product line development for cardiac pacemakers. In order to model variability within the product line we decided for the Common Variability Language (CVL) [HMPO$^+$08], as an EMF-based prototype implementation is available and CVL is proposed for standardization by the OMG. Even more important, CVL separates the variability modeling from the domain modeling used to design the base elements of the product line. For the domain model we use the SCADE Suite as development framework, since SCADE is qualified for the design of certifiable, dependable software [Est09] and provides a ready to market automated verification facilities. Our motivation for applying a systematic product line approach on the family of pacemaker origins from the disillusioning results for automated verification we experienced in previous work [HB11, DHM11]: Even for medium size industrial design models, fully automated verification yields only very few results due to complexity problems but most properties could be proven only when applying advanced abstraction techniques manually. Thus the question arises whether the strict structuring and the formally defined replacements and substitutions needed for the product line specification will have a (hopefully positive) impact on verification tasks. The rationale for that is that product line systematics will be reflected in the design and give structural insights how to decompose safety requirements and where to apply verification heuristics in the design model for the products.

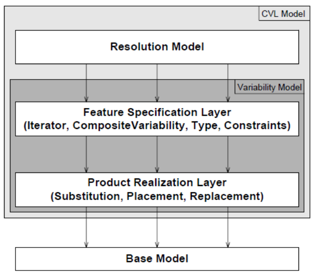
The contribution of this paper is twofold: (1) We model a pacemaker product line by using CVL for the variability modeling and SCADE to model the domain. (2) We provide a showcase for the efficiency gain one may obtain from a strict product line design approach in the subsequent verification: We can easily prove a set of safety requirements to be valid for the pacemaker variants by fully automated model checking.

The rest of the paper is organized as follows: Sec. 2 sketches the basics. The pacemaker is introduced in Sec. 3. Sec. 4 describes the design of a product line for a family of pacemakers for which safety analysis and verification is presented in Sec. 5. In Sec. 6 we conclude.

2 Background

2.1 CVL - Common Variability Language

The Common Variability Language (CVL) [HMPO$^+$08] is an approach to specify product lines upon a domain specific modeling language (DSL). CVL adds the possibility to model product lines or product variabilities on these models. To be able to model a product line with CVL one has to understand first the different layers of CVL (see fig. 1). The variability model of CVL consists of two layers, the feature specification layer (FSL) and the product realization layer (PRL). In the FSL the elements defining a particular product variability are specified from a user's point of view. The PRL consists of the definitions needed to build one product of the product line from the base models and their modifications. Beneath FSL and the PRL the base model has to be provided as the starting


CVL Model
Resolution Model
Variability Model
Feature Specification Layer
(Iterator, CompositeVariability, Type, Constraints)
Product Realization Layer
(Substitution, Placement, Replacement)
Base Model
Figure 1: CVL layers, source: [cvl10]

point of the product line from which all possible product variabilities are derived. The base model and the library models are modeled in a DSL or UML. A single product variability is described by a resolution model, the last element of the CVL layers. A resolution model consists of a subset of elements of the PRL and FSL. With the help of these resolution models, new variability models can be created by a model-to-model transformation.

2.2 The SCADE Tool Suite

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 [Est09]. The language *Scade* underlying the tool is data-flow oriented. Its formal semantics is based on a synchronous model of computation, i.e. cyclic execution of the model. The SCADE Suite is an integrated development environment that covers many development activities like modeling, formal verification using the SAT-based SCADE Design Verifier [ADS$^+$04], certified automatic code generation, requirements tracing, simulation and testing on the code level, inclusive coverage metrics.

2.3 Deductive Cause Consequence Analysis

A major goal in safety analysis is to determine how faults modes at the component level causally relate to system hazards. Among the various formally founded techniques proposed for this task we have selected *Deductive Cause Consequence Analysis (DCCA)* by Ortmeier et al. [ORS06, GOR07], because DCCA does not only formalize techniques like FTA (*Fault Tree Analysis*) and an FMEA (*Failure Mode and Effect Analysis*) [IEC06], which are well-established and recommended by the standards. In addition, the identified fault modes and hazards can be reused in safety assurance to formally verify that sufficient measures have been taken to prevent the identified hazards (for details and formalization see [GOR07, DHM11]). Hazards are specified as observer nodes that read signals from the control logic and evaluate them according to the *negation* of the hazard predicate.

2.4 Related Work

Since PACEMAKER Formal Methods Challenge [Sci07], pacemakers were investigated intensively within the formal methods community. For brevity, we only refer to three of them: Jee, Lee, and Sokolsky worked on assurance cases of the pacemaker software [JLS10]. They focused on the basic VVI mode, and employed UPPAAL for both, design and verification. In [TZT10] the authors used timed CSP to specify a number of pacemaker modes and to verify several timing constraints for them. Liu et al. [LDL07] explicitly model a product line of pacemakers and address its safety analysis.

3 The Pacemaker

3.1 The Human Heart

From a bio-mechanical point of view, the human heart is the pump of the circulatory system. It consists of two atria and two ventricles. The contraction of the heart is initiated at the so-called *sinoatrial node (SA node)*, an area of self-excitable cells within the right atrium known as *P wave*. The electrical impulses spread through the atria and ventricles with a dedicated timing characteristics (see the *electrocardiogram (EKG)* shown in Fig. 2), thereby causing the contraction of the chambers.

Nowadays, a too low or sporadically missing pulse generation by the SA node, called *bradycardia*, and defects in the cardiac conduction system are cured by an implanted artificial cardiac pacemaker. Artificial pacemakers have to respect the timing characteristics of the sinus rhythms as they are critical. Foremost, pulses must not be generated within the refractory intervals after depolarization, as this may cause life-threatening cardiac fibrillation.

3.2 Informal Specification of Pacemakers

The most complex pacemakers are variants of the modern DDD mode [Sci07]. Here we consider a family of pacemakers with the aim to build a product line out of them. The functions of a pacemaker are given by a sequence of letters according to the international NASPE/BPEG Code [BDF+02].

The simplest pacemakers stimulate the heart, whenever a timer expires without sensing the natural pace. The A00 and V00 mode stimulate either the atrium or ventricle, respectively. The D00 stimulates both, atrium and ventricle. For this, it has two consecutive timers monitoring a modified base interval (BI) and the AV interval (AVI). The AVI is the time between an atrial impulse and a ventricle one. The modified BI of D00 is the time between a ventricle impulse and an atrial one.

Todays most common mode is the VVI mode. It only stimulates the heart - in case of the VVI of the ventricle - when the natural pace is missing. The VVI uses a timer for the base interval as well to trigger pulse generation. The BI of the

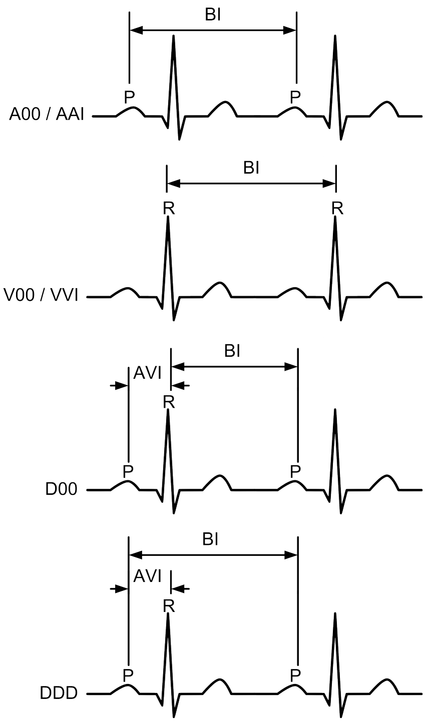


Figure 2: Intervals of pacemaker variants

VVI is the maximal time between two ventricular paces. But the BI is reset if a natural pace of the ventricle is detected and no stimulation takes place in this case. After each artificial or natural pace the pacemaker starts a refractory period in which no detection or stimulation can occur. The pacemaker mode AAI functions analogously for the atrium.

DDD means *dual pacing, dual sensing, and dual response mode*, see the NASPE/BPEG code. A DDD pacemaker senses both right chambers and can also stimulate them both, if no natural pulses are detected. The DDD pacemaker uses two timers for each chamber. The AVI is the time between an atrial pace and a ventricular one, the BI is the time between two atrial paces. Moreover, the DDD mode adapts the AVI (so-called AV hysteresis) for the next cycle, in case a stimulation occurs and it is able to handle a ventricular extra-systole (VES), i.e. an additional ventricle pace occurring in a particular interval. The DDD pacemaker also implements other modes: In case of a battery voltage drop it switches to VVI mode in order to save energy. The D00 mode is also integrated for usage during a magnet test, which may be performed by a doctor to check the pacemaker's function. The single intervals of each pacemaker are illustrated in Fig. 2.

4 Creating a Product Line of Pacemakers

4.1 Pacemaker Modeling Objects

Pacemakers are constructed in several variants, differing in possible features and functions. Furthermore, enhanced pacemakers implement simpler ones as an emergency or diagnostic mode. Thus a product line built from the various modes is helpful to illustrate these relations and even more important to simplify the modeling of the pacemaker modes.

In order to create a product line due to [cvl10], first we have to identify the base elements of the pacemaker modes. We start with the simplest modes, the A00 or V00 pacemaker which consist of the same modeling objects. The A00 and the V00 differ only in the length of the base interval which is a single property of a modeling object. So we identify the following three modeling objects which are depicted in Fig. 3:

- *Timer*: starts at zero and increments by one at each cycle
- *Reset*: resets the timer to zero when it reaches the end of the base interval
- *Output*: stimulates the heart chamber whenever a reset occurs

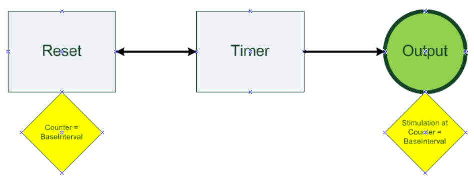


Figure 3: Schematic use of modeling objects to build an A00 pacemaker

Starting with these three base elements we study the D00 pacemaker. For each heart chamber we can reuse the known modeling objects but we have to adapt the timer interval to the atrium ventricle interval (AVI). Furthermore, we decide to rename when doubling the objects because of clarity. The timeout of the second timer is set to the sum of the BI and AVI because we can use the reset this way without changes. To model the consecutive order of the BI and AVI interval we insert a new element, the *block*. It blocks the restart of the AVI timer until the reset of the second timer occurs.

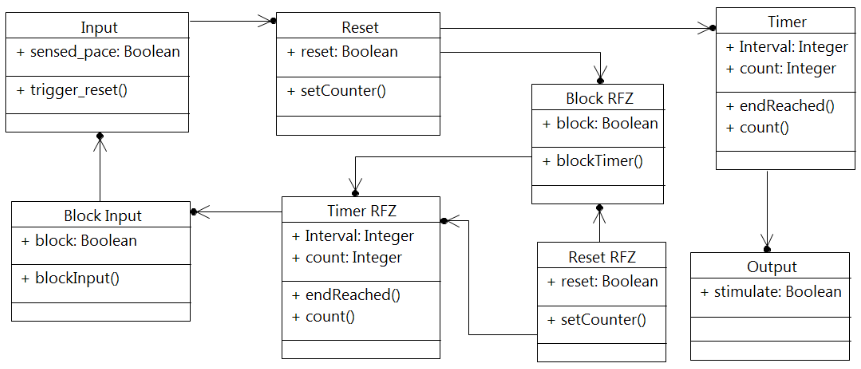


Figure 4: UML model of AAI pacemaker

Next we consider the AAI and VVI pacemaker, respectively. The AAI and VVI pacemaker coincide regarding the modeling objects as well as the base interval. In comparison to the A00 pacemaker we have an additional *input* to sense the heart's impulses as a new modeling object and a refractory period which blocks the input. Hence we add an input which triggers the reset whenever a heart impulse occurs. Moreover, an additional timer and reset object for the refractory period are introduced and connected using block objects.

The DDD pacemaker can be viewed as combination of two AAI pacemaker models. They are connected by a block object which is used for the AVI timer to wait for the reset of the BI timer. In order to adapt the hysteresis (AVI) and to detect a ventricular extra-systole (VES), we insert a new modeling object called *hysteresis* which can dynamically change the timeout value in the AVI timer in the DDD pacemaker and a hysteresis object which is connected with the output for the ventricle and the ventricle timer.

The final modeling object *VES* is connected with the ventricular input to detect a natural pace. To regard the time constraints the VES object is also connected with the atrial input block (detect if refractory period is over) and the AVI timer block (detect if AVI timer is stopped). Furthermore, the VES is connected with the atrial reset and the block of the ventricular refractory period.

4.2 Modeling a Pacemaker Product Line with CVL

After identifying the modeling objects which represent single functions of the pacemaker, we model the different pacemakers in UML.We decide to define the AAI model as base model (see Fig. 4) and the models of the other pacemaker modes as library models. These UML models consist of several classes which represent the already defined modeling objects. The FSL of the CVL variability model consists just of the single modes of pacemaker because they represent the different products of our product line. In the PRL we insert several replacements which consist not just of single modeling objects but of whole groups of modeling objects. This is done to keep the associations between the single classes. We derive an A00 pacemaker from the AAI model by eliminating all objects we do not need.

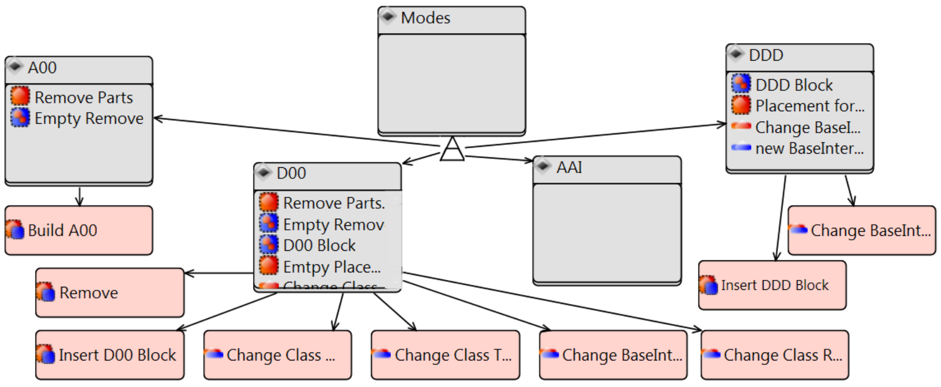


Figure 5: CVL model of pacemaker product line

To build the D00 pacemaker we first eliminate all objects to receive an A00 pacemaker and then we add the D00 library model which includes a V00 model combined with a block element. The DDD pacemaker is built by adding the DDD library model to the AAI base model. Furthermore we have to modify several values like class names and timeout values in the DDD and D00 product line descriptions. Several connections between classes had to be redefined, too, as we combine the base model with each of the library models. The complete CVL model is shown in Fig. 5.

4.3 A Pacemaker Product Line in SCADE

We use the named function structuring of the pacemaker modes again to model them in SCADE. In SCADE we construct a family of data flow models corresponding to the CVL modeling objects. I.e. we employ SCADE as the DSL for the base model. SCADE provides no means to build product lines, but we can mimicry the product line modeling by modeling the CVL base elements with their behavior as operators and then reuse them and combine fragments

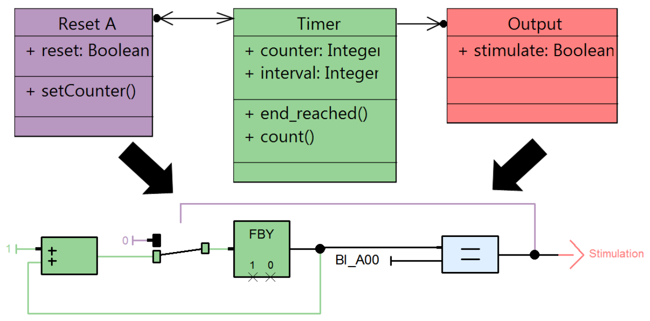


Figure 6: From CVL model to SCADE model

when building the different variants. Hence we start with the simplest pacemaker to ease reuse of already existing parts and operators in more complex ones. An exemplary development starting with a CVL description, generating a UML model und resulting in a SCADE model is shown exemplary for an A00 pacemaker in Fig. 6.

So far, we transfer the CVL model manually and do not implement an interface for an automatic transformation from UML or SysML to SCADE because Esterel announced such an interface for the new version SCADE 6.3, to be published in early 2012.

5 The Safety Analysis and Verification

The safety standards [Int06, Com10] prescribe a safety analysis that identifies hazards and traces them back to potential failures. From the identified hazards, system safety requirements are deduced that are capable to eliminate failures or mitigate the effects. In consistency with the architectural decomposition of the system (see Fig. 7), the safety requirements are split into subrequirements that are assigned to individual components. This process is iterated until basic software and hardware components are derived that are to be realized and for which evidence has to be provided that they fulfill their safety requirements.

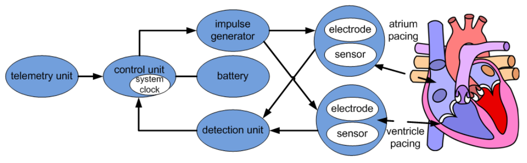


Figure 7: Principal architecture of a cardiac pacemaker

5.1 Safety Analysis

For safety analysis we focus on those hazards and the induced safety requirements that refer to the *functional level* of the software control of the pacemaker whereas the mechanics, the electrics, and the deployment are analyzed no further. We performed an FTA and an FMEA [IEC06] and formalized it according to the DCCA approach [GOR07]. The resulting safety requirements for the functional level of the software control are as follows:

Refractory periods: Within the refractory periods after the atrium (ARP) and ventricle pace (VRP), detection and impulse generation have to pause in that chamber in order to guarantee that neither an artificial pulse is sensed nor disturbances after depolarization are misinterpreted. *Time intervals BI, AVI + AVH:* The timing constraints as the base interval, the AV interval with the AV hysteresis and their sequencing are respected within specified tolerances. *Pacing:* An artificial atrium pace is triggered if the base interval expires without sensing a natural P wave. An artificial ventricle impulse is generated iff the AV timer was started and expires without sensing a natural pace there.

5.2 Verification of Safety Requirements

As described in Sec. 4 we designed a product line of pacemakers using the SCADE Suite. From the models, C-code is generated automatically by the certified code generator. Next we verify the safety requirements that we derived in the safety analysis. SCADE Design Verifier offers SAT-based model checking of reachability properties [ADS+04]. Therefore the predicate has to be encoded in an observer node and connected to the design model. Then SCADE Design Verifier searches through all possible input combinations and traces whether the predicate can be falsified. In case such an input configuration is found, this configuration and its trace yields a witness for the violation of the predicate. If model checking results in a positive answer, the property holds on all reachable states.

From the safety analysis we derived eight constraints to be verified. The notation of the constraints is according to the NASPE/BPEG Code including placeholders. x is the placeholder for any possible letter at that position, $\overline{0}$ stands for any possibility except 0.

- One pace x00 (mode without sensing)
  During the base interval, an artificial pace occurs exactly once.
- At most one pace x$\overline{0}$x (mode with sensing)
  Within each a specified interval at most one (natural or artificial) pace occurs.
- At least one pace
  If no natural pace is detected in a specified interval, an artificial pace is triggered exactly once.
- Atrial pace D$\overline{0}$x (D with sensing)
  A natural or artificial atrial pace takes place only once in the BI and after the end of the AV Interval.
- Refractory periods
  During a refractory period neither a pace detection nor a stimulation occurs.
- No VES
  If no ventricular extra-systole is detected in the specified interval then a natural or an artificial pace occurs exactly once until the end of the BI.
- VES
  If a ventricular extra-systole is detected then no atrial pace is sensed and no artificial pace is triggered during the following BI.
- Hysteresis
  If an artificial pace is stimulated after the AVI, the AVI length is prolonged for the next cycle.

We have two constraints for natural paces and two for exactly one pace. This is due to the different requirements of single chamber pacemakers in comparison to dual chamber pacemakers. Furthermore, the constraint *One pace x$\overline{0}$x* is instantiated twice for the verification of the D00 pacemaker.

Table 1 presents the verification results. In the columns we list the properties, the pacemaker variants are arranged in the rows. An ”–” indicates that a property doesn’t apply

	1 pace x00	**max 1 pace $x\bar{0}x$**	**min 1 pace $x\bar{0}x$**	**Atrial pace $D\bar{0}x$**	**Refrac. periods**	**No VES**	**VES**	**Hysteresis**
A00	0s	-	-	-	-	-	-	-
V00	0s	-	-	-	-	-	-	-
D00	0s	-	-	-	-	-	-	-
AAI	-	0 s	0 s	-	5379 s	-	-	-
VVI	-	0 s	1 s	-	4931 s	-	-	-
DDD	-	0 s	1s *	0s*	242 s*	0 s*	204 s*	0 s

Table 1: Verification runtimes of the data flow models, * means time intervals divided by 10

on this pacemaker variant. The times indicate the time obtained for a positive proof from SCADE Design Verifier on executed on a Intel Core 2 Duo P9700 2,80 GHz.

Most properties could be verified nearly instantly. Only the requirements about the refractory periods needed about one and a half hours. To prove for some of the properties of the DDD pacemaker we rescaled the timing constants by a factor 1:10, because only then we were able achieve verification results in reasonable time. When interpreting the current results we have to compare them to our previous work on the verification of monolithic data-flow-oriented pacemaker models [HB11]. For the previous models we derived the pacemaker variants more informally from each other. But we could *not* produce any verification results for most properties in a reasonable time by just using SCADE Design Verifier. Only when applying time abstraction by transferring the verification problem to UPPAAL, we could prove them correct. So, design preferences imposed by the product line modeling lead to a better applicability of built-in heuristics to speed up verification. This observation is backed by a manual inspection of the product models: The effort for manually applying abstraction heuristics like cone-of-influence abstraction or symmetry reduction is decreased significantly compared to the models we investigated in [HB11].

6 Conclusion

We have applied CVL as a product line approach to systematically develop a family of safety critical embedded systems, namely pacemakers. We employed the SCADE suite for domain modeling and code generation for the inner control logic, as SCADE is qualified for safety-critical software development due the most relevant safety standards. It turned out that the concepts for variability modeling strengthen a strict design discipline, e.g. decoupling and small interfaces are privileged. As a consequence we observed that efficiency of model checking for the safety requirements is improved significantly - compared to the previous, informally derived pacemaker variants from [HB11]. In short, the concepts introduced for the product line modeling pay off well for verification.

The next steps are to integrate SysML as a bridge to SCADE into the CVL framework and further investigations on the interrelation between formally founded product line development and verification.

References

[ADS+04] Abdulla, Deneux, Stålmarck, Ågren, and Åkerlund. Designing Safe, Reliable Systems Using Scade. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA*, volume 4313 of *LNCS*, pages 115–129. Springer, 2004.

[BDF+02] Bernstein, Daubert, Fletcher, Hayes, Lüderitz, Reynolds, Schoenfeld, and Sutton. The Revised NASPE/BPEG Generic Code for Antibradycardia, Adaptive-Rate, and Multisite Pacing. *Journal of Pacing and Clinical Electrophysiology*, 25:260 – 264, 2002.

[Com10] Intern. Electrotechnical Commission. *IEC 61508-3:2010: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 3: Software requirements*, 2010.

[cvl10] D2.1.4 - Consolidated CVL language and tool. http://www.omgwiki.org/variability/lib/exe/fetch.php?id=cvl_tool_from_sintef&cache=cache&media=d2.1.4_-_cvl_consolidated_sintef_v1.0.pdf, 2010. June 2, 2010.

[DHM11] Daskaya, Huhn, and Milius. Formal Safety Analysis in Industrial Practice. In Gwenn Salaün and Bernhard Schätz, editors, *16th Intern. Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, volume 6959 of *LNCS*. Springer, 2011.

[Est09] Esterel Technologies. SCADE Suite KCG 6.1: Safety Case Report of KCG 6.1.2, July 2009.

[GOR07] Güdemann, Ortmeier, and Reif. Using deductive cause-consequence analysis (DCCA) with SCADE. In *Proc. 26th Intern. Conference on Computer Safety, Reliability and Security (SAFECOMP)*, volume 4680 of *LNCS*, pages 465–478. Springer, 2007.

[HB11] Huhn and Bessling. Towards Certifiable Software for Medical Devices: The Pacemaker Case Study Revisited. In *5th International Workshop on Harnessing Theories for Tool Support in Software*, pages 8–14, 2011.

[HMPO+08] Haugen, Møller-Pedersen, Oldevik, Olsen, and Svendsen. Adding Standardized Variability to Domain Specific Languages. In *Proceedings of the 2008 12th International Software Product Line Conference*, pages 139–148. IEEE Computer Society, 2008.

[IEC06] International Electrotechnical Commission. *IEC 60812: Analysis Techniques for System Reliability*, 2006.

[Int06] International Electrotechnical Commission. *IEC62304: Medical device software - Software life-cycle processes*, 2006.

[JLS10] Jee, Lee, and Sokolsky. Assurance Cases in Model-Driven Development of the Pacemaker Software. In Tiziana Margaria and Bernhard Steffen, editors, *4th Intern. Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, volume 6416 of *LNCS*, pages 343–356. Springer, 2010.

[LDL07] Liu, Dehlinger, and Lutz. Safety Analysis of software product lines using state-based modeling. *The Journal of Systems and Software*, 80:1879–1892, 2007.

[ORS06] Ortmeier, Reif, and Schellhorn. Deductive Cause Consequence Analysis (DCCA). In *Proc. IFAC World Congress*, Amsterdam, 2006. Elsevier.

[Sci07] Boston Scientific. *PACEMAKER System Specification*, January 2007.

[TZT10] Tuan, Zheng, and Tho. Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker. In *4th Conf. on Secure Software Integration and Reliability Improvement*, pages 23–32. IEEE, 2010.