

Towards the Trustworthy Development of Active Medical Devices: A Hemodialysis Case Study

Atif Mashkoor and Miklos Biro

Abstract—The use of embedded software is advancing in modern medical devices, so does its capabilities and complexity. This paradigm shift brings many challenges such as an increased rate of medical device failures due to software faults. In this letter, we present a rigorous “correct by construction” approach for the trustworthy development of hemodialysis machines, a sub-class of active medical devices. We show how informal requirements of hemodialysis machines are modeled and analyzed through a rigorous process and suggest a generalization to a larger class of active medical devices.

Index Terms—Formal methods, requirements modeling, verification and validation, active medical devices, hemodialysis machines

I. INTRODUCTION

An Active Medical Device (AMD) is a health-care device whose operation depends on a source of electrical energy or any source of power other than that directly generated by the human body or gravity and which acts by converting this energy [1]. Until recently, AMDs were mostly composed of mechanical components. However, recently embedded software has shown to have a determining impact on the consumer value of AMDs and their competitive differentiation. Consequently, according to the latest directive 2007/47/EC of the EU concerning medical devices [2], a stand-alone software can also be considered as an AMD. The main reason of this change is that software lends itself to adaptation to individual requirements and requirements changes clearly much faster than hardware.

As AMDs become more and more software-intensive, due to the immaterial nature of software, their certification becomes a crucial issue. Certification is the process to determine the fitness of a device for public use. Despite the stringent mechanisms already in place to check the quality of medical products with respect to their safe operation, several incidents have been reported that were caused by software faults. Sandler et al. [3] show that more than a fourth of the recalls of defective medical devices during the first half of 2010 were likely caused by software problems. Furthermore, because of the increased deployment of powerful programmable processors in medical devices, software-related recalls are on the rise [4].

Certification regimes have responded to these issues by proposing various medical software-related international stan-

dards and guidelines such as IEC 62304 [5] and FDA General Principles of Software Validation [6] or more general software-related standards such as IEC 61508-3 [7]. The standard IEC 62304 categories medical software into three classes. Class A software cause no injury or damage to health. Class B software cause no serious injury to health. Finally, class C software may cause serious injury or even death. Additionally, the standard also describes the software documents which are required to be produced for each class. Likewise, FDA also proposes several guidelines for medical software development. Some of the key documents of both IEC standards and FDA guidelines are: a requirements specification, a detailed architecture and design document, and an elaborated plan for early and rigorous verification and validation (V&V) supporting all phases of software development life cycle. Although a device manufacturer has some flexibility in choosing V&V principles, the manufacturer retains the ultimate responsibility for demonstrating that the software has been proven correct.

One of the key recommendations of these standards and guidelines is to adopt formal methods for the development of software-intensive critical systems. The use of formal methods is, in fact, “highly recommended” at higher Safety Integrity Levels (SILs). The safety integrity of a system can be defined as the probability of a safety-related system performing the required safety function under all of the stated conditions within a stated period of time. Highly recommended means that if the mentioned technique or measure is not used, then the rationale behind this choice has to be justified during safety planning and assessment. IEC 61508-3 further states that the confidence that can be placed in the software safety requirements specification, as a basis for safe software, depends on the rigor of techniques by which the desirable properties of the specification have been achieved.

The main contribution of this letter is to show how a rigorous refinement-based approach can be applied to the trustworthy development of a sub-system of Hemodialysis (HD) machines, a sub-class of AMDs. The resulting formal model demonstrates an example of generalization, that how requirements can be rigorously specified and analyzed through a chain of refinements to be represented at various abstraction levels, to a larger class of AMDs. The approach also leads to a software safety requirements specification that guarantees correctness of the addressed aspects of behavior, supports verification of the specification based on systematic analysis, avoids intrinsic specification faults, reduces ambiguities in specification writing, and ultimately generates programming language code. The combined approach of requirements mod-

A. Mashkoor and M. Biro work at Software Competence Center Hagenberg, Hagenberg, Austria (e-mail: firstname.lastname@scch.at). The writing of this article is supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

The final publication is available at IEEE via <https://doi.org/10.1109/LES.2015.2494459>.

eling, analysis and development based on techniques such as refinement, V&V and translation, and tools such as proof checkers, model checkers, animation engines and code generators results in obtaining high-assurance and trustworthy AMDs. According to IEC 62304, HD machines are characterized as class C devices and the applied rigorous technique is particularly suitable for this class of AMDs as most of its components belong to a higher SIL.

Formal methods have been used in the past for the development of various health-care devices such as cardiac-care products [8], [9], [10] and infusion pumps [11], [12]. However, the novelty of this work is that this is the first instance of the application of formal methods for the modeling, analysis and development of a sub-class of AMDs responsible for renal replacement therapy. We believe that our work will inspire the manufacturers of such systems to adopt formal paradigms for the safe and trustworthy development of variants of this domain.

II. METHODOLOGY

The development of embedded software for AMDs is a complex process. The degree of complexity often leads to an artifact that, although requires a great amount of time, resources and attention to develop, yet proving its safe operation is challenging. While guaranteeing the absence of mistakes in a piece of software is not always possible, even the identification of their presence is not an easy task. Traditional quality assurance techniques like code reviews or test case generation are also insufficient in this case due to the critical nature of the medical domain. Additionally, the lack of domain knowledge of software engineers makes the matter worse.

We present an approach where a system is synthesized using an incremental refinement process synchronizing and integrating different views and abstraction levels of the system. The process of quality assurance is embedded in the model development. Requirements of the system are supplied to a refinement-based development process that rigorously checks them for consistency and conformance. Every time a new requirement is specified, it first undergoes an internal consistency check and then, additionally, it is also confirmed with the stakeholders whether this requirement indeed captures the desired behavior. The stakeholders, in this way, become part of the development process right from the start and also the chance of an error to trickle down to the later stages of the development process is minimized.

As shown in Figure 1, our approach for the development of high-assurance AMDs consists of four major steps:

- 1) formal requirements specification,
- 2) verification,
- 3) validation, and
- 4) code generation.

In the requirements specification step, informal user and system requirements are translated into a formal specification using a rigorous method. During this process, requirements are precisely written using mathematical and logical structures

which are amenable to formal analysis to determine their correctness.

One of the important cornerstones of the specification process is the representation of requirements at various abstraction levels using the notion of refinement. By following this technique, requirements are easy to specify, analyze and implement. In this style of specification writing, requirements are incrementally added to the model until the model is detailed enough to be effectively implemented.

Once the informal requirements have been translated into a formal specification, the next step is to make sure that the requirements conform to verification standards, i.e., requirements are consistent and verifiable. During this process, it is determined that a specification conforms to some precisely expressed properties that the model is intended to fulfill such as well-definedness, invariant preservation and other safety conditions.

According to [13], two well-established formal verification approaches are theorem proving and model checking. While the former refers to reasoning about defined properties using a rigorous mathematical approach, the latter is the process of exploration of the whole state space of a model to verify dynamic properties. Both deductive theorem proving and model checking are important for proving the consistency of an AMD. While theorem proving is helpful in ensuring safety constraints of the system, model checking is effective in verifying temporal constraints of the system such as liveness and fairness properties.

Once a requirement is specified and verified, the next step to consider is its validation. Validation is a process where it is established by examination and provision of objective evidence that the stakeholders' requirements have been captured correctly and completely in the requirements specification document. Verification alone is not sufficient to guarantee correctness of the model because it does not check whether the specification documents the requirements from the viewpoints of stakeholders.

In order to make stakeholders understand the formal specification, we propose to animate the specification. Animation is a process to demonstrate the fundamental operations of a specification using a dynamic and interactive graphical display. This technique is very well-suited for making a quick mental image of the model even for non-technical domain experts.

The last step of the formal development process is the translation of the requirements specification into programmable code. The last refinement step of the specification writing process is, in fact, already very detailed and close to the implementation stage. An automatic code generation utility capable of translating a formal specification into the target language code such as [14] or [15] can be used at this step.

III. CASE STUDY: HEMODIALYSIS MACHINES

For the experimental validation of our approach, we apply our approach to a HD machine case study [16]. The approach is applied by employing the formal method Event-B [17] and its support platform Rodin [18]. A typical specification

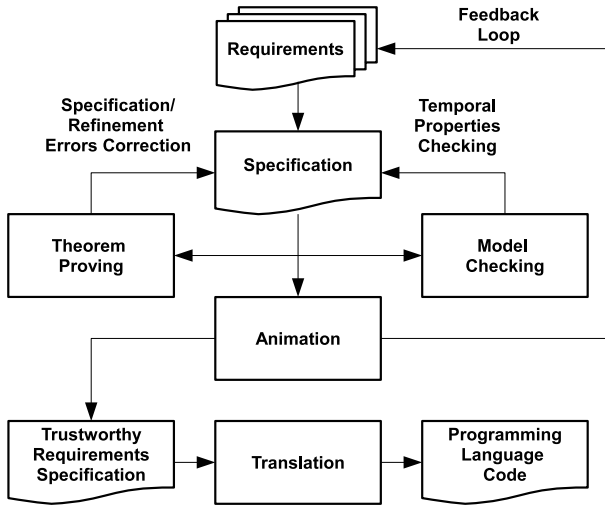


Fig. 1. The formal development paradigm

in Event-B consists of two parts: states and events. A state is a mapping between names and values constrained by an invariant. An event is responsible for transitions between one state and another. For practical purposes, Event-B models are split into Contexts and Machines, each describing the constant and the variable part of the state respectively.

A. Hemodialysis process

HD is a treatment for kidney failure that uses a machine to send the patient's blood through a filter, called a dialyzer, for extracorporeal removal of waste products. The blood is taken through the arterial access of the patient's body. The blood then travels through a tube that takes it to the dialyzer. Inside the dialyzer, the blood flows through thin fibers that filter out wastes and extra fluid using dialysate, a chemical substance that is used in HD to draw fluids and toxins out and to supply electrolytes and other chemicals to the bloodstream. The cleaned blood is then recycled back to the patient through the venous access. A vascular access lets large amounts of blood flow continuously during HD treatments to filter as much blood as possible per treatment. A specific amount of blood is conducted through the machine every minute. The working principle of HD machines is depicted by Figure 2.

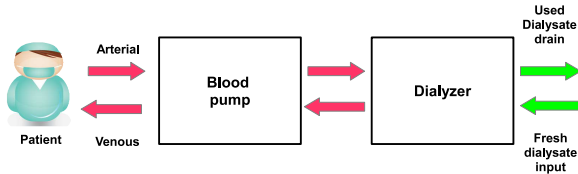


Fig. 2. Working principle of HD machines

B. Model development

1) *Specification step*: The model has been developed using the following pattern:

- 1) Synchronize each requirement with a refinement step,
- 2) Distinguish and specify the static and dynamic elements of the requirement in the context and the machine of the related refinement respectively,
- 3) Introduce the safety properties expressed in the requirement as machine invariants,
- 4) Introduce the monitoring events.

The following is an example of how the requirements of the HD machine case study are specified in the Event-B model.

Requirement: If the system is in the preparation mode and performs priming or rinsing or if the system is in the therapy mode and if the dialysate temperature exceeds the maximum temperature of 41°C, then the software shall disconnect the dialyser from the dialysate within 60 seconds and execute an alarm signal.

We first initiate a context that defines requirement-related static data such as modes, operations, and alarms. Then the corresponding machine of the refinement is specified. It contains several variables and invariants. The following invariants *inv1* and *inv2* specify the related safety requirements.

```

inv1 softwareMode = Preparation  $\wedge$  (operation = Priming  $\vee$  operation = Rinsing)
 $\wedge$  dialysateTemperature > 41  $\Rightarrow$ 
dialyserState = {Dialysate  $\mapsto$  DialyserDisconnected}  $\wedge$ 
dialyserDisconnectionTime < 60  $\wedge$  alarm = ALM377
inv2 softwareMode = Therapy  $\wedge$  dialysateTemperature > 41  $\Rightarrow$ 
dialyserState = {Dialysate  $\mapsto$  DialyserDisconnected}  $\wedge$ 
dialyserDisconnectionTime < 60  $\wedge$  alarm = ALM639

```

Then we specify two monitoring events to capture the behavior of the system because both events trigger different alarms. As shown in Figure 3, the event *disconnectDialyserPreparation* is triggered when the software is in the preparation mode and the temperature of the dialysate rises to more than 41°C during the operation. However, if the software is in the therapy mode while the same thing happens, then the other monitoring event is triggered.

```

Event disconnectDialyserPreparation
Where
softwareMode = Preparation  $\wedge$  dialysateTemperature > 41  $\wedge$ 
(operation = Priming  $\vee$  operation = Rinsing)  $\wedge$ 
dialyserState = {Dialysate  $\mapsto$  DialyserConnected}  $\wedge$ 
dialyserDisconnectionTime < 60
Then
dialyserState := {Dialysate  $\mapsto$  DialyserDisconnected}  $\wedge$ 
alarm := ALM377  $\wedge$  dialyserDisconnectionTime := 0
End

```

Fig. 3. Event *disconnectDialyserPreparation*

An additional event *dialyserDisconnectionClock* is also specified to monitor the timing constraint of the requirement. The tick of the clock is modeled as \mathbb{N} that increments *dialyserDisconnectionTime* by 1 (second).

2) *Verification step*: Verification of a model is achieved when it is proven that the model is free from specification errors and inconsistencies. Our fully-proven specification ensures that the model is consistent, well-defined and its events preserve its invariants. Additionally, we also prove that concrete events in later refinement steps maintain invariants of

the abstract refinements, maintain abstraction invariants, and, when appropriate, decrease variants monotonically.

Temporal properties (safety and liveness) of the system are checked by a combination of theorem proving, model checking and animation. Using theorem proving, we proved that safety invariants are preserved by the behavior of the system, the system is non-divergent, i.e., new events do not take control forever by preventing events from the abstract models from happening, and the system preserves enabledness, i.e., if an event is enabled at an abstract level then it is also enabled at the concrete level. Using model checking, we ensured that legal states of the model are reachable, specified formulas are satisfiable and the model does not contain any deadlock. Using animation, we successfully checked that system traces eventually reach their intended final states.

3) *Validation step*: Validation of a model is achieved when it is demonstrated that the model is free from requirements errors and reflects the stakeholders' wishes adequately. The most common way to validate a specification in the Event-B method is to animate the specification by invoking its operational semantics to inspect its behavior.

For animation and model checking of our specification, we have used the ProB tool [19] that supports automated consistency checking of Event-B machines via constraint solving techniques. Animation using ProB worked very well. We created several behavioral scenarios and executed them accordingly to demonstrate the behavior of the system to stakeholders. The ProB tool assisted us in finding potential invariant problems and their improvement by generating counterexamples whenever an invariant violation is discovered. ProB also helped us in improving invariant expressions by providing hints for strengthening invariants each time an invariant was modified or a new proof was generated by the Rodin platform. We corrected more errors during specification modeling and reviewing than during discharging proofs and animation.

4) *Code generation step*: At the last refinement step, when the specification is sufficiently detailed, we extract the programming language code out of it. This is the result of the translation process that converts the B code into a sequential programming language code that runs on the given hardware. Our target language is C. We use the EB2All tool [15] for the translation purpose.

IV. DISCUSSION, CONCLUSION AND FUTURE WORK

This letter focuses on the rigorous development of a software-controlled safety-critical AMD responsible for renal replacement therapy. Our employed approach successfully enabled us to specify and analyze various critical components of HD machines at different abstraction levels. Our approach also enabled us to detect and correct errors and omissions close to the point of their introduction. The formal Event-B method supported by the Eclipse-based open source Rodin tool has also lent itself to the development of such systems. We have found Event-B an adequate method for the modeling and analysis of critical medical devices. Its refinement principles

and V&V mechanisms provide all the elements that are necessary for the safe development of AMDs and are in full accordance with IEC standards and FDA guidelines.

However, during the development, we also faced several challenges. For example, sophisticated tools and elaborated guidelines for managing the complexity of growing models by decomposition are missing in Event-B and Rodin. There is also no implicit notion of time in Event-B, although this will be necessary for an elegant expression of timing properties which play a critical role in medical devices. Currently, we resort to ProB for proving various temporal properties of the system, but ProB often fails (at the detailed level of refinements) due to the state space enumeration and explosion problems. In our opinion, a standard and more natural way is required to specify and prove that temporal properties of the system are preserved by Event-B refinements. Finally, a tool that is able to generate *ready-to-deploy* machine code from Event-B formal models is also missing; currently available tools can not do this. The notable problems with these tools is that the Event-B model must be restricted to a well-defined subset in order to generate C code and a formal proof that the translation process preserves the safety properties of the model is missing. Extensions of available code generation tools in these directions is an issue for future work.

REFERENCES

- [1] EU, "Council Directive 93/42/EEC," Official Journal of the European Union, June 1993.
- [2] —, "Directive 2007/47/EC of the European Parliament and of the Council," Official Journal of the European Union, September 2007.
- [3] K. Sandler, L. Ohrstrom, L. Moy, and R. McVay, "Killed by code: Software transparency in implantable medical devices," *Software Freedom Law Center*, pp. 308–319, 2010.
- [4] D. R. Wallace and D. R. Kuhn, "Failure modes in medical device software: an analysis of 15 years of recall data," *International Journal of Reliability, Quality and Safety Engineering*, vol. 8, no. 04, pp. 351–371, 2001.
- [5] IEC 62304:2006, "Medical device software - Software life cycle processes," Geneva, Switzerland, 2006.
- [6] Food and Drug Administration (FDA), "General Principles of Software Validation; Final Guidance for Industry and FDA Staff," 2002.
- [7] IEC 61508-3 Ed 2.0, "Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements," Geneva, Switzerland, 2010.
- [8] C. Li, A. Raghunathan, and N. Jha, "Improving the trustworthiness of medical device software with formal verification methods," *Embedded Systems Letters, IEEE*, vol. 5, no. 3, pp. 50–53, Sept 2013.
- [9] D. Méry and N. K. Singh, "Formal specification of medical systems by proof-based refinement," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1, pp. 15:1–15:25, Jan. 2013.
- [10] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam, "From verification to implementation: A model translation tool and a pacemaker case study," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, April 2012, pp. 173–184.
- [11] J. Bowen and S. Reeves, "Modelling safety properties of interactive medical systems," in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS'13. New York, NY, USA: ACM, 2013, pp. 91–100.
- [12] P. Masci, A. Ayoub, P. Curzon, I. Lee, O. Sokolsky, and H. Thimbleby, "Model-Based Development of the Generic PCA Infusion Pump User Interface Prototype in PVS," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, F. Bitsch, J. Guiochet, and M. Kaniche, Eds. Springer Berlin Heidelberg, 2013, vol. 8153, pp. 228–240.

- [13] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Comput. Surv.*, vol. 28, no. 4, pp. 626–643, 1996.
- [14] S. Wright, "Automatic Generation of C from Event-B," in *Workshop on Integration of Model-based Formal Methods and Tools*, 2009.
- [15] N. Singh, "EB2ALL: An Automatic Code Generation Tool," in *Using Event-B for Critical Device Software Systems*. Springer London, 2013, pp. 105–141.
- [16] A. Mashkoor, "The hemodialysis machine case study," Software Competence Center Hagenberg GmbH, Tech. Rep., 2015.
- [17] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
- [18] J.-R. Abrial, M. Butler, S. Hallerstede, T. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, 2010.
- [19] M. Leuschel and M. Butler, "ProB: An Automated Analysis Toolset for the B Method," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 2, pp. 185–203, 2008.