CrossMark

Model-driven development of high-assurance active medical devices

Atif Mashkoor[1]

Abstract   Advanced medical devices exploit the advantages of embedded software whose development is subject to compliance with stringent requirements of standardization and certification regimes due to the critical nature of such systems. This paper presents initial results and lessons learned from an ongoing project focusing on the development of a formal model of a subsystem of a software-controlled safety-critical active medical device (AMD) responsible for renal replacement therapy. The use of formal approaches for the development of AMDs is highly recommended by standards and regulations, and motivates the recent advancement of the state of the art of related methods and tools including Event-B and Rodin applied in this paper. It is expected that the presented model development approach and the specification of a high-confidence medical system will contribute to the still sparse experience base available at the disposal of the scientific and practitioner community of formal methods and software engineering.

Keywords   Model-driven development · Formal methods · Event-B · Active medical devices · Hemodialysis

1 Introduction

With aging and the prevalence of noncommunicable diseases such as diabetes and hypertension, the risk of chronic kidney failure is also increasing. At the terminal phase of chronic kidney failure, renal replacement therapy is required for treatment. One of the possible forms of this therapy is hemodialysis, also known as "artificial kidney" treatment. It is a process in which using a pump system, the patient's blood is flowed through a special filter (dialyzer) which filters out the accumulated waste to be removed together with

✉ Atif Mashkoor
    atif.mashkoor@scch.at

[1]   Software Competence Center Hagenberg GmbH, Hagenberg, Austira

the washer fluid at the other side of the dialyzer. The machine responsible for this therapy is a classical example of an active medical device (AMD).

The council directive 93/42/EEC of the European Union (EU) concerning medical devices (EU 1993) classifies any medical device as an AMD 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. Earlier AMDs were mostly based on hardware solutions. However, lately 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 (EU 2007), 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 change clearly much faster than hardware.

As AMDs become more and more software dependent, due to the immaterial nature of software, their certification becomes a crucial issue. Certification regimes have responded to this issue by proposing various related international standards such as FDA QSR, ISO 13485 (2003), IEC 60601-1 (2005) and IEC 62304 (2006). However, instead of containing actual recommendations for techniques, tools and methods for medical software development, these standards often encourage the use of more general standards and guidelines such as IEC 61508-3 (2010) and FDA General Principles of Software Validation (Food and Drug Administration 2002) as a source for the selection of the appropriate software methods, techniques and tools.

One of the key recommendations of almost all of the standards is to adopt formal methods for the development of software-intensive critical systems. Their use 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 the techniques by which the desirable properties of the specification have been achieved.

The overall aim of this article is to evaluate the possibility of application of a refinement-based formal approach for the improved industrial development of software-controlled safety-critical AMDs. In this case, improvement refers to increased reliability of medical systems. Specifically, we try to answer the following two research questions: (1) Is the refinement-based formal approach suitable for modeling and analyzing all important elements of a complex AMD? and (2) What are advantages and challenges associated with such an approach?

A formal model-driven development approach lets users build systems and software that are correct by construction. The combined approach of requirements modeling and analysis based on techniques such as refinement, verification and validation, and tools such as proof checkers, model checkers and animation engines results in obtaining high-assurance and trustworthy AMDs. The employed notions of formal verification and model validation are in full accordance with the related standards such as IEC 61508-3 and IEC 62304.

To evaluate the applicability of the formal refinement-based approach, we apply it to a hemodialysis machine (Mashkoor et al. 2015), that is an AMD responsible for renal replacement therapy, using the state-based formal method Event-B (Abrial 2010) and its support platform Rodin (Abrial et al. 2010). The application is presented in this paper

following the approach of conducting case study research in software engineering advocated by Runeson and Höst (2009).

The main contribution of this work is the obtained formal model that demonstrates an example of the way in which the requirements of software of modern AMDs can be rigorously specified through a chain of refinements to represent the requirements at different abstraction levels. Additionally, the paper 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 and reduces ambiguities in the specification writing process by involving customers in earlier stages of the development. It is a well-documented fact (Boehm and Papaccio 1988) that the sooner an error is discovered, the lesser it costs.

The fact that the emerging role of embedded software increases the quality/cost ratio of AMDs is obviously advantageous for all the producers of such equipment. It is, however, of especially high importance for small and medium enterprises (SMEs), which traditionally play a significant role in the medical device industry and which are bound to be in the forefront of innovation with their own products, as well as suppliers of larger companies. Innovation always involves higher uncertainties. However, the responsibility of SMEs in reducing the risk of the products to cause any harm is as high as that of any other company. The rigorous approach described in this paper is consequently of essential importance for SMEs in particular.

The European Network and Information Security Agency (ENISA) ad hoc working group on risk assessment and risk management report (2006) already encourages SMEs to use formal paradigms for systems with medium or high criticality for businesses. The survey performed by Woodcock et al. (2009) that examined industrial application of formal methods in 62 projects in the span of 25 years also yielded several interesting results, particularly for SMEs. For example, three times as many participants reported a reduction in development time as reported an increase, five times as many projects reported reductions in costs as reported an increase, and 92 % of projects reported enhanced quality compared to other techniques. The improvement was attributed to better fault detection (36 %), improved design (12 %), confidence in correctness (10 %) and better understanding (10 %).

The paper is organized as follows. Section 2 presents the rigorous approach for modeling and analysis of high-confidence medical systems. Section 3 gives an overview of the selected case study. Section 4 discusses the case study design including objectives, the research instrument and the model development strategy. Section 5 first presents how the refinement-based approach has been used to model various components of a hemodialysis machine and then provides a brief account of the analysis of the presented formal model. Section 6 presents the lessons learned during the model development activity. Section 7 presents some related work. The paper is concluded in Sect. 8.

2 Refinement-based model-driven development approach

The development of embedded software for AMDs is a complex process. The degree of complexity often leads to an artifact that requires a great amount of time, resources and attention to develop. However, proving its safe operation is a challenging task. While guaranteeing the absence of mistakes in a piece of software is not always possible (Dijkstra 1972), even the identification of their presence is not an easy task. Traditional quality

assurance techniques like code reviews or test case generation are also not helpful 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 (Bjørner 2010).

We have proposed 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. Every time a requirement is specified, it goes to an internal consistency check. Once it is ensured that the requirement is specified in the right way, it is also confirmed with the stakeholders whether it 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 in the later stages of the development is minimized.

As shown in Fig. 1, our approach for the development of high-assurance AMDs consists of three majors steps:

1. formal requirements specification,
2. their verification, and
3. their validation.

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. Using this technique, requirements are easy to specify, analyze and implement. In this style of specification writing, requirements are added to the model in a gradual manner. Ultimately, we have a requirements model that 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 they conform to the verification standards, i.e., requirements
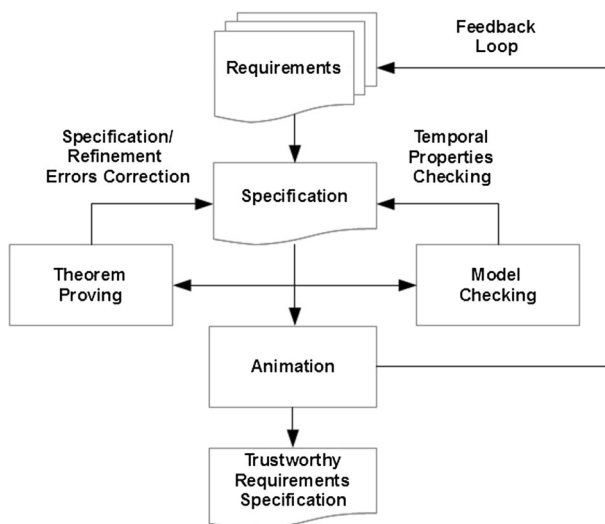

Fig. 1 The triptych approach of formal development of AMDs

are consistent and verifiable. During this process, it is verified that a specification conforms to some precisely expressed properties that the model is intended to fulfill such as well-definedness, invariant preservation and guard strengthening in a refinement using standard verification techniques.

According to Clarke and Wing (1996), two well-established formal verification approaches are theorem proving and model checking. While the former refers to the reasoning of 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. It is a process where it is established by examination and provision of an objective evidence that the stakeholders' requirements have been captured correctly and completely in the requirements specification document. Verification alone is not sufficient to guarantee the correctness of the model because it does not check whether the specification documents the requirements useful for stakeholders.

In order to make stakeholders understand the formal specification, we animate it. 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. It is similar to rapid prototyping; however during animation, a specification is executed without being translated into code.

3 Case description

A hemodialysis machine is used when kidneys do not perform their functions properly. i.e., removal of waste products from blood. It pumps blood from the patient's body through the arteries to the dialyzer that functions as an artificial kidney or a filter. Inside the dialyzer, metabolic waste products are separated from the blood. The dialyzer operates as a filter that is divided into two parts by a semipermeable membrane. On one side, the patient's blood is flowing and on the other side, the dialyzate.

The dialyzate, a chemical substance that is used in hemodialysis to draw fluids and toxins out of the bloodstream and to supply electrolytes and other chemicals to the bloodstream, is prepared by the hemodialysis machine for the therapy. It consists of prepared water that contains certain quantities of electrolyte and bicarbonate, depending on the individual patient's requirements. The concentrations of electrolyte and bicarbonate in the dialyzate are adjusted in such a way that certain substances can be removed from the blood through convection, diffusion and osmosis, while other substances are added at the same time. This is achieved mainly by diffusive clearance through the semipermeable membrane of the dialyzer. The dialyzate transports the metabolic waste products from the dialyzer into the discharge line. The cleaned blood is then recycled back to the patient through the venous access. The working principle of the hemodialysis machine is depicted in Fig. 2.

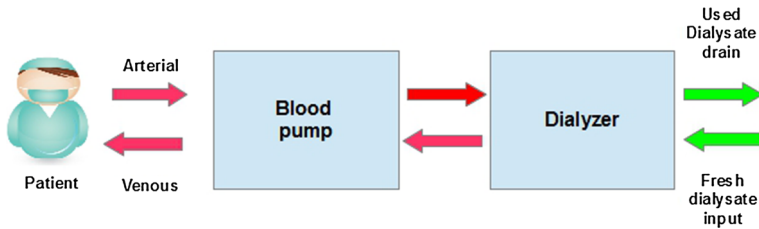The detailed description of the case study is available in Mashkoor (2015).

Fig. 2 Working principle of hemodialysis machines

4 Case study design

4.1 Objectives

The objective of this study was to evaluate the applicability of refinement-based processes for improved embedded software development of active medical devices. We want to determine whether this approach results in an increased reliability of a medical device (demonstrable by proof) while having an effective grasp on the notion of verification and validation throughout its development life cycle. With the employed rigorous model-based approach, the goal is expected to be achieved. Thus, we wanted to investigate advantages and challenges of the proposed approach, and evaluate the effects of implementing it.

The concrete research questions for this study are as follows:

1. Is the formal model-based approach sufficient for modeling all elements of a complex active medical device?
2. What are advantages and challenges associated with this approach?

4.2 Research instrument

We use the formal method Event-B (Abrial 2010) for model development. It is based on Zermelo–Fraenkel set theory with the axiom of choice. It is the successor of the B method (Abrial 1996) for the development of complex reactive systems. We have chosen this method for the formal development of AMDs because of its ability to represent systems at various abstraction levels using its refinement mechanism, easy to use modeling notation and the extensive tool support.

4.2.1 The modeling language

A typical Event-B model is composed of two constructs: machines and contexts. Machines define the dynamic behavior of the model. A typical Event-B machine includes:

-   variables, which define the state space of the machine and can be expressed using natural numbers, integers, real numbers, boolean, sets, relations, functions or any other set-theoretical construct,
-   invariants, which are used either to type variables or to constrain the state space of the machine,
-   variants, which are used to define the convergence property of events, i.e., they can be triggered only for a finite number of times. They are defined using either a natural number or a finite set,

- and events, which describe state transitions. An event is defined as a binary relation composed of guards and actions. A guard is a predicate and all the guards together construct the domain of the corresponding relation. An action is an assignment statement to a state variable and is achieved by a generalized substitution. Combined together, all actions form the range of the corresponding relation. The actions of a particular event are executed simultaneously and non-deterministically.

Contexts define the static elements of a model. They contain carrier sets, constants, axioms and theorems. Carrier sets are used to define types. Axioms are used to constrain carrier sets and constants. Theorems define properties that are derived from axioms.

4.2.2 The refinement process

The Event-B method uses the refinement process to transform an abstract specification into a concrete one. An Event-B model can be refined in several ways:

- new variables and invariants can be introduced and existing invariants can be strengthened,
- existing events can be refined to include and preserve new and existing variables and invariants, respectively,
- existing events can be split into several new events, and,
- completely new events can be introduced to the model.

A machine can be refined into another machine which then contains a more detailed description of the model. A machine can see several contexts, i.e., use the constants and axioms they contain. A context can also be further refined into one or more contexts and can be seen by several machines.

4.2.3 The Rodin toolset

Rodin (Abrial et al. 2010) is the tool that supports modeling and analysis in the Event-B method. Rodin is built upon the Eclipse platform and is extensible by plug-ins such as the model checking and animation plug-in ProB (Leuschel and Butler 2003). The main tasks that are supported by Rodin are:

- specification of machines and contexts,
- their refinement and
- their consistency checking by automatically generating proof obligations (POs).

Proofs can be discharged either automatically, with the help of third-party theorem provers, or interactively. Animation can be achieved by making scenarios and then making sure that events are being fired in the desired order.

4.3 Model development and refinement strategy

During the requirements modeling process, following the advice of Mashkoor and Jacquot (2011) to take small refinement steps, we chose to introduce one requirement per refinement level. Every refinement step introduces a new monolithic requirement of the corresponding component into the model. The static data related to requirements are modeled in contexts, and the general behavior of the system is presented in machines using events. The safety requirements are specified as machine invariants.

In order to improve the legibility of our requirements specification, as originally proposed by Mashkoor and Jacquot (2010), we have classified our axioms into three groups: technical axioms, typing axioms and property axioms. This practice helps in distilling the actual software requirements from technical expressions.

As the Event-B method lacks the explicit notion of time, we have used the timing pattern for Event-B proposed by Cansell et al. (2007). In this technique, $\mathbb{N}$ is used to model the notion of time.

In order to conserve space, the shown refinements contain only the newly introduced information instead of the complete model.

5 Case study report

5.1 Model development

The components of the hemodialysis machine we have chosen to demonstrate as the case study in this paper belong to three different categories:

1. The first one is responsible for connecting the patient to the machine.
2. The second is responsible for monitoring the blood flow at set rates from a patient's arterial access.
3. The last one is responsible for maintaining the temperature of the dialyzate.

5.1.1 The patient connection component

The *connect patient* component is responsible for establishing a connection between a patient and a machine.

Abstract model

The abstract model is comprised of the following requirement:

> The software shall monitor the blood flow in the Extra-corporeal Blood Circuit (EBC) and if no flow is detected, then the software shall stop the blood pump and execute an alarm signal.

We first initiate a context (Context CCP0), as shown in Fig. 3, that contains the static data to specify this requirement. It has two sets: BloodPumpingValues, which models

```
CONTEXT
  CCP0
SETS
  BloodPumpingValues, Alarms
CONSTANTS
  BPStarted, BPStopped, ALM382, Null
AXIOMS
  typ1  partition (BloodPumpingValues, {BPStarted}, {BPStopped})
  typ2  partition (Alarms, {ALM382}, {Null})
END
```

Fig. 3 Context CCP0

the state of the blood pumping process (`BPStarted` or `BPStopped`), and `Alarms`, which contains different types of alarms of the system. The alarm `ALM382` is related to this particular requirement. The constant `Null` defines a state where no alarm has been triggered.

The corresponding machine `MCP0` specifies the aforementioned requirement as shown by `inv4` in Fig. 4, i.e., if no flow is detected, then the software shall stop the blood pump and execute the related alarm. The first three invariants of the machine specify the typing of the variables. The last invariant states that the blood pumping process also implicates the blood flowing process.

In order to capture the behavior of the system, the following events have been introduced to the machine.

Fig. 4 Machine MCP0

```
MACHINE
  MCP0
SEES
  CCP0
VARIABLES
  bloodFlow, alarm, bloodPumping
INVARIANTS
  inv1 bloodFlow ∈ 𝔹
  inv2 alarm ∈ Alarms
  inv3 bloodPumping ∈ bloodPumpingValues
  inv4 bloodPumping = BPStarted ∧ bloodFlow = FALSE ⇒
       bloodPumping = BPStopped ∧ alarm = ALM382
  inv5 bloodPumping = BPStarted ⇒ bloodFlow = TRUE
EVENTS
  Event INITIALISATION
    Then
      act1 bloodFlow := FALSE
      act2 alarm := Null
      act3 bloodPumping := BPStopped
  End
  Event startBloodPumping
    Where
      grd1 bloodPumping = BPStopped
    Then
      act1 bloodPumping := BPStarted
      act2 bloodFlow := TRUE
  End
  Event stopBloodPumping
    Where
      grd1 bloodPumping = BPStarted
    Then
      act1 bloodFlow := FALSE
      act2 bloodPumping := BPStopped
  End
  Event bloodFlowMonitoring
    Where
      grd1 bloodFlow = FALSE
      grd2 bloodPumping = BPStarted
    Then
      act1 bloodPumping := BPStopped
      act2 alarm := ALM382
  End
END
```

- The event INITIALISATION is the default event to initialize the values of newly introduced variables.
- The event startBloodPumping is trivial as it is used to start the blood pumping process.
- The event stopBloodPumping is also trivial as it just checks if the blood pumping process is already started. If so, it stops this process and sets the blood flow state as false.
- The event bloodFlowMonitoring actually specifies the monitoring process of blood flow. If no flow is detected, then the action part of the event stops the blood flow pumping process and triggers the related alarm.

First refinement

The first refinement includes the following requirement into the model:

> The software shall monitor the filling blood volume of the EBC and if the blood volume exceeds 400 ml then the software shall stop the blood pump and execute an alarm signal.

To model this requirement, the context CCP0 is extended to CCP1 which simply introduces the new alarm type, i.e., ALM344, related to this requirement.

The corresponding machine MCP1 at this level specifies the requirement using the following invariant:

$$bloodPumping = BPStarted \wedge bloodFlow = TRUE \wedge$$
$$fillingBloodVolume > 400 \Rightarrow$$
$$bloodPumping = BPStopped \wedge alarm = ALM344$$

A new variable fillingBloodVolume is introduced to keep track of the filling volume along with a new monitoring event fillingBloodVolumeMonitoring that is shown in Fig. 5.

Second refinement

The second refinement includes the following requirement into the model:

> The software shall use a timeout of 310 seconds after the first start of the blood pump. After this timeout the software shall change to the therapy mode.

Fig. 5 Event
fillingBloodVolumeMonitoring

```
Event fillingBloodVolumeMonitoring
  Where
   grd1 fillingBloodVolume > 400
   grd2 bloodPumping = BPStarted
   grd3 bloodFlow = TRUE
  THEN
   act1 bloodPumping := BPStopped
   act2 alarm := ALM344
   act3 bloodFlow := FALSE
  END
```

The context CCP2 extends the context CCP1 by adding a new set SoftwareMode that is comprised of the following modes: Therapy, TherapySelection, Preparation, EndOfTherapy, Disinfection.

The corresponding machine MCP2 at this level specifies the current requirement by the following invariant.

$$bloodPumping = BPStarted \wedge bloodPumpingTime > 310 \wedge$$
$$softwareMode = Preparation \Rightarrow softwareMode = Therapy$$

Two new variables are added to the machine: The variable bloodPumpingTime simulates the tick of the clock for the blood pumping process, and the variable softwareMode represents the current mode of the software.

Two new events are also introduced to model the functionality: Event tick, shown in Fig. 6, acts as a clock that ticks at regular intervals. Event changeMode, shown in Fig. 7, is the event that is responsible for changing the mode of the software from preparation to therapy after 310 s.

Third refinement

The third refinement includes the following requirement into the model:

> The software shall monitor the blood flow direction and if the reverse direction is detected, then the software shall stop the blood pump and execute an alarm signal.

The context CCP3 extends the context CCP2 by adding a new set BloodFlowDirectionValues that are either Forward or Backward. The new related alarm ALM737 is also added to the context.

The corresponding machine MCP3 introduces a new variable bloodFlowDirection to monitor the direction of the blood flow. The requirement is then specified using the following invariant:

Fig. 6 Event tick

```
Event tick
 Where
  grd1 bloodPumping = BPStarted
  grd2 bloodPumpingTime ≤ 310

 Then
  act1 bloodPumpingTime := bloodPumpingTime + 1
End
```

Fig. 7 Event changeMode

```
Event changeMode
 Where
  grd1 bloodPumping = BPStarted
  grd2 bloodPumpingTime > 310
  grd3 softwareMode = Preparation
 Then
  act1 softwareMode := Therapy
End
```

$$bloodPumping = BPStarted \wedge bloodFlowDirection = Backward \Rightarrow$$
$$bloodPumping = BPStopped \wedge alarm = ALM737$$

The newly introduced event `bloodFlowDirectionMonitoring`, shown in Fig. 8, defines the monitoring event that checks the current blood flow direction, and if it is going backward, it immediately stops the blood pumping process and triggers the related alarm.

5.1.2 The blood pumping component

The *blood pumping* component is responsible for blood flow at set rates from the arterial access of patients through the dialyzer to their venous access.

Abstract model

The abstract model of the *blood pumping* component contains the following requirement:

> The software shall monitor the blood flow in the EBC and if no flow is detected for more than 120 seconds, then the software shall stop the blood pump and execute an alarm signal.

Like the previous component, we first initiated a context (`Context CBP0`) that is exactly the same as the context `Context CCP0` shown in Fig. 3; the same alarm is triggered in both cases.

The corresponding machine `MBP0` introduces a new variable `noFlowDetectionTime` that is used to simulate the behavior of a clock related to the blood flow. The requirement is then specified by the following invariant:

$$bloodPumping = BPStarted \wedge noFlowDetectionTime > 120 \Rightarrow$$
$$bloodPumping = BPStopped \wedge alarm = ALM382$$

The monitoring event `noFlowMonitoring`, shown in Fig. 9, specifies the behavior of the model in case that no blood flow is detected for more than 120 s. The blood pumping process is then stopped, the related alarm is triggered, and the related clock is reset.

First refinement

The first refinement introduces the following requirement into the model:

> If the system is not in bypass then the software shall monitor the blood in the EBC and if the actual blood flow is less than 70% of the set blood flow, then the software shall execute an alarm signal.

Fig. 8 Event bloodFlowDirectionMonitoring

```
Event bloodFlowDirectionMonitoring
 Where
  grd1 bloodFlowDirection = Backward
  grd2 bloodPumping = BPStarted
 THEN
  act1 bloodPumping := BPStopped
  act2 alarm := ALM737
 END
```

Fig. 9 Event noFlowMonitoring

```
Event noFlowMonitoring
 Where
  grd1 bloodPumping = BPStarted
  grd2 noFlowDetectionTime ≥ 120
  grd3 bloodFlow = FALSE
 Then
  act1 bloodPumping := BPStopped
  act2 alarm := ALM382
  act3 noFlowDetectionTime := 0
End
```

The context CBP0 is extended into CBP1 which simply introduces the new alarm type ALM755 related to this particular requirement along with a constant SetBloodFlow that sets the desired amount of blood flow in the system.

The corresponding machine MBP1 at this level introduces two new variables into the specification. The variable actualBloodFlow represents the current amount of flowing blood, and the variable bypass represents the bypass state of the system, i.e., the bypass valve is closed or not. The following invariant is introduced for modeling the related requirement:

$$bypass = FALSE \land actualBloodFlow < ((7 * SetBloodFlow) \div 10) \Rightarrow$$
$$alarm = ALM755$$

This refinement also introduces the monitoring event lessBloodFlowMonitoring into the model that is shown in Fig. 10. It makes sure that the actual amount of flowing blood does not fall short of its set amount. If this happens, then it triggers the related alarm.

Second refinement

The second refinement includes the following requirement into the model:

> The software shall monitor the rotation direction of the blood flow pump and if the software detects that the blood flow pump rotates backwards, then the software shall stop the blood flow pump and execute an alarm signal.

This requirement is pretty much same as the requirement which we modeled in the third refinement of the *connect patient* component. Hence, we simply copy the same formalization into the model.

Fig. 10 Event lessBloodFlowMonitoring

```
Event lessBloodFlowMonitoring
 Where
  grd1 bypass = FALSE
  grd2 bloodPumping = BPStarted
  grd3 actualBloodFlow < ((7*SetBloodFlow)/10)
 THEN
  alarm := ALM755
 END
```

5.1.3 The temperature monitoring component

This component is responsible for monitoring the temperature of the dialyzate delivered to the dialyzer. The dialyzate preparation consists in mixing the heated and degassed dialyzate water with other fluid concentrates.

Abstract model

The abstract model of the *temperature monitoring* component contains the following 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 dialyzate temperature exceeds the maximum temperature of 41°C, then the software shall disconnect the dialyzer from the dialyzate and execute an alarm signal.

Like the previous components, we first initiate a context (Context CTM0) that is shown in Fig. 11. The first five axioms of the context are typing axioms. The last one is specified to initialize the starting state of the dialyzer fluids, i.e., all the fluids are disconnected at the time of initialization.

The corresponding machine MTM0 is shown in Fig. 12. It contains several variables whose typing is given by first five invariants. Invariants inv6 and inv7 specify the related requirement. The requirement is split into two invariants because each software mode executes a different alarm if the fluid temperature exceeds the limit of 41 °C.

Correspondingly, we specify two different events to capture the behavior of the system. If software is in the preparation mode and the temperature of the dialyzate rises to more than 41 °C during the operation, then the event disconnectDialyzerPreparation is triggered. If the software is in the therapy mode while the same thing happens, then the event disconnectDialyzerTherapy is triggered.

First refinement

The first refinement of the *temperature monitoring* component contains the following requirement:

```
CONTEXT
 CTM0
SETS
 Operations, DialyzerStates , Fluids , Alarms, SoftwareModes
CONSTANTS
 Priming, Rinsing , DialzserConnected , DialzserDisconnected , ALM377, Default, ALM639, Dialyzate,
 StartingDialysingState , Therapy, TherapySelection , Preparation , EndOfTherapy, Disinfection , Null
AXIOMS
 typ1  partition (Operations, {Priming}, {Rinsing}, {Default})
 typ2  partition ( DialyzerStates , {DialyzerConnected}, {DialyzerDisconnected})
 typ3  partition (Alarms, {Null}, {ALM377}, {ALM639})
 typ4  partition (Fluids , {Dialyzate})
 typ5  partition (SoftwareModes, {Therapy}, {TherapySelection}, {Preparation}, {EndOfTherapy},
       { Disinfection })
 tec1  StartingDialysingState  ∈ Fluids → {DialyzerDisconnected}
END
```

Fig. 11 Context CTM0

```
MACHINE
 MTM0
SEES
 CTM0
VARIABLES
 dialyzerState , dialyzateTemperature , operation , softwareMode, alarm
INVARIANTS
 inv1  dialyzerState  ∈ Fluids  →  DialyzerStates
 inv2  dialyzateTemperature  ∈ N
 inv3  operation  ∈ Operations
 inv4  softwareMode ∈ SoftwareModes
 inv5  alarm  ∈ Alarms
 inv6  softwareMode = Preparation ∧ (operation = Priming ∨ operation = Rinsing) ∧
       dialyzateTemperature > 41 ⇒ dialyzerState = {Dialyzate ↦ DialyzerDisconnected} ∧
       alarm = ALM377
 inv7  softwareMode = Therapy ∧ dialyzateTemperature > 41 ⇒
       dialyzerState  = {Dialyzate ↦ DialyzerDisconnected} ∧ alarm = ALM639
EVENTS
 Event   Initialization
  Then
   act1  alarm := Null
   act2  operation := Default
   act3  dialyzerState  := StartingDialysingState
   act4  dialyzateTemperature := 0
   act5  softwareMode := Preparation
  End
 Event  disconnectDialyzerPreparation
  Where
   grd1  softwareMode = Preparation
   grd2  dialyzateTemperature  > 41
   grd3  operation  = Priming ∨ operation = Rinsing
   grd4  dialyzerState  = {Dialyzate ↦ DialyzerConnected}
  Then
   act1  dialyzerState  := {Dialyzate ↦ DialyzerDisconnected}
   act2  alarm := ALM377
  End
 Event  disconnectDialyzerTherapy
  Where
   grd1  softwareMode = Therapy
   grd2  dialyzateTemperature  > 41
   grd3  dialyzerState  = {Dialyzate ↦ DialyzerConnected}
  Then
   act1  dialyzerState  := {Dialyzate ↦ DialyzerDisconnected}
   act2  alarm := ALM639
  End
END
```

Fig. 12 Machine MTM0

> If the system is in the therapy mode and if the dialyzate temperature falls below the minimum temperature of 33°C, then the software shall disconnect the dialyzer from the dialyzate and execute an alarm signal.

The context CTM0 is extended to CTM1 which simply introduces the new alarm type ALM757 related to this particular requirement.

Fig. 13 Event
disconnectDialyzerTherapyII

```
Event disconnectDialyzerTherapyII
 Where
  grd1 softwareMode = Therapy
  grd2 dialyzateTemperature < 33
  grd3 dialyzerState = {Dialyzate ↦ DialyzerConnected}
 Then
  act1 dialyzerState := {Dialyzate ↦ DialyzerDisconnected}
  act2 alarm := ALM757
End
```

The corresponding machine `MTM1` introduces the following invariant:

$$softwareMode = Therapy \land dialyzateTemperature < 33 \Rightarrow$$
$$dialyzerState = \{Dialyzate \mapsto DialyzerDisconnected\} \land$$
$$alarm = ALM757$$

The model is further strengthened by the introduction of the event `discon-nectDialyzerTherapyII` as shown in Fig. 13. It states that if the dialyzate temperature drops from 33 °C during the therapy mode, then the software should disconnect the dialyzate from the dialyzer and trigger the related alarm.

Second refinement

The second refinement of the *temperature monitoring* component contains the following requirement:

> If the system is in the preparation mode and performs priming or rinsing or if the system is in the therapy mode and the HemoDiaFiltration (HDF) option is available and if the Substitution Fluid (SF) valve is opened and if the dialyzate temperature exceeds 42 °C, then the software shall disconnect the dialyzer from the dialyzate and the EBC from the SF, request bypass and execute an alarm signal.

The context `CTM1` is extended to `CTM2` as shown in Fig. 14. It introduces three sets, i.e., `HDFValues`, `SFValveValues` and `EBCStates`, and various constants. Axioms `typ1` to `typ4` assign the constants to the sets. The last axiom `tec1` defines the initial state of the EBC.

The corresponding machine defines several new variables. The variable `optionHDF` is defined to check the availability of the HDF option[1]. The variable `SFValveValue` determines the opening and closing of the substitution fluid valve. The variable `EBCState` determines which fluid is currently connected to the EBC. The variable `dialyzateBypass` determines whether the bypass option for the dialyzate has been selected or not. The following invariants are also included in the model to state the requirement:

---

[1] Hemodiafiltration (HDF) is a process in which a high rate of ultrafiltration is used for dialysis.

```
CONTEXT
 CTM2
EXTENDS
 CTM1
SETS
 HDFValues, SFValveValues, EBCStates
CONSTANTS
 Online, Offline , Opened, Closed, SubstituitionFluid ,  CircuitConnected,
 CircuitDisconnected , StartingEBCState
AXIOMS
 typ1  partition (HDFValues, {Online}, {Offline})
 typ2  partition (SFValues, {Opened}, {Closed})
 typ3  SubstituitionFluid  ∈ Fluids
 typ4  partition (EBCStates, {CircuitConnected}, {CircuitDisconnected})
 tec1 StartingEBCState ∈ Fluids → {CircuitDisconnected}
End
```

Fig. 14 Context CTM2

$$softwareMode = Preparation \land (operation = Priming \lor operation = Rinsing)$$
$$\land optionHDF = Online \land SFValveValue = Opened \land$$
$$dialyzateTemperature > 42 \Rightarrow$$
$$dialyzerState = \{Dialyzate \mapsto DialyzerDisconnected\} \land$$
$$EBCState = \{SubstituitionFluid \mapsto CircuitDisconnected\} \land$$
$$dialyzateBypass = TRUE \land alarm = ALM377$$
$$softwareMode = Therapy \land optionHDF = Online \land SFValveValue = Opened \land$$
$$dialyzateTemperature > 42 \Rightarrow$$
$$dialyzerState = \{Dialyzate \mapsto DialyzerDisconnected\} \land$$
$$EBCState = \{SubstituitionFluid \mapsto CircuitDisconnected\} \land$$
$$dialyzateBypass = TRUE \land alarm = ALM639$$

Two new events are also added to the model. The event `disconnectDialyz-erTherapyIII`, shown in Fig. 15, demonstrates how and when to disconnect the dialyzer during the therapy mode when the temperature exceeds the specified limit. The event `disconnectDialyzerPreparationII`, shown in Fig. 16, demonstrates the same requirement for the preparation mode.

Third refinement

The third refinement of the *temperature monitoring* component contains the following requirement:

> If the system is in the therapy mode and the HDF option is available and if the SF valve is opened and if the dialyzate temperature falls below 33°C, then the software shall disconnect the dialyzer from the dialyzate and the EBC from the SF, request bypass and execute an alarm signal.

No new context is added to the model as no new static piece of information is introduced by this requirement.

The corresponding machine introduces the following invariant:

Fig. 15 Event
disconnectDialyzerTherapyIII

```
Event  disconnectDialyzerTherapyIII
 Where
  grd1 softwareMode = Therapy
  grd2 optionHDF = Online
  grd3 SFValveValue = Opened
  grd4 dialyzateTemperature > 42

 Then
  act1  dialyzerState  :=
       {Dialyzate ↦ DialyzerDisconnected}
  act2 EBCState :=
       { SubstituitionFluid  ↦ CircuitDisconnected}
  act3  alarm := ALM639
  act4  dialyzateBypass := TRUE
 End
```

Fig. 16 Event
disconnectDialyzerPreparationII

```
Event  disconnectDialyzerPreparationII
 Where
  grd1 softwareMode = Preparation
  grd2 operation = Priming ∨ operation = Rinsing
  grd3 optionHDF = Online
  grd4 SFValveValue = Opened
  grd5 dialyzateTemperature > 42
 Then
  act1  dialyzerState  :=
       {Dialyzate ↦ DialyzerDisconnected}
  act2 EBCState :=
       { SubstituitionFluid  ↦ CircuitDisconnected}
  act3  alarm := ALM377
  act4  dialyzateBypass := TRUE
 End
```

$$softwareMode = Therapy \land optionHDF = Online \land SFValveValue = Opened \land$$
$$dialyzateTemperature < 33 \Rightarrow$$
$$dialyzerState = \{Dialyzate \mapsto DialyzerDisconnected\} \land$$
$$EBCState = \{SubstituitionFluid \mapsto CircuitDisconnected\} \land$$
$$dialyzateBypass = TRUE \land alarm = ALM757$$

To capture the behavior specified by the requirement, a new event `discon-nectDialyzerTherapyIV` as shown in Fig. 17 is added to the model. It specifies how the system should react in the therapy mode when the HDF option is enabled, the SF valve is open and the temperature of the dialyzate drops below the minimum threshold of 33 °C. In this case, both the dialyzate and the SF are disconnected, dialyzate bypass is enabled, and the related alarm is triggered.

5.2 Formal analysis of the model

A model is considered to be formally correct when it is both verified and validated. Verification of a model is achieved when it is proved that it is free from specification errors and inconsistencies. This is usually done either through the system of POs or through model checking. A proved specification ensures that it is consistent, well defined and its

```
Event disconnectDialyzerTherapyIV
 Where
  grd1 softwareMode = Therapy
  grd2 optionHDF = Online
  grd3 SFValveValue = Opened
  grd4 dialyzateTemperature < 33
 Then
  act1  dialyzerState  := {Dialyzate ↦  DialyzerDisconnected}
  act2  EBCState := {SubstituitionFluid ↦ CircuitDisconnected}
  act3  dialyzateBypass := TRUE
  act4  alarm := ALM757
 End
```

Fig. 17 Event disconnectDialyzerTherapyIV

events preserve its invariants. However, proving a refinement requires to prove that concrete events maintain invariants of the abstract model, maintain abstraction invariants and, when appropriate, decrease variants monotonically. Using model checking, we make sure that states of a model are reachable, its formulas are satisfiable, and it does not contain deadlocks.

For our model, Rodin generated three kinds of POs: (1) invariants preservation, (2) well-definedness of guards and invariants and (3) equality of a preserved variable.

Invariants preservation relates to the condition that each variable affected by the assignment statement must preserve the invariant. For example, the event `stopBloodPumping` of machine `MCP0`, shown in Fig. 4, using its guard and both actions ensures that the related invariants `inv2` and `inv5` of the machine are preserved.

The notion of well-definedness relates to the condition which leads to safe evaluation of an expression. For example, the invariant of machine `MCP1` states a condition where the variable `actualBloodFlow` of type $\mathbb{N}$ is compared to an expression of type $\mathbb{R}$. However, as the value assigned to `actualBloodFlow` is always of type $\mathbb{N}$, well-definedness is provable.

Equality of a preserved variable amounts to proving that if a variable is present in both the abstract and the concrete machine and an event of the concrete machine assigns a (new) value to this variable, then it must be proven that this value is consistent with the previous one. For example, the variable `alarm` in all the machines is assigned with a new alarm; however, all the alarms belong to the same type, i.e., `Alarms`.

Table 1 expresses the proof statistics for our formal development using the Rodin platform. These statistics measure the total number of generated POs, automatically discharged POs by the Rodin platform and manually discharged POs. The development of components of the dialysis machine resulted in 106 POs, out of which 105 were discharged automatically. Only one PO required manual interaction (only a few clicks). The employed approach of incremental development helped us to achieve a high degree of automatically discharged POs.

Validation of a model is achieved when it is demonstrated that the model is free from requirements errors and reflects the stakeholders' wishes adequately. This can be done using several techniques, e.g., animation, review or walk-through. The most common way to validate a specification in the Event-B method is to animate the specification by invoking its operation semantics to inspect its behavior. We create behavioral scenarios and execute them. It is then examined whether the specification contains the desired functionality or not.

Table 1 Proof statistics

	Total	Automatic	Interactive
Patient connection	35	35	0
*MCP*0	8	8	0
*MCP*1	9	9	0
*MCP*2	10	10	0
*MCP*3	8	8	0
Blood pumping	25	24	1
*MBP*0	9	9	0
*MBP*1	8	7	1
*MBP*2	8	8	0
Temperature monitoring	46	46	0
*MTM*0	10	10	0
*MTM*1	6	6	0
*MTM*2	19	19	0
*MTM*3	11	11	0
Total	106	105	1

For model checking and animation of our specification, we have used the ProB tool (Leuschel and Butler 2003) that supports automated consistency checking of Event-B machines via constraint solving techniques. Both model checking and animation using ProB worked very well.

The ProB tool assisted us in finding potential invariant problems and their improvement by generating counterexamples whenever it discovered an invariant violation. It also helped us proving the deadlock freedom property of the model. ProB may also help in improving invariant expression by providing hints for strengthening invariants each time an invariant is modified or a new PO is generated by the Rodin platform.

For animation, we created behavioral scenarios and executed them accordingly. We mainly demonstrated that the system is behaving as per expectations, no unintended path of executions is permissible and there is no violation of safety conditions by the specification. The resulting animation was easy to follow, especially for non-technical stakeholders.

As a matter of fact, we corrected more errors during specification modeling and reviewing than during discharging POs and animation.

6 Lessons learned

During the model development exercise, we made certain experiences which are as follows:

- *Formal models provide a consistent and complete repository of requirements*
  The information presented in this paper as case study requirements does not possess a one-to-one mapping from the requirements document to the requirements specification. In fact, the data related to requirements were spread across several documents. For example, the alarm numbers were not explicitly stated in the requirements. Mining the relevant data from these documents is a time-consuming and tricky task. The broken or missing links may sometimes lead to incoherent information that may impact the

correctness of the model. However, one of the advantages of the current modeling exercise is also to provide a repository of adequate and consistent requirements that will positively impact the development of software.

- *Technical details impact the intelligibility of specifications*
  The original purpose of formal specifications is to model and analyze requirements and design decisions in a way that leads to their systematic transformation into correct software. However, during the specification phase, sometimes we need to introduce additional constraints that are necessary to discharge POs but are not part of the original requirements document. Such technical elements impede the understandability of specifications for non-technical stakeholders. The practice of classification of axioms as described in Sect. 4.3 not only increases the intelligibility of a specification but also helps distilling software requirements from technical constraints. The same procedure can be adopted for specifying machines. The guidelines proposed by Kossak et al. (2014) for writing understandable formal specifications by using proper naming conventions and structuring also help rendering specifications intelligible.

- *There is no standard recipe for formal modeling*
  Formal modeling is an overly complex engineering task that cannot be solved by applying some precooked recipe (Su and Abrial 2014). Different formal developments may require different modeling solutions. Traditionally, refinement-based development approaches follow a waterfall-like development structure where requirements are added to the model in a linear sequence. However, this practice does not suit well for large-scale model development (Mashkoor and Jacquot 2015). Therefore, diverging directions for model development are required. We tried multiple modeling approaches for specification development. They are as follows:

  - *Linear sequence development*
    We first tried the traditional linear sequence approach. We started with one initial requirement and then continued enriching the model by introducing one requirement per refinement level. As expected, the model started becoming complex, proofs started becoming complicated and time-consuming, and creating and running animation scenarios started becoming tedious with every refinement step. After six refinement steps, we reached a point where we decided that the model needs to be split. This is due to complex interactions involved between every newly introduced invariant and already existing events.

  - *Decomposable model development*
    The good thing about Event-B is that it provides means to decompose a large model based on either shared variables (A-style) (Abrial and Hallerstede 2007) or shared events (B-style) (Butler 2009). The former approach decomposes a model in such a way that sub-models can contain shared variables. However, the shared variables cannot be refined. For instance, a model $M = \{v1, v2, v3\}$ can be split into $M1 = \{v1, v2\}$ and $M2 = \{v2, v3\}$ where $v2$ is shared. The latter approach decomposes the model in such a way that sub-models have only distinct variables. For instance, a model $M = \{v1, v2, v3\}$ can be split into $M1 = \{v1\}$ and $M2 = \{v2, v3\}$ where no variable is shared.
    When we tried to decompose the dialysis machine model, we found out that the variable `alarm` that is integral to the whole model cannot be present and further refined in every sub-model at the same time. So neither A-style decomposition (although the variable `alarm` would be present in every sub-model but could not

be refined) nor B-style decomposition (the variable `alarm` would become part of only one sub-model) work in our case.

- *Atomic structure development*
  The next solution that we tried was the atomic modeling of components; we modeled each component independently of the other components. This approach works well so far. The modeling is easy, proofs are straight forward, and animation is quick. This is also the formalization that has been presented in this paper as the case study. The problem, however, with this approach is that currently we do not know how the components will interact with each other when they are plugged together. This requires further investigation.

7 Related work

Like other safety-critical systems, medical devices also benefit from formal methods. The use of formal methods has been in place for the development of various healthcare products for a long time. Several decades ago, Hewlett Packard used HP-SL (Bear 1991) to enhance the quality of a range of their cardiac care products (Bowen and Stavridou 1993). The University of Washington used the Z method (Spivey 1988) for the development of a computer control system of a cyclotron and treatment facility that provides particle beams for cancer treatment (Jacky 1990).

In recent years, the use of formal methods is escalating for the development of software-intensive medical systems. For example, Osaiweran et al. (2013) use the formal analytical software design (ASD) (Broadfoot 2005) approach for developing the power control service of an interventional X-ray system. Jiang et al. (2010) present a methodology based on timed automata to extract timing properties of a heart that can be used for the verification and validation of implantable cardiac devices. Tuan et al. (2010) provide a solution for the pacemaker challenge using the model checker process analysis toolkit (PAT) (Sun et al. 2008). Méry and Singh (2013) and Macedo et al. (2008) present a model of pacemakers in Event-B and VDM (Jones 1990), respectively.

One of the medical devices relatively close to hemodialysis machines is an infusion pump. It is primarily responsible for delivering fluids, such as nutrients and medications, into a patient's body in controlled amounts. Arney et al. (2007) present a reference model of patient controlled analgesia (PCA) infusion pumps and test the model for structural and safety properties, Campos and Harrison (2011) present a formal model in modal action logic (MAL) (Campos and Harrison 2008) that helps compare different infusion devices and their provided functionalities, and Bowen and Reeves (2013) use the ProZ model checker (Plagge and Leuschel 2007) to test various safety properties of infusion pumps.

The formal basis for medical software components development we have used in this paper is shared with aforementioned works. However, apart from the work of Méry and Singh (2013), the verification and validation activities are better integrated into our proposed development process as compared to others. We cover a multitude of model analysis activities, e.g., model checking, model review, and animation, that give us a grasp on the notion of correctness far better than approaches which are comprised of only a subset of analysis techniques we have employed. The work of Méry and Singh (2013), though based on Event-B, is still different than our work because they use the refinement chart approach for model development. It is a graphical modeling technique that provides a view of

different subsystems offering assistance in their later integration into a single system. In contrast, our work is based on (different) conventional modeling strategies, i.e., linear sequential, decomposable and atomicity, whose semantics are well defined and whose efficacy has been proven by several industrial success stories such as Behm et al. (1999), Badeau and Amelot (2005) and Iliasov et al. (2013). The system under development is another difference; they work on pacemaker systems, and our work is related to hemodialysis machines.

According to the best of our knowledge, this is the first instance of application of formal methods for the modeling and analysis of AMDs responsible for renal replacement therapy such as dialysis machines. We believe that our specification can act as a reference model that will inspire and facilitate manufacturers of such systems to adopt the formal paradigm for the safe and trustworthy development of variants of this domain.

8 Conclusion and future work

This paper addresses the formal development of safety-critical software components embedded in AMDs. Ethics, as well as the necessity to comply with standards and regulations, make it imperative to follow an approach that helps in analyzing, specifying, implementing and testing such devices.

In this paper, we also report on a case study of model-driven development of a hemodialysis machine, an instance of AMDs. Our conclusion regarding the two research questions defined in Sect. 4 is as follows:

1. Is the formal model-based approach sufficient for modeling all elements of a complex AMD?

   The answer is Yes! Our employed approach successfully enabled us to specify and analyze various critical components of hemodialysis machines at different abstraction levels. 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 verification and validation mechanisms, provide all the elements that are necessary for the safe development of AMDs.

2. What are advantages and challenges associated with this approach?

   The apparent advantage of this approach is that we were able to ensure that errors and omissions in requirements are detected and corrected close to the point of their introduction. We were also successful in integrating non-technical stakeholders in the earlier phases of development cycle by showing them model animations and recording their feedback. The resulting formal model also provided a repository of adequate and consistent requirements that positively impacted the development of software. Last but not least, we found the Event-B notation relatively easy to learn and use.

   However, we also faced several challenges. For example, sophisticated tools and elaborated guidelines for managing the complexity of growing models by decomposition are missing. There is no implicit notion of time in Event-B, that is necessary for an elegant expression of timing properties which play a very important role in medical devices. Currently, we resort to ProB for proving temporal properties of the system. 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 formal models is also

missing. Currently available Event-B supported tools for this purpose are clearly insufficient to produce code that can be deployed on hemodialysis machines without any further human intervention.

We are encouraged to proceed with the further development of components of hemodialysis machines. In future, we also plan to research on the transformation of requirements models into *ready-to-deploy* code artifacts.

References

Abrial, J. R. (1996). *The B book*. Cambridge: Cambridge University Press.

Abrial, J. R. (2010). *Modeling in Event-B: System and software engineering*. Cambridge: Cambridge University Press.

Abrial, J. R., Butler, M., Hallerstede, S., Hoang, T., Mehta, F., & Voisin, L. (2010). Rodin: An open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, *12*(6), 447–466.

Abrial, J. R., & Hallerstede, S. (2007). Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, *77*(1–2), 1–28.

Ad hoc Working Group on Risk Assessment and Risk Management. (2006). *Risk assessment and risk management methods: Information packages for Small and Medium sized Enterprises (SMEs)*. Technical report, ENISA.

Arney, D., Jetley, R., Jones, P., Lee, I., & Sokolsky, O. (2007). Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project. In *Joint Workshop on high confidence medical devices, software, and systems and medical device plug-and-play interoperability, 2007. HCMDSS-MDPnP* (pp. 23–33).

Badeau, F., & Amelot, A. (2005). Using B as a high level programming language in an industrial project: Roissy VAL. In *Formal specification and development in Z and B. Volume 3455 of LNCS* (pp. 334–354). Berlina: Springer.

Bear, S. (1991). An overview of HP-SL. In S. Prehn & W. Toetenel (Eds.), *VDM'91 Formal software development methods. Vol. 551 of Lecture Notes in Computer Science* (pp. 571–587). Berlin: Springer.

Behm, P., Benoit, P., & Meynadier, J. M. (1999). METEOR: A successful application of B in a large project. In *FM. Vol. 1708 of LNCS* (pp. 369–387). Berlin: Springer.

Bjørner, D. (2010). Role of domain engineering in software development—Why current requirements engineering is flawed! In A. Pnueli, I. Virbitskaite, & A. Voronkov (Eds.), *Perspectives of systems informatics. Vol. 5947 of Lecture Notes in Computer Science* (pp. 2–34). Berlin: Springer.

Boehm, B., & Papaccio, P. (1988). Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, *14*(10), 1462–1477.

Bowen, J., & Reeves, S. (2013). Modelling safety properties of interactive medical systems. In *Proceedings of the 5th ACM SIGCHI symposium on engineering interactive computing systems. EICS'13* (pp. 91–100). New York: ACM.

Bowen, J., & Stavridou, V. (1993). Safety-critical systems, formal methods and standards. *Software Engineering Journal*, *8*(4), 189–209.

Broadfoot, G. H. (2005). ASD case notes: Costs and benefits of applying formal methods to industrial control software. In J. Fitzgerald, I. Hayes, & A. Tarlecki (Eds.), *FM 2005: Formal methods. Vol. 3582 of Lecture Notes in Computer Science* (pp. 548–551). Berlin: Springer.

Butler, M. (2009). Decomposition structures for Event-B. In *Proceedings of the 7th international conference on integrated formal methods. IFM'09* (pp. 20–38). Berlin: Springer.

Campos, J. C., & Harrison, M. D. (2008). Systematic analysis of control panel interfaces using formal tools. In T.C. N. Graham and P. Palanque (Eds.), *Interactive systems. Design, specification, and verification, Vol. 5136 of Lecture Notes in Computer Science* (pp. 72–85). Berlin: Springer.

Campos, J. C., & Harrison, M. D. (2011). Modelling and analysing the interactive behaviour of an infusion pump. In *Proceedings of 4th International ECEASST*.

Cansell, D., Mery, D., & Rehm, J. (2007). Time constraint patterns for Event-B development. In J. Julliand & O. Kouchnarenko (Eds.), *7th International conference of B users. Vol. 4355 of LNCS* (pp. 140–154). Berlin: Springer.

Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys, 28*(4), 626–643.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM, 15*(10), 859–866.

EU. (June 1993). Council Directive 93/42/EEC. *Official Journal of the European Union.*

EU. (September 2007). Directive 2007/47/EC of the European Parliament and of the Council. *Official Journal of the European Union.*

Food and Drug Administration (FDA). (2002). General principles of software validation; Final guidance for industry and FDA Staff.

IEC 60601-1:2005. (2005). *Medical electrical equipment—Part 1: General requirements for basic safety and essential performance.*

IEC 61508-3 Ed 2.0. (2010). *Functional safety of electrical/electronic/programmable electronic safety-related systems—Part 3: Software requirements.*

IEC 62304:2006. (2006). *Medical device software—Software life cycle processes.*

Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., et al. (2013). Developing mode-rich satellite software by refinement in Event-B. *Science of Computer Programming, 78*(7), 884–905.

ISO 13485:2003. (2003). *Medical devices—Quality management systems—Requirements for regulatory purposes.*

Jacky, J. (1990). Formal specification for a clinical cyclotron control system. *SIGSOFT Software Engineering Notes, 15*(4), 45–54.

Jiang, Z., Pajic, M., Connolly, A., Dixit, S., & Mangharam, R. (2010). A platform for implantable medical device validation: Demo abstract. In *Wireless Health 2010. WH '10* (pp. 208–209). New York: ACM.

Jones, C. B. (1990). *Systematic software development using VDM* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall Inc.

Kossak, F., Mashkoor, A., Geist, V., & Illibauer, C. (2014). Improving the understandability of formal specifications: An experience report. In C. Salinesi & I. Weerd (Eds.), *Requirements engineering: Foundation for software quality. Vol. 8396 of Lecture Notes in Computer Science* (pp. 184–199). Berlin: Springer.

Leuschel, M., & Butler, M. (2003). ProB: A model checker for B. In K. Araki, S. Gnesi, & D. Mandrioli (Eds.), *FME 2003: Formal methods LNCS 2805* (pp. 855–874). Berlin: Springer.

Macedo, H., Larsen, P., & Fitzgerald, J. (2008). Incremental development of a distributed real-time model of a cardiac pacing system using VDM. In J. Cuellar, T. Maibaum, & K. Sere (Eds.), *FM 2008: Formal methods. Vol. 5014 of Lecture Notes in computer science* (pp. 181–197). Berlin: Springer.

Mashkoor, A. (2015). *The hemodialysis machine case study.* Technical report SCCH-TR-1542. Austria: Software Competence Center Hagenberg GmbH.

Mashkoor, A., & Jacquot, J. P. (2015). Observation-level-driven formal modeling. In *2015 IEEE 16th International Symposium on High-assurance systems engineering (HASE)* (pp. 158–165).

Mashkoor, A., & Jacquot, J. P. (Sept 2010). Domain Engineering with Event-B: Some lessons we learned. In *Requirements engineering conference (RE), 2010 18th IEEE international* (pp. 252–261).

Mashkoor, A., Biro, M., Dolgos, M., & Timar, P. (2015). Refinement-based development of software-controlled safety-critical active medical devices. In *Software Quality. Software and Systems Quality in Distributed and Mobile Environments, Vol. 200 of Lecture Notes in Business Information Processing* (pp. 120–132). Berlin: Springer.

Mashkoor, A., & Jacquot, J. P. (2011). Guidelines for Formal Domain Modeling in Event-B. In *2011 IEEE 13th international symposium on high-assurance systems engineering (HASE)* (pp. 138–145).

Méry, D., & Singh, N. K. (2013). Formal specification of medical systems by proof-based refinement. *ACM Transactions on Embedded Computing Systems, 12*(1), 15:1–15:25.

Osaiweran, A., Schuts, M., Hooman, J., & Wesselius, J. (2013). Incorporating formal techniques into industrial practice: An experience report. *Electronic Notes Theoretical Computer Science, 295*, 49–63.

Plagge, D., & Leuschel, M. (2007). Validating Z specifications using the ProB animator and model checker. In J. Davies & J. Gibbons (Eds.), *Integrated formal methods. Vol. 4591 of Lecture Notes in Computer Science* (pp. 480–500). Berlin: Springer.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), 131–164.

Spivey, J. M. (1988). *Understanding Z: A specification language and its formal semantics.* Cambridge: Cambridge University Press.

Su, W., & Abrial, J. R. (2014). Aircraft landing gear system: Approaches with Event-B to the modeling of an industrial system. In F. Boniol, V. Wiels, Y. Ait Ameur & K. D. Schewe (Eds.), *ABZ 2014: The landing gear case study. Vol. 433 of Communications in computer and information science* (pp. 19–35). Berlin: Springer.

Sun, J., Liu, Y., & Dong, J. (2008). Model checking CSP revisited: Introducing a process analysis toolkit. In T. Margaria & B. Steffen (Eds.), *Leveraging applications of formal methods, verification and validation. Vol. 17 of communications in computer and information science* (pp. 307–322). Berlin: Springer.

Tuan, L. A., Zheng, M. C., & Tho, Q. T. (2010). Modeling and verification of safety critical systems: A case study on pacemaker. In: *2010 Fourth international conference on secure software integration and reliability improvement (SSIRI)* (pp. 23–32).

Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). Formal methods: Practice and experience. *ACM Computing Surveys, 41*(4), 19:1–19:36.

Atif Mashkoor is the Scientific Head of the Rigorous Methods in Software Engineering (RSE) Research Focus at the Software Competence Center Hagenberg (SCCH) GmbH. He has several years of experience in rigorous systems and software engineering ranging from domain modeling to testing of executable code. Previously, he has taught at the University of Lorraine (France), COMSATS Institute of Information Technology (Pakistan) and National University of Modern Languages (Pakistan), and worked as a Research Associate at the University of Minho (Portugal). He holds a doctoral degree from the University of Lorraine (France) and a master degree from the Umeå University (Sweden), both in Computer Science. In addition, he also obtained postgraduate diploma in Computational Linguistics from the Rovira i Virgili University (Spain).