Validating and Verifying the Requirements and Design of a Haemodialysis Machine using the Rodin Toolset

Thai Son Hoang[a,*], Colin Snook[a,*], Asieh Salehi[a], Michael Butler[a], Lukas Ladenberger[b]

[a] *ECS, University of Southampton, Southampton, UK*
[b] *University of Düsseldorf, Düsseldorf, Germany*

Abstract

We present a formal specification and analysis of a haemodialysis machine (HD machine) in Event-B using the Rodin Toolset. The medical device domain is a particularly complex multidisciplinary field involving disparate branches of engineering, biological and medical fields as well as a critical patient-machine interface. Requirements include safety properties, process steps, human-machine interfaces, timing constraints, dynamic control algorithms, and design features. Our aim is to demonstrate that the Event-B based modelling, verification and validation tools deal with the variety of requirements involved in a typical medical device. We utilise ProR for structuring and tracking requirements. We model the HD machine using iUML-B state-machines and class diagrams, and build a corresponding BMotion Studio visualisation. For verification, we use both theorem proving and model checking techniques. We validate the design of the system using (i) diagrams to aid the modelling of the sequential properties of the requirements, and (ii) ProB-based animation and visualisation tools to explore the system's behaviour. Some of the safety properties involve dynamic behaviour which is difficult to verify in Event-B. For these properties we use (iii) co-simulation tools to validate against a continuous model of the physical behaviour. We conclude that the Event-B based modelling tools are particularly rich in verification and validation techniques and with the help of supporting tools for requirements tracking, are able to address the different kinds of requirements in a medical device.

Keywords: Haemodialysis Machine, Event-B, ProR, ProB, iUML-B, BMotion Studio, Co-Simulation.

*Corresponding author
Email addresses: T.S.Hoang@ecs.soton.ac.uk (Thai Son Hoang), cfs@ecs.soton.ac.uk (Colin Snook), asf08r@ecs.soton.ac.uk (Asieh Salehi), mjb@ecs.soton.ac.uk (Michael Butler), ladenberger@cs.uni-dusseldorf.de (Lukas Ladenberger)

1. Introduction

This paper describes our approach to modelling formally the requirements and design of a haemodialysis machine (HD machine) [1]. The HD machine is used for patients with kidney failure to remove waste products from their blood. We identify how we deal with the requirements that are defined for the HD machine [1]. Since the requirements document goes further than functional requirements and describes how an HD machine should consist of various sub-components and how those sub-components interact, we view this as lower-level or design requirements.

We use Event-B [2], a formal method for system development, and structure our model using refinements to deal with complexity. Since the HD machine's requirements involve extensive sequencing and user interactions as well as dynamic interaction with the HD machine, we use diagrams to aid the modelling of the sequential properties of the requirements. Where appropriate we use the proof capabilities of Event-B to verify safety constraints. For temporal properties related to timing constraints, we use ProB model checker [3]. Furthermore, to validate the model, we extensively use ProB-based animation, visualisation and simulation tools to explore the behaviour of our models.

Our contribution is to demonstrate how different kinds of requirements can be traced and verified or validated using the tools available. The medical device domain is a particularly complex multidisciplinary field involving contributions from a large range of engineering branches, highly skilled and expert medical practitioners as well as the sensitive and somewhat unpredictable patient-machine interface. Given this diverse group of stakeholders, we believe a range of validation techniques are essential to facilitate concensus amongst stakeholders that the requirements and design are safe, sensible and useful. For example, biologists and control engineers may confer over dynamic models of the interaction between the HD machine and the patient's cardiovascular system while medical practitioners and domain expert software engineers might utilise state-machine animations and visual simulations to agree on the procedural and operator interface aspects of the device.

The hypothesis of the case study is that the Event-B based modelling, verification and validation tools are sufficient to deal with all types of requirement involved in a typical medical device such as the HD machine. To be more precise, the reason for modelling is to provide re-assurance that the requirements and design are sensible (i.e. well-formed and non-conflicting) and useful (i.e. subjectively desirable based on expert domain knowledge) within the context of the device's purpose. Hence requirements are sufficiently dealt with if they can be modelled in a way that supports verification and validation activities that are appropriate to the requirement's nature, and that these verification and validation activities provide stakeholders with a reasonable mechanism for accepting or rejecting the requirements/design. We also require this process to be reasonable in the sense that it should be tractable, efficient and within the stakeholders expected capabilities albeit with appropriate training where necessary. The requirements include safety properties, process steps, human-machine interfaces, timing constraints, dynamic control algorithms, and design features. We utilise a range of Event-B modelling techniques and tools to test the hypothesis and perform a literature study to assess other methods for comparison.

The criteria for judging the result of the case study are (i) whether all kinds of requirement can be handled by the available Event-B tools, and (ii) an assessment of the appropriateness of the way each kind of requirement is handled. The latter will necessarily be somewhat subjective. For example, a safety property should be rigorously verified whereas a user interface should be validated by animation.

To do this we provide (i) a model of the HD machine using iUML-B [4, 5, 6] state-machines and class diagrams, (ii) a BMotion Studio visualisation for the developed Event-B model, (iii) a co-simulation of the closed-loop parts of the controller with a continuous domain model of the environment. On the one hand, we use (1) the built-in theorem provers of the Rodin platform [7] to discharge the proof obligations related to safety invariants, and (2) ProB model checker to verify temporal (liveness) properties. On the other hand, the graphical model and visualisation enable us to analyse and validate the behaviour of the HD machine. Finally, to track the system requirements to the formal models, we use ProR [8] which allows us to attach elements of the formal models to the requirements.

This paper is built on the work reported in [9]. Compared to the previous work, we explicitly model the timing constraints of the system and make use of the ProB model checker to verify the consistency of the model, in particular, checking temporal liveness properties. As a result, our model covers the requirements more completely compared to [9], where the timing constraints are still abstracted. Furthermore, we have utilised the ProR tool for structuring and tracking the requirements and recording the validation/verification methods used for each requirement.

The rest of the paper is structured as follows. Section 2 gives some background on the methods and tools that we use. The main content of the paper is in Section 3 describing the development of the HD machine, verification and validation of its requirements and design using iUML-B, ProB, BMotion Studio, and co-simulation. At the same time, we also show how we use ProR for requirements tracing. Finally, we summarise and conclude in Section 5.

2. Background

In this section, we first present some background information on the HD machine case study (Section 2.1). In subsequent (sub-)sections, we review ProR for managing requirements (Section 2.2), the Event-B modelling method (Section 2.3), iUML-B graphical front-end for Event-B (Section 2.4), BMotion Studio for visualising Event-B models (Section 2.5), and finally the co-simulation framework that we used (Section 2.6).

2.1. The HDMachine Case Study

The HD machine case study is introduced in [1]. The HD machine is used for patients with kidney failure to remove waste products from their blood. The schematic view of the HD machine can be seen in Figure 1. The scope of our study is the control system which interfaces with the user via the User Interface (UI) and also monitors and controls the functions of the HDMachine. Here we distinguish between the "patient" who receives treatment from the HD machine and the "user" who operates the HD machine.

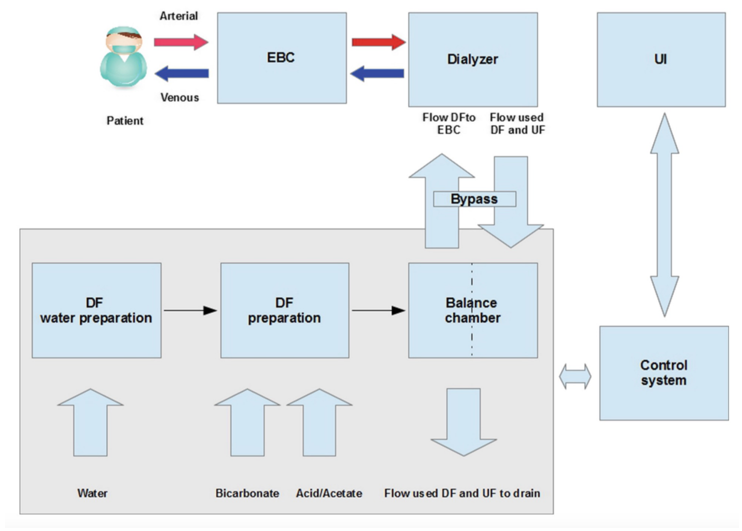


Figure 1: Schematic view of the HD machine [1]

The blood is extracted and returned to the patient through an Extracorpreal Blood Circuit (EBC) using peristaltic pumps. The overview of the EBC can be seen in Figure 2 on the following page.

A haemodialysis therapy session contains 3 main phases: *preparation*, *initiation*, and *ending*. Each phase contains smaller sub-steps. The overview of the main phases is as follows.

1. **Preparation.** The preparation phase allows the user to prepare for the therapy, including testing the control functions, preparing the tubing systems and setting various parameters for the therapy.
2. **Initiation.** The first step of this phase is to connect the patient to the HD machine. When the patient is connected successfully, the therapy starts. During the therapy, the HD machine monitors various conditions such as the blood pressure (using blood monitors) and the blood flow, and stops the machine if some unexpected problem occurs.
3. **Ending.** When therapy finishes, the machine starts the reinfusion process and disconnects the patient from the HD machine. After emptying the dialyzer and cartridge, the machine displays the summary of the therapy.

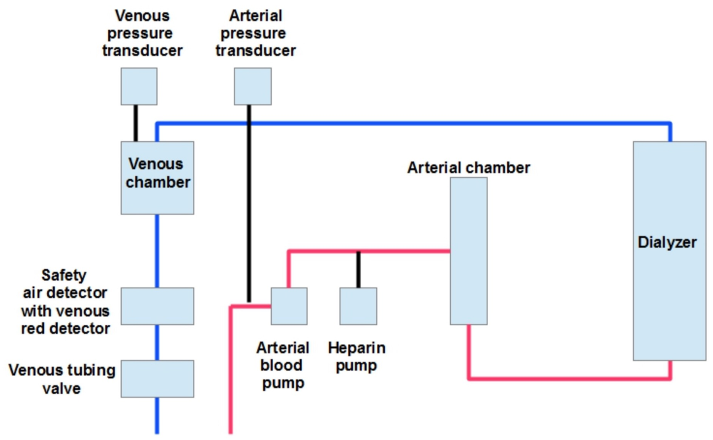

Venous pressure transducer
Arterial pressure transducer
Venous chamber
Arterial chamber
Dialyzer
Safety air detector with venous red detector
Venous tubing valve
Arterial blood pump
Heparin pump

Figure 2: Overview of the EBC of the HD machine [1]

The safety requirements of the HD machine are separated into general requirements (that specify the overall behaviour of the system) and software requirements (more specific requirements about the controller's behaviour for each functional area of the system, e.g., arterial bolus application, controlling blood pump, monitoring blood-side entry pressure, etc.). We will not repeat the full set of requirements here and instead refer the readers to [1], however, throughout the paper, we will show particular requirements relevant to understanding the paper.

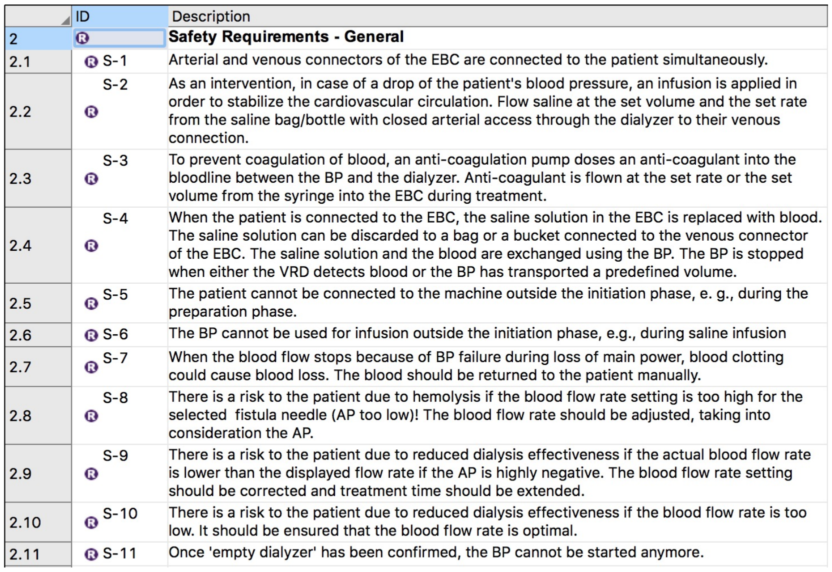
2.2. ProR

ProR is an Eclipse editor and GUI for managing requirements in accordance with the Requirements Interchange Format (ReqIF) standard [10]. ProR uses the Eclipse Requirements Modelling Framework (RMF)[1] which is an Eclipse Modelling Framework (EMF) based implementation of ReqIF. ProR allows users to structure requirements and insert links in order to trace requirements either to more detailed requirements or to other artefacts such as model elements and ultimately implementation. In our case we are interested in linking the HD machine requirements to our iUML-B model elements.

ProR supports extension of the requirements model and editor so that users may configure the tool to describe new attributes of their requirements or new kinds of artefacts involved in the requirements management process. Since ReqIF is a standard data exchange format for requirements, ProR allows requirements to be exchanged with other tools including IBM Rational DOORS[2].

[1]http://www.eclipse.org/rmf/
[2]http://www-03.ibm.com/software/products/en/ratidoor

Figure 3 shows the general safety requirements for the HD machine from [1] entered into ProR.



	ID	Description
2	Ⓡ	**Safety Requirements - General**
2.1	Ⓡ S-1	Arterial and venous connectors of the EBC are connected to the patient simultaneously.
2.2	Ⓡ S-2	As an intervention, in case of a drop of the patient's blood pressure, an infusion is applied in order to stabilize the cardiovascular circulation. Flow saline at the set volume and the set rate from the saline bag/bottle with closed arterial access through the dialyzer to their venous connection.
2.3	Ⓡ S-3	To prevent coagulation of blood, an anti-coagulation pump doses an anti-coagulant into the bloodline between the BP and the dialyzer. Anti-coagulant is flown at the set rate or the set volume from the syringe into the EBC during treatment.
2.4	Ⓡ S-4	When the patient is connected to the EBC, the saline solution in the EBC is replaced with blood. The saline solution can be discarded to a bag or a bucket connected to the venous connector of the EBC. The saline solution and the blood are exchanged using the BP. The BP is stopped when either the VRD detects blood or the BP has transported a predefined volume.
2.5	Ⓡ S-5	The patient cannot be connected to the machine outside the initiation phase, e. g., during the preparation phase.
2.6	Ⓡ S-6	The BP cannot be used for infusion outside the initiation phase, e.g., during saline infusion
2.7	Ⓡ S-7	When the blood flow stops because of BP failure during loss of main power, blood clotting could cause blood loss. The blood should be returned to the patient manually.
2.8	Ⓡ S-8	There is a risk to the patient due to hemolysis if the blood flow rate setting is too high for the selected fistula needle (AP too low)! The blood flow rate should be adjusted, taking into consideration the AP.
2.9	Ⓡ S-9	There is a risk to the patient due to reduced dialysis effectiveness if the actual blood flow rate is lower than the displayed flow rate if the AP is highly negative. The blood flow rate setting should be corrected and treatment time should be extended.
2.10	Ⓡ S-10	There is a risk to the patient due to reduced dialysis effectiveness if the blood flow rate is too low. It should be ensured that the blood flow rate is optimal.
2.11	Ⓡ S-11	Once 'empty dialyzer' has been confirmed, the BP cannot be started anymore.

Figure 3: General safety requirements for the HD machine using ProR

2.3. Event-B

Event-B [2] is a formal method for system development. Main features of Event-B include the use of *refinement* to introduce system details gradually into the formal model. An Event-B model contains two parts: *contexts* and *machines*. Contexts contain *carrier sets*, *constants*, and *axioms* constraining the carrier sets and constants. Machines contain *variables* v, *invariants* $I(v)$ constraining the variables, and *events*. An event comprises a guard denoting its enabled-condition and an action describing how the variables are modified when the event is executed. In general, an event e has the following form, where t are the event parameters, $G(t, v)$ is the guard of the event, and $S(t, v)$ is the action of the event.

$$\mathsf{e} \;\widehat{=}\; \textbf{any } t \textbf{ where } G(t, v) \textbf{ then } S(t, v) \textbf{ end} \tag{1}$$

In the case where the event has no parameters, we use the following form

$$\mathsf{e} \;\widehat{=}\; \textbf{when } G(v) \textbf{ then } S(v) \textbf{ end} \quad , \tag{2}$$

and when the event has no parameters and guard, we use

$$\mathsf{e} \;\widehat{=}\; \textbf{begin } S(v) \textbf{ end} \quad . \tag{3}$$

The action of an event comprises of one or more assignments, each of them has one of the following forms.

$$v := E(t, v) \tag{4}$$

$$v :\in E(t, v) \tag{5}$$

$$v :| \ P(t, v) \tag{6}$$

Assignments of the form (4) are deterministic, assign value of expression $E(t, v)$ to v. Assignments of the forms (5) and (6) are non-deterministic. (5) assigns any value from the set $E(t, v)$ to v, while (6) assigns any value satisified predicate $P(t, v)$ to v. Note that invariants $I(v)$ are inductive, i.e., they must be *maintained* by all events. This is more strict than general safety properties which hold for all reachable states of the Event-B machine. This is also the difference between verifying the consistency of Event-B machines using theorem proving and model checking (e.g., ProB) techniques: model checkers explore all reachable states of the system while interpreting the invariants as safety properties.

A machine in Event-B corresponds to a transition system where *variables* represent the states and *events* specify the transitions. More information about Event-B can be found in [11]. Event-B is supported by the Rodin Platform (Rodin) [7], an extensible toolkit which includes facilities for modelling, verifying the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches.

In this paper, we use several Event-B modelling "patterns" for developing the HD machine model. The patterns are Event-B *model templates* that can be instantiated to the actual system. We present templates as Event-B model fragments, i.e., a set of variables, events, etc. Each pattern addresses some common modelling aspects that occured during the development of the HD machine, e.g., controlling physical equipment (Section 3.2.3) or capturing timing contraints (Section 3.3.1). In particular, the pattern for controlling physical equipment consists of machines at different level of refinement, i.e., a refinement pattern.

2.4. iUML-B

iUML-B provides a diagrammatic modelling notation for Event-B in the form of state-machines and class diagrams. The diagrammatic models are contained within an Event-B machine and generate or contribute to parts of it. For example a state-machine will automatically generate the Event-B data elements (sets, constants, axioms, variables, and invariants) to implement the states while Event-B events are expected to already exist to represent the transitions. Transitions contribute further guards and actions representing their state change, to the events that they elaborate. A choice of two alternative translation encodings are supported by the iUML-B tools. State-machines are typically refined by adding nested state-machines to states. Class diagrams provide a way to visually model data relationships. For the HD machine we use state-machine diagrams extensively to model the sequential processes and exploit both Event-B encodings. We used class diagrams to model environmental interfaces but do not show this here.

Figure 4 shows an example of a state-machine, named **SM**. Here we show a

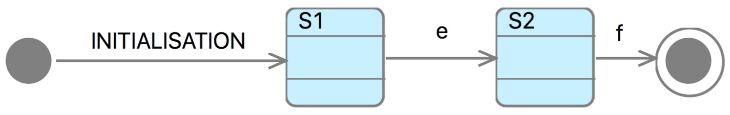


Figure 4: An example iUML-B state-machine **SM**

translation of state-machine **SM** using the *enumeration* encoding, where each state is encoded as a constant from an enumerated set SM_STATES. Variable SM, which represents the current state of the state-machine, is initialised to $S1$. Events e and f change the value of SM according to the transitions in the state-machine. The Event-B translation can be seen below.

sets : SM_STATES

constants : $SM\_NULL, S1, S2$

axioms :
partition$(SM_STATES, SM_NULL, S1, S2)$

variables : SM

invariants :
$SM \in SM\_STATES$

INITIALISATION :
begin
$SM := S1$
end

e :
when
$SM = S1$
then
$SM := S2$
end

f :
when
$SM = S2$
then
$SM := SM\_NULL$
end

2.5. BMotion Studio

In this paper we have used the new version[3] of BMotion Studio [13] to create a *domain specific visualisation* (DSV) of our Event-B model of the HD machine. BMotion Studio comes with a graphical environment including a visual editor that provides various *graphical elements* to create a visualisation of the model. A graphical element is based on Scalable Vector Graphics (SVG) and HTML, two markup languages which support widgets like shapes, images, labels, tables and lists. Moreover, *observers* are used to link the model with the visualisation. For instance, the tool provides a *formula observer* that binds a formula (e.g. an expression or a variable) to a graphical element and allows the tool to compute

[3]Originally BMotion Studio was developed as a separate plug-in for Rodin [12].

a visualisation for any given state by changing the properties of the graphical element (e.g. the colour or position) according to the evaluation of the formula in the respective state. Finally, *event handlers* can be attached to the visualisation to provide interactive functionalities, such as an *execute event handler* that binds an Event-B event to a graphical element and executes the event when the user clicks on the graphical element.

2.6. Co-Simulation

The Rodin tools and plug-ins are aimed at modelling discrete state-changing events; they are not so good at validating continuous behaviour. Despite this we often need to model the requirements for a system that periodically controls some continuous dynamic behaviour of the environment. Modelling tools such as Dymola [14] are available and are able to simulate dynamic properties and their discrete controllers but they do not provide the refinement and verification techniques of Event-B. In order to validate such models a *MultiSim* plug-in [15] was developed by Savicks et al. The plugin allows an Event-B model and a continuous model to be simulated synchronously. Typically the Event-B part will model a cyclic control system that monitors process variables from the continuous model and calculates a controlled output variable. The Event-B model is animated using ProB and the continuous model is a Functional Mock-up Unit (FMU) [16] which has been exported from a continuous domain modelling tool such as Dymola. The exported FMU is a program that can be run independently of the source modelling tool in order to provide a simulation of the continuous domain model. The plug-in controls the coordination and communication between the co-operating simulations repeating the following sequence:

- animate the Event-B until a designated event is reached where the control outputs are available,
- set these control output values as inputs in the FMU ,
- run the FMU simulation for a fixed period of time,
- read the FMU output values,
- set these output values in some designated monitoring variables of the Event-B.

3. Development

In this section, we give some highlights of our formal development of the HD machine. The requirements and design of the HD machine are given in [1] and we will not repeat those requirements in this paper. We suggest that readers study this section together with the requirements document [1] and the formal model available from the web site `http://eprints.soton.ac.uk/401360/`.

We first give an overview of the development strategy that we have applied for this formal model in Section 3.1. Subsequently, we highlight the key important modelling decisions using iUML-B (Section 3.2). In Section 3.3, we show

how we model timing constraints and the use of ProB for verifying temporal liveness properties of our model. We illustrate how we use BMotion Studio to validate our model (Section 3.4). For dynamic properties that cannot be expressed in Event-B, we show how co-simulation helps us to validate such properties (Section 3.5). Finally, in Section 3.6, we present the links between the requirements and our formal model using ProR.

The work-flow involved in our formal development process may be summarised as follows.

- The requirements are arranged and structured in ProR. This allows us to allocate the appropriate modelling, verification and validation methods to each requirement as well as structure the requirements into categories and levels.
- We choose an initial abstraction of the system that represents the highest level requirements as an overview of what the system achieves and then make new versions of the model in refinements to incorporate further details and requirements from ProR.
- For each refinement step
  - We model this refinement step using iUML-B class diagrams and state-machines.
  - We verify and validate the requirements incorporated in this level of the model according to the selected methods noted in ProR.
    1. When a requirement involves continuous domain behaviour we co-simulate the discrete Event-B control model with a continuous domain model of the controlled phenomenon.
    2. When a requirement specifies a safety property we use theorem provers to verify that the property is upheld.
    3. When a requirement specifies a sequential process we use animation to validate that the process is modelled correctly.
    4. When a requirement specifies timing properties we use the model checker to verify LTL temporal properties.
  - For traceability, we record in the ProR system, a reference to the model element that represents each ProR requirement.
- To validate the completed model we create an animated visualisation of the model using BMotion Studio.

3.1. Development Strategy

The haemodialysis process is highly sequential with several sub-processes. In the design described in [1], the HD machine's control system contains two parts: a top-level and a low-level control system, working independently. The top-level control system manages the communication with the users, and transmits data from/to other modules. The low-level control system manages the

HD machine while interacting with the top-level control system. Our formal model reflects this design of the control system: the top-level one manages the overall haemodialysis process, interacting with the users, while the low-level one controls the sub-processes by monitoring and regulating the behaviour of the HD machine. The patient interacts with the machine in two ways.

1. Discrete actions of the patient and operator in response to outputs of the machine. This is modelled by a sequence of transitions, one to output the instruction from the machine and the next to respond to the completion of the patient's action.
2. Once the machine is connected and running, the patient's blood pressure is affected by the blood pump. This interaction with the patient is modelled via the low level control system's blood pump controller responding to the dynamic behaviour of the patient's blood pressure. The patient is modelled using a continuous domain modelling tool which is co-simulated with the control model.

We omitted requirements related to Ultra Filtration (UF), the Safety Air Detector (SAD), the temperature of the HD machine, and loss of main power. These can easily be incorporated via refinements using similar modelling techniques.

Refinement strategy in Event-B is often influenced by the correctness proofs. It is sometimes necessary to choose small refinement steps to make it easier to discharge proof obligations. However, in this case, there are no difficult proofs and this is not a driving factor for the choice of refinement strategy. Despite this, we still choose to introduce details in incremental refinement steps so that the model is easier to understand and communicate. For example, the abstract levels could be animated to explore the top level processes before going on to animate levels that incorporate more details about setting parameters and making patient connections. Hence our refinement strategy follows the abstraction levels of the sequential steps of the system. This strategy also fits the nested state-machine architecture in iUML-B.

m00: Models the main phases of the haemodialysis process for the top-level control system, i.e., Preparation, Initiation, and Ending

m01: Models the *sub-processes* within each main phases for the top-level control system.

m02: Models the user's interaction with the HD machine to turn the machine on and off.

m03: Models the low-level control's *automatic testing of control functions.*

m04: Models the actual (physical) result of testing the HD machine's control functions.

m05: Model the *message passing communication* between the low-level control system and the HD machine for testing control functions.

m06: Models the set of *signals*.

m07: Models the signal for indication of control function testing result.

m08: Models the connection of concentrate to the HD machine.

m09: Models the *setting of rinsing parameters*.

m10: Models the sequence of connecting patient

m11: Models the physical connection of the patient (arterially and venously).

m12: Models the three pressure monitors and the system normal/abnormal states.

m13: Models various abnormal blood-side pressures.

m14: Models the blood pump, actual blood flow and abnormal situations when monitoring the blood flow.

m15: Models arterial bolus.

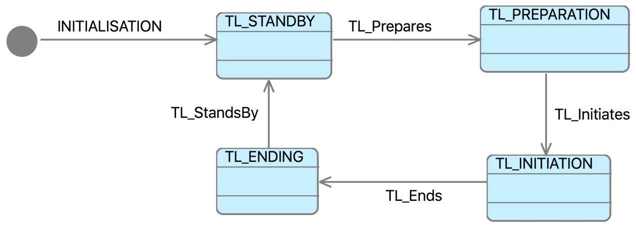
m16: Models heparin bolus.

3.2. Modelling using iUML-B

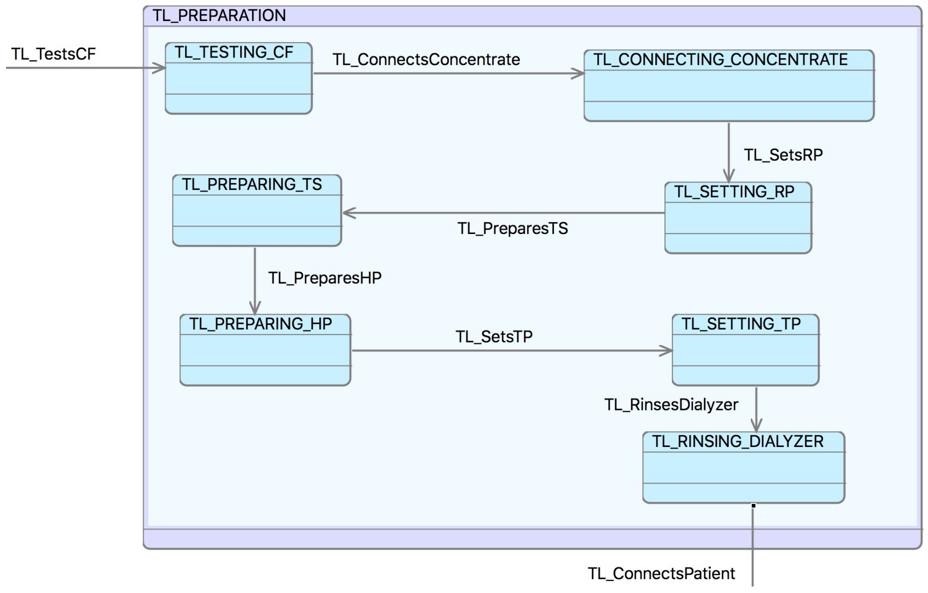
3.2.1. Modelling Sequential Processes

The haemodialysis process contains three main phases: *preparation*, *initiation*, *ending*. Each main phase is composed of several sequential steps. Using iUML-B state-machines, it is straight-forward to model such sequential processes/sub-processes. Furthermore, the notion of nested state-machines (which can be naturally introduced via refinement) fits perfectly for refining the processes further into smaller sequential steps. Figure 5 on the next page illustrates how we model the sequential processes in **m00** and **m01**. Figure 5a shows the main phases of the haemodialysis process (with the additional *TL_STANDBY* state). In **m01**, we introduce nested state-machines for states *TL_PREPARATION*, *TL_INITIATION*, *TL_ENDING* to model the sequential sub-steps of each main phase. Figure 5b gives an example of the nested state-machine for *TL_PREPARATION*.

The incoming/outgoing transitions of the super-state *TL_PREPARATION* in **m00**, i.e., TL_Prepares and TL_Initiates, are respectively refined to TL_TestsCF and TL_ConnectsPatient in **m01**. Using the encoding where each iUML-B state is represented by a constant from an enumerated carrier set, TL_Prepares and TL_TestsCF are straightforwardly translated into Event-B as follows. Here, variable *TopLevel* indicates the current state of the top-level state-machine, and the current state of the nested state-machine in state *TL_PREPARATION* is represented by variable *TL_PREPARATION_sm*.



(a) State-machine *TopLevel* in **m00**



(b) Nested state-machine for state *TL_PREPARATION* in **m01**

Figure 5: Modelling sequential processes with state-machines

```
TL_Prepares :
when
  TopLevel = TL_STANDBY
then
  TopLevel := TL_PREPARATION
end
```

```
TL_TestsCF :
when
 TopLevel = TL_STANDBY
then
 TopLevel := TL_PREPARATION
 TL_PREPARATION_sm := TL_TESTING_CF
end
```

3.2.2. Top-level vs. Low-level Control Systems

The top-level control system, which maintains the sequential haemodialysis process and its sub-steps, is modelled in a single state-machine. The low-level, responsible for direct control of the HD machine to perform certain tasks, is modelled using several state-machines each corresponding to a particular task. Figure 6 illustrates the state-machine *LowLevel_TestingCF* for the low-level control system performing testing of Control Functions (CF) in **m03**.

Transitions LL_TestsCF and LowLevel_StandsBy of the *LowLevel_TestingCF* state-machine are guarded accordingly, to ensure that they can only be carried out in the correct phases as specified in the top-level control state-machine *TopLevel*.

```
LL_TestsCF :
when
 ...
 TL_PREPARATION_sm = TL_TESTING_CF
then
 ...
end
```

```
LL_TestsCF :
when
 ...
 TopLevel = TL_STANDBY
then
 ...
end
```

The top-level control system can only move from state *TL_TESTING_CF* to the next state, i.e., *TL_CONNECTING_CONCENTRATE*, when the CF testing is successful. Using the Event-B encoding for state-machine *LowLevel_TestingCF*

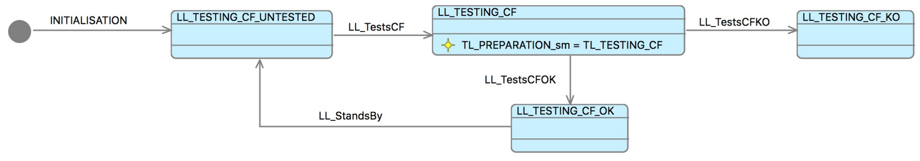


Figure 6: Low-level control system for CF testing in **m03**

where each state is represented by a constant, this is modelled by a guard on the transition TL_ConnectsConcentrate in state-machine *TopLevel* (Figure 5b on page 13) stating that

$$LowLevel\_TestingCF = LL\_TESTING\_CF\_OK\ .$$

3.2.3. A Pattern for Controlling Physical Equipment

A common pattern that we used in modelling the HD machine is to formalise how the controller interacts with the physical equipment in the environment. The pattern involves two refinement levels. At the abstract level, the controller and the physical equipments can have access to the states of the other components. In the refinement, this direct access is refined by message passing communication. This pattern is similar to the action/reaction patterns in [2, Chapter 3] and we extended with refinement and applied them to iUML-B state-machines. We show below how we incorporate the physical testing of CF into the formal model.

Recall the low-level control system for CF testing in Figure 6. In **m04**, a variable *HDM_TestingCFOK* is introduced to denote the result of the HD machine's CF test and a new event HDM_TestsCF is allowed to set this variable non-deterministically representing the result of the test. The guard of the event ensures that the physical tests are carried out only when the low-level controller is in the testing state, i.e., *LL_TESTING_CF*.

```
HDM_TestsCF :
when
  $LowLevel\_TestingCF = LL\_TESTING\_CF$
then
  $HDM\_TestingCFOK :\in \mathrm{BOOL}$
end
```

Transitions LL_TestsCFOK and LL_TestsCFKO of state-machine *LowLevel_TestingCF* are directed by the actual result of the test: they are guarded to select a pass/fail response according to *HDM_TestingCFOK*.

```
LL_TestsCFOK :
when
  ...
  $HDM\_TestingCFOK = \mathrm{TRUE}$
then
  ...
end
```

```
LL_TestsCFKO :
when
  ...
  $HDM\_TestingCFOK = \mathrm{FALSE}$
then
  ...
end
```

In the next refinement **m05**, we introduce the communication between the controller and the HD machine. Two new variables *LL_2_HDM_TestsCF* and *HDM_2_LL_TestsCFFinished* are introduced to model the message exchange. Flag *LL_2_HDM_TestsCF* is set in event LL_TestsCF and unset in HDM_TestsCF. Invariant

$$LL\_2\_HDM\_TestsCF = \mathrm{TRUE} \Rightarrow \mathsf{LL\_TestsCF} = \mathrm{TRUE}$$

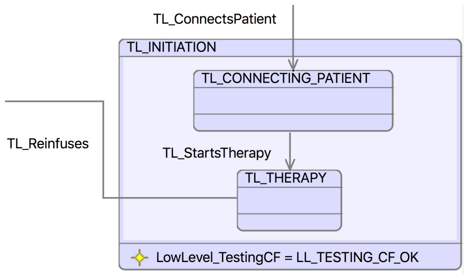


Figure 7: Invariant on the *TL_INITIATION* state of state-machine *TopLevel* in **m03**

allows us to refine HDM_TestsCF's guard to

$$LL\_2\_HDM\_TestsCF = \mathrm{TRUE}\ .$$

Similarly, *HDM_2_LL_TestsCFFinished* is set in HDM_TestsCF and unset in LL_TestsCFOK and LL_TestsCFKO. At the same time, the guards of the two events are strengthened by adding the condition stating that the actual CF testing has been finished, i.e.,

$$HDM\_2\_LL\_TestsCFFinished = \mathrm{TRUE}\ .$$

3.2.4. Safety Properties as State-machine Invariants

An important feature of iUML-B is state-invariants. They can be used to express safety properties that must hold in a certain state. A state-invariant is translated into an Event-B machine invariant by having an additional condition stating that the state-machine is in the corresponding state. For example, assuming that we use the enumeration encoding for state-machine **SM** (i.e., each state is encoded as a constant from an enumerated set), we have a variable *SM* represents the current state of the state-machine. A state-invariant $P(v)$ of the state *S* of **SM** is then translated into Event-B as follows:

$$SM = S \ \Rightarrow\ P(v)\ .$$

Consider the state-machine *TopLevel* in **m03**. We wish to ensure that when the system is in the *TL_INITIATION* phase, the CF should have been successfully tested. We add an invariant

$$LowLevel\_TestingCF = LL\_TestsCFOK$$

to state *TL_INITIATION* as shown in Figure 7. The translation of the state-invariant in Event-B is, as expected, i.e.,

$$TopLevel = TL\_INITIATION \Rightarrow LowLevel\_TestingCF = LL\_TestsCFOK\ .$$

To prove the above invariant, an invariant is added to the *TL_PREPARATION* state stating that if the system pass the *TL_TESTING_CF* state, then the CF have been tested successfully, i.e.,

$$TL\_PREPARATION\_sm \neq TL\_TESTING\_CF \Rightarrow \\ LowLevel\_TestingCF = LL\_TestsCFOK\ .$$

3.2.5. Animation/Model Checking to Validate Requirements

Consider **m11**, we introduce a state-machine to model the physical connections/disconnections of the patient to the HD machine arterially and venously (Figure 8). The patient is connected to the machine in the first step of the *TL_INITIATION* phase and disconnected from the machine in the first step of the *TL_ENDING* phase. Requirements **S-1** and **S-5** from [1] are directly related to the connections status of the patient and are as follows.

S-1 Arterial and venous connectors of the EBC are connected to the patient simultaneously.

S-5 The patient cannot be connected to the machine outside the initiation phase, e. g., during the preparation phase.

Initially, we model **S-1** as an invariant

$$\begin{aligned} &\mathit{USR\_ConnectingPatient} \neq \mathit{USR\_DISCONNECTED} \Rightarrow \\ &\quad \mathit{USR\_ConnectingPatient} = \mathit{USR\_CONNECTED\_BOTH} \ . \end{aligned}$$

and **S-5** as state-invariants for states *TL_PREPARATION* and *TL_ENDING* specifying that

$$\mathit{USR\_ConnectingPatient} = \mathit{USR\_DISCONNECTED} \ .$$

Attempts to prove these invariants lead to failure. We use the ProB model checker to find counter-example traces and iUML-B state-machine animation to visualise the obtained traces. The visualisation helps us to identify the cause of the problems and how to fix them. In this case, the requirements are clearly too strong and contradict other requirements. On the one hand, during the preparation phase, i.e., when *TopLevel* is *TL_PREPARATION*, the patient is connected first arterially and then venously contradicting **S-1**. On the other hand, the patient is still connected both arterially and venously when the re-infusion step starts, i.e., outside the initiation phase contradicting **S-5**. We therefore weaken the requirements **S-1** and **S-5** as follows.

S-1' Arterial and venous connectors of the EBC are *both* connected to the patient *during therapy*.

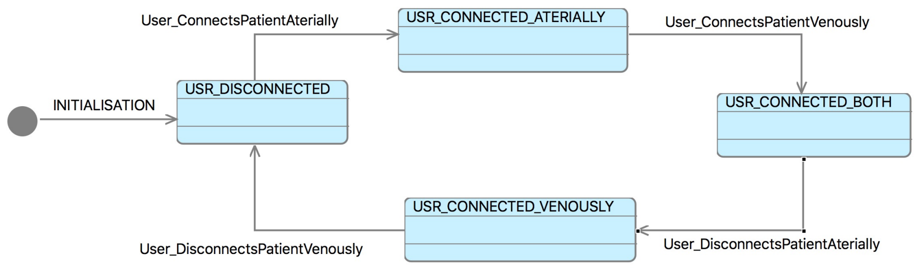
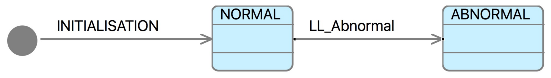

User_ConnectsPatientAterially
USR_CONNECTED_ATERIALLY
User_ConnectsPatientVenously
INITIALISATION
USR_DISCONNECTED
USR_CONNECTED_BOTH
USR_CONNECTED_VENOUSLY
User_DisconnectsPatientVenously
User_DisconnectsPatientAterially

Figure 8: Patient connections to the HD machine

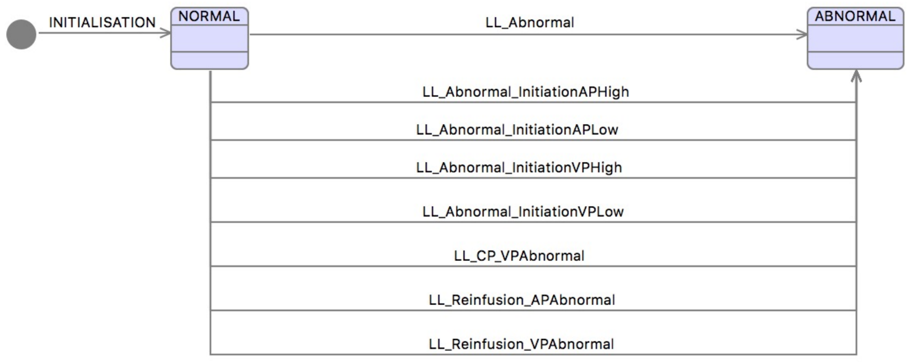
S-5' The patient cannot be connected to the machine outside the *initiation and reinfusion phases.*

3.2.6. Modelling Abnormal Behaviours

During haemodialysis treatment, an important part of the HD machine is to monitor the various aspects of the patient and the machine, and raise the alarm when abnormal behaviours are detected. This includes low/high blood pressures, incorrect blood flow directions, etc. We have developed a pattern for modelling such behaviour. An abstract state-machine for the low-level control system is added in **m12** (Figure 9a). When we introduce the pressure monitor in **m13**, various abnormal conditions can be detected. The events modelling such detection are a refinement of the abstract event LL_Abnormal (Figure 9b). Note that we still keep the abstract event LL_Abnormal to be able to detect more abnormal behaviours in future refinements.



(a) State-machine *LowLevel_Status* in **m12**


INITIALISATION
NORMAL
LL_Abnormal
ABNORMAL
LL_Abnormal_InitiationAPHigh
LL_Abnormal_InitiationAPLow
LL_Abnormal_InitiationVPHigh
LL_Abnormal_InitiationVPLow
LL_CP_VPAbnormal
LL_Reinfusion_APAbnormal
LL_Reinfusion_VPAbnormal

(b) State-machine *LowLevel_Status* in **m13**

Figure 9: Modelling Abnormal Behaviours

3.3. Modelling Timing Constraints

A majority of the software requirements for the HD machine are timing constraints. Inspired by [17], we develop patterns for modelling timing constraints such as *deadline*, *expiry*, and *delay*. Instead of using a natural number to represent a global time in the machine, we define a specific"timer" variable for each timing constraint in order to model the relationship between occurrences of events. We first abstractly present our modelling patterns for the different timing problem before applying them to the HD machine case study.

3.3.1. Timing Contraint Modelling Patterns

Our presentation of the timing modelling patterns focuses on a single triggering event Trigger and a single responding event Response. However, the patterns can be extended to multiple triggering/responding events. Moreover, for simplification, we present Trigger and Response without any parameters.

For each pattern, we use a variable *timer* to present the timer. We use special value -1 for the timer to denote that it is deactivated. Otherwise, i.e., when it is active, the timer will be a natural number, representing the number of remaining clock ticks until the timer is expired. A Time event, representing the clock, adjusts the timer accordingly. We assume that Time can advance the clock some arbitrary d ticks (without taking into account the timing constraints), i.e., it is of the following form.

$$\mathsf{Time} \ \widehat{=} \ \textbf{any} \ d \ \textbf{where} \ d \in \mathbb{N}_1 \land \ldots \ \textbf{then} \ \ldots \ \textbf{end}$$

The additional guards (conditions on *timer*) and actions (update to *timer*) of the Time event depending on the actual patterns. Furthermore, all the timing modelling patterns contribute to the same Time event. In the following presentation of the patterns, the omitted guards and actions (for Trigger, Response, and Time) are from the actual system under development or from other timing patterns.

Delay. We want to model a delay by duration D, i.e., after the occurrence of Trigger, the next occurrence of Response (if any) must be *after* D clock ticks. We denote this timing constraint as $Delay(\mathsf{Trigger}, \mathsf{Response}, D)$. The following Event-B pattern is used to model the delay.

```
(Delay)Trigger :
when
  $timer = -1$
  ...
then
  $timer := D$
  ...
end
```

```
(Delay)Response :
when
  $timer \leq 0$
  ...
then
  $timer := -1$
  ...
end
```

```
(Delay)Time :
any $d$ where
  $d \in \mathbb{N}_1$
  ...
then
  $timer := \max(\{-1, timer - d\})$
  ...
end
```

Variable *timer* is set by Trigger, where the guard of Response ensures that it cannot execute where the timer has not yet expired. The Time's action ensures that it decreases the timer by d but cannot go below -1. Note that the *timer* can be reset by both Response and Time.

Expiry. We want to model an expiry duration D, i.e., after the occurrence of Trigger, the next occurrence of Response (if any) must be *before* D clock ticks. We use *Expiry*(Trigger, Response, D) to denote this timing constraint. The following Event-B pattern is used to model the expiry.

```
(Expiry)Trigger :
when
  $timer = -1$
  ...
then
  $timer := D$
  ...
end
```

```
(Expiry)Response :
when
  $timer \geq 0$
  ...
then
  $timer := -1$
  ...
end
```

```
(Expiry)Time :
any $d$ where
  $d \in \mathbb{N}_1$
  ...
then
  $timer := \max(\{-1, timer - d\})$
  ...
end
```

Similar to the previous pattern, $timer$ is set by Trigger. The guard of Response ensures that it cannot execute where the timer has already expired. The action of Time ensure that it decreases the timer by d but cannot go below -1. Here, $timer$ can be reset by both Response and Time.

Deadline. We want to model a deadline of D, i.e., after the occurrence of Trigger, within D of clock ticks, Response must occur. We denote this timing constraint as *Deadline*(Trigger, Response, D). The following Event-B pattern is used to model the deadline.

```
(Deadline)Trigger :
when
  $timer = -1$
  ...
then
  $timer := D$
  ...
end
```

```
(Deadline)Response :
when
  $timer \geq 0$
  ...
then
  $timer := -1$
  ...
end
```

```
(Deadline)Time :
any $d$ where
  $d \in \mathbb{N}_1$
  $(timer \neq -1 \Rightarrow d \leq timer)$
  ...
then
  $timer := \max(\{-1, timer - d\})$
  ...
end
```

Similar to the *Expiry* pattern, *timer* is set by Trigger and the guard of Response ensures that it cannot execute where the timer has already expired. The action of Time ensure that it decreases the timer by *d* but cannot go below -1. The difference with the previous pattern is an additional guard to Time. This ensures that in the case where the timer is currently active, i.e., $timer \neq -1$, the advancement duration *d* is no more than the current *timer* value. As a result, the timer cannot be deactivated by Time: *timer* is reset only when there is an occurrence of Response.

3.3.2. Timing Constraints in the HD machine

As an example for *Deadline* pattern, we consider the user's action of turning on/off the HD machine. Let *maxUserCommDelay* be a constant corresponding to the *maximum communication delay* between the user's action and the response from the HD machine. In other words, the response of the HD machine has a deadline represented by *maxUserCommDelay* after an occurrence of the user's actions. We model the user's action in **m02** using two events User_PressesOn and User_PressesOff. The corresponding events of the HD machine is TL_TestsCF and TL_StandsBy, to start testing control functions and to put the machine in stand-by mode, respectively. Using the notation introduced earlier, the timing constraints between these events can be denoted as

$$Deadline(\mathsf{User\_PressesOn}, \mathsf{TL\_TestsCF}, maxUserCommDelay) \text{ and} \quad (7)$$

$$Deadline(\mathsf{User\_PressesOff}, \mathsf{TL\_StandsBy}, maxUserCommDelay)\ . \quad (8)$$

Applying the *Deadline* pattern to the first timing constraint (7) results in the following events.

```
User_PressesOn :
when
  $TopLevel = TL\_STANDBY$
  $\mathsf{User\_PressesOnTimer} = -1$
then
  $\mathsf{User\_PressesOnTimer} := maxUserCommDelay$
end
```

```
TL_TestsCF :
when
  $TopLevel = TL\_STANDBY$
  $\mathsf{User\_PressesOnTimer} \geq 0$
then
  $\mathsf{User\_PressesOnTimer} := maxUserCommDelay$
end
```

```
Time :
any $d$ where
  $d \in \mathbb{N}_1$
  $(\text{User\_PressesOnTimer} \neq -1 \Rightarrow d \leq \text{User\_PressesOnTimer})$
  ...
then
  $\text{User\_PressesOnTimer} := \max(\{-1, \text{User\_PressesOnTimer} - d\})$
  ...
end
```

Other applications of the timing patterns are related to monitoring the situation during the operation of the HD machine. Consider requirement **R-9** from the requirement document [1] as follows.

R-9 While connecting the patient, if the software detects that the pressure at the VP transducer exceeds +450mmHg for more than 3 seconds, then the software shall stop the BP and execute an alarm signal.

To capture requirement **R-9**, we introduce state-machine *LowLevel_CP_VPStatus* as shown in Figure 10. Focus on the transitions LL_CP_VPHigh, LL_CP_VPNormal,

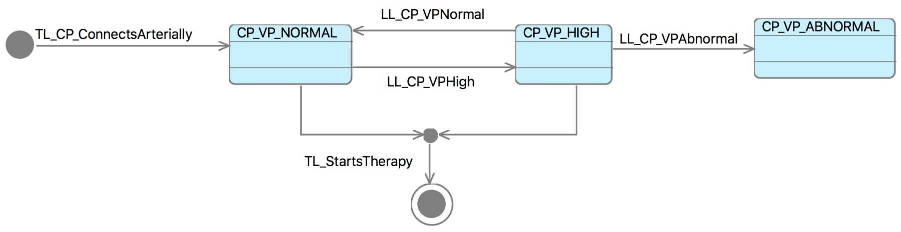


Figure 10: State-machine for monitoring VP during connecting patient

and LL_CP_VPAbnormal. Transition LL_CP_VPHigh happens when the low-level system detects that the VP pressure is too high (exceeding 450mmHg). Transition LL_CP_VPNormal is executed when the VP pressure becomes normal within 3 seconds. Finally, transition LL_CP_VPAbnormal takes place when the VP pressure stays for more than 3 seconds. As a result, we have the following timing constraints between the events.

$$Expiry(\text{LL\_CP\_VPHigh}, \text{LL\_CP\_VPNormal}, VP\_CP\_MAX\_TIME) \tag{9}$$

$$Deadline(\text{LL\_CP\_VPHigh}, \text{LL\_CP\_VPAbnormal}, VP\_CP\_MAX\_TIME) \tag{10}$$

$$Delay(\text{LL\_CP\_VPHigh}, \text{LL\_CP\_VPAbnormal}, VP\_CP\_MAX\_TIME) \tag{11}$$

The first constraint (9) ensures that the system can only go back to state *CP_VP_NORMAL* within the expiry time *VP_CP_MAX_TIME* (3 seconds) since the system detects that the VP pressure is too high, i.e., the last occurrence of *CP_VP_HIGH*. The constraints (10) and (11) specify that the system will go to the *CP_VP_ABNORMAL* state (exactly) when the deadline *VP_CP_MAX_TIME* passed since the last occurrence of *CP_VP_HIGH*.

Applying the modelling patterns from Section 3.3.1, we obtain the following Event-B events.

```
LL_CP_VPHigh :
when
  $LL\_CP\_VPTimer = -1$
  ...
then
  $LL\_CP\_VPTimer := VP\_CP\_MAX\_TIME$
  ...
end
```

```
LL_CP_VPNormal :
when
  $LL\_CP\_VPTimer \geq 0$
  ...
then
  $LL\_CP\_VPTimer := -1$
  ...
end
```

```
LL_CP_VPHigh :
when
  $LL\_CP\_VPTimer = 0$
  ...
then
  $LL\_CP\_VPTimer := -1$
  ...
end
```

```
Time :
any $d$ where
  $d \in \mathbb{N}_1$
  $(LL\_CP\_VPTimer \neq -1 \;\Rightarrow\; d \leq LL\_CP\_VPTimer)$
  ...
then
  $LL\_CP\_VPTimer := \max(\{-1, LL\_CP\_VPTimer - d\})$
  ...
end
```

Note that we have used the same timer variable $LL\_CP\_VPTimer$ for all the timing constraints. Moreover the guard $LL\_CP\_VPTimer = 0$ of LL_CP_VPHigh is a combination of the effect of the *Deadline* and *Delay* patterns.

Other requirements involving timing constraints are captured using a similar approach and omitted here.

3.3.3. Using ProB for Verifying Temporal Properties

With the introduction of the timing constraints, we are able to specify the temporal relationships between the different events of the system. An important point for us is to ensure that the model of the timing constraints for our model is *consistent*, in particular with respect to the temporal liveness properties [18]. With respect to our timing patterns, one of the most important consistency properties is to ensure the progression of time, i.e., occurrences of event Time cannot be blocked infinitely. In particular, for the *Deadline* pattern, the Time event is guarded, restricting its availability. In general, we want to verify that under reasonable assumptions, Time appears infinitely often, which is a liveness property.

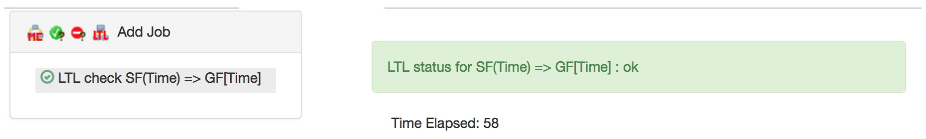
While safety properties, i.e., *something (bad) will not happen*, are usually modelled by invariants, *liveness properties* stating that *something (good) must* *happen* are not part of Event-B [19]. In this case, we make use of the capability of the ProB model checker to verify LTL temporal properties. Using the temporal language supported by ProB, we want to verify the following properties:

$$\mathsf{SF}(\mathsf{Time}) \quad \Rightarrow \quad \mathsf{G}\ \mathsf{F}[\mathsf{Time}]\ . \tag{12}$$

The full ProB-supported syntax of the LTL temporal logic can be seen in [20]. Without going into the semantical details of ProB-supported LTL, we present the meaning of some operators that are used is as follows.

- $\mathsf{G}\phi$ (*globally*): (temporal) property ϕ *always* holds.
- $\mathsf{F}\phi$ (*finally*): (temporal) property ϕ holds *eventually*.
- $\mathsf{SF}(\mathsf{e})$ (*strong fairness* for event e): if e is *enabled infinitely often* then *eventually* e is *is executed*.
- $[\mathsf{e}]$ (*occurrence* of event e): Event e is executed.

Property (12) states that under *strong-fairness* assumption of Time, Time executes infinitely often. The verification using ProB for Time event from **m02** can be seen in Figure 11.



Add Job

LTL check SF(Time) => GF[Time]

LTL status for SF(Time) => GF[Time] : ok

Time Elapsed: 58

Figure 11: Verification of (12) using ProB in **m02**

Another important temporal property is related to the *Deadline* pattern. For the *Expiry* and *Delay* patterns, occurrences of Response are optional. However, for the *Deadline* pattern, Response must be executed before the deadline is over. While the guard of the (*Deadline*)Response ensures that Response can only occur when the timer is not expired, there is nothing to guarantee that Response will indeed occur. The temporal property that we are interested in this case has the following form:

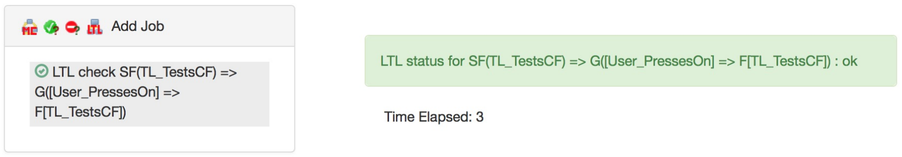
$$\mathsf{SF}(\mathsf{Response}) \Rightarrow \mathsf{G}([\mathsf{Trigger}] \Rightarrow \mathsf{F}[\mathsf{Response}])\ . \tag{13}$$

This states that under strong-fairness assumption of Response, (it is always the case) if Trigger happens then eventually Response is executed. Consider the deadline property (7), we want to verify the liveness property:

$$\mathsf{SF}(\mathsf{TL\_TestsCF}) \Rightarrow \mathsf{G}([\mathsf{User\_PressesOn}] \Rightarrow \mathsf{F}[\mathsf{TL\_TestsCF}])\ . \tag{14}$$

The verification result using ProB for this property from **m02** can be seen in Figure 12.

Note that while Event-B refinement preserves safety properties, it does not preserve liveness ones. As a result, we verify these properties using ProB in each refinement.


Add Job

LTL check SF(TL_TestsCF) => G([User_PressesOn] => F[TL_TestsCF])

LTL status for SF(TL_TestsCF) => G([User_PressesOn] => F[TL_TestsCF]) : ok

Time Elapsed: 3

Figure 12: Verification of (12) using ProB in **m02**

3.4. Validating using BMotion Studio

We use BMotion Studio to create a Domain-Specific Visualisation (DSV) of our iUML-B/Event-B model of the HD machine. The DSV consists of two views: a view of the user interface (UI) and a view of the environment of the HD machine. The description of the DSV is supported by listings in which observers and event handlers are described using JavaScript. However, the visual editor of BMotion Studio also provides a graphical user interface for creating observers and event handlers.

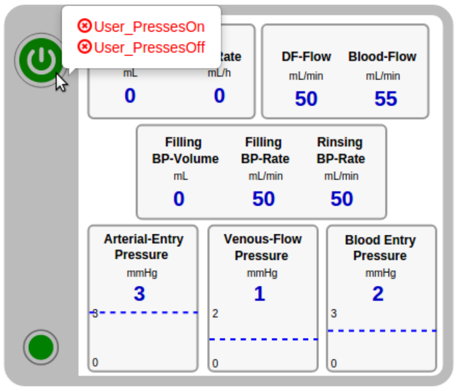
3.4.1. Visualising the UI display panel

Figure 13a demonstrates the DSV of the UI display panel. Each dialysis parameter is represented using simple graphical elements to display its description, unity and current value. In addition, for pressure parameters, the width and thresholds of the limits window are shown with the current value being represented by a horizontal dashed line.

Each dialysis parameter binds a *formula observer* that observes the respective state variable of the parameter. For instance, Listing 13b shows the formula observer for the blood flow parameter. Line 1 and 2 state that we register a new formula observer on the graphical element that matches the selector "#bloodFlow" (The prefix "#" is used to match a graphical element by its ID.[4]) Line 3 states that the observer should observe the variable *bloodFlow* during the animation. In lines 4 to 6 we define a trigger function that is called whenever a state change occurs. The reference to the matched graphical element (`e`) and the state values of the observed formulas (`v`) are passed as arguments to the trigger function. In line 5 we define the action which is made on the label whenever a state change occurs: the observer sets the text content of the label to the current value of the state variable *bloodFlow* (`val[0]`). We have defined the observers for the other dialysis parameters in a similar fashion.

The visualisation of the UI display panel also contains graphical elements and observers for the automated self test signal lamp (see lower left side of Fig. 13a), which is represented by a circle. The corresponding observer is responsible for indicating whether the automated self test has been successfully completed (change the signal lamp to green) or not (change the signal lamp to red) based on the observed formula `signal_status(CF_TESTING_SIGNAL)`.

[4]See jQuery selector API : http://api.jquery.com/category/selectors/.



(a) UI display panel visualisation

```
bms.observe("formula", {
 selector: "#bloodFlow",
 formulas: ["bloodFlow"],
 trigger: function(e, v) {
  e.text(v[0]);
}});

bms.executeEvent({
 selector: "#bt_power",
 events: [
  {name: "User_PressesOn"},
  {name: "User_PressesOff"}
 ]});
```

(b) Blood flow observer and on/off button execute event handler

Figure 13: Domain specific visualisation of UI display panel

We have used the *execute event handler* feature of BMotion Studio to add interactive components to our visualisation. As an example, Listing 13b (lines 8 to 13) shows the execute event handler for the HD machine on/off button (`#bt_power`) that wires the events User_PressesOn and User_PressesOff. In case of hovering the graphical element with the mouse a tooltip with the available events will be shown as demonstrated in Fig. 13a.

3.4.2. Visualising the environment of the HD machine

The DSV of the HD machine provides a second view that visualises the HD machine's environment as shown in Fig. 14. The objective of this view is to show how the different parts of the system are connected together. For instance, it contains graphical elements and observers that represent the dialysis pressure parameters (arterial-, venous-, and blood entry pressure) and their connection to the environment.

The visualisation is subdivided into SVG groups, where each group represents a different refinement level. Furthermore, each group binds a *refinement observer* that is responsible for showing or hiding the group depending on whether the observed refinement is part of the running animation or not. For instance, the group for refinement **m11** that contains the dialysis pressure parameter graphical elements, binds a refinement observer that observes refinement **m11**. Whenever **m11** is part of the running animation the observer sets the *visibility* attribute of the group to the value "visible" otherwise to "hidden". We have also created similar refinement observers for the UI display panel view. This helped us to focus on relevant parts of the system while validating a specific refinement level.

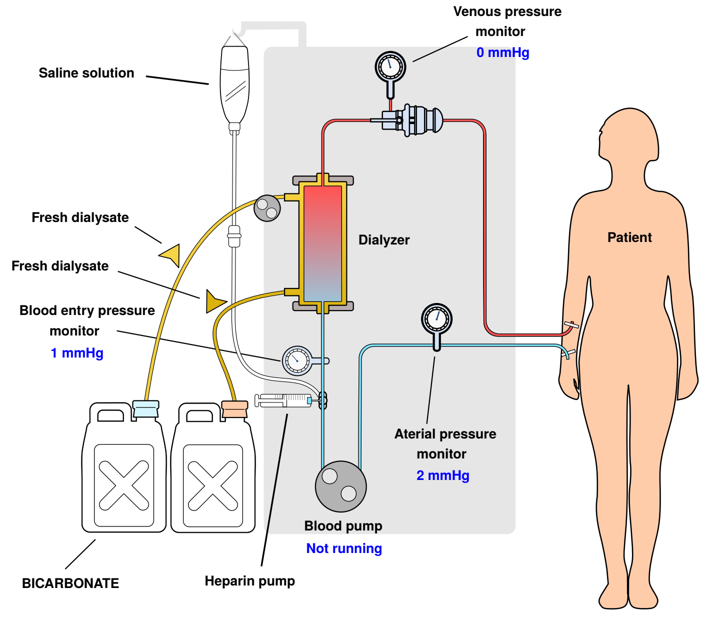


Figure 14: Domain specific visualisation of the environment of the HD machine

3.4.3. Application of the DSV

The benefits of more effective validation of the HD machine's Event-B model justify the extra effort required to develop a DSV. The visualisation helped us reach a common understanding about the model and to identify faulty behaviour and errors during development. This is particularly valuable when the formal model becomes complex in later refinement levels. Animation tools with only textual representation of the state are insufficient for validation purposes. In the case of the HD machine, requirements such as **S-2**–**S-4**, **R-5**–**R-13** are modelled by the enabled-ness of iUML-B transitions (ultimately events). Such properties are cumbersome to formulate as a proof obligation in Event-B but can be readily demonstrated via BMotion Studio. Hence in many cases we use BMotion Studio to validate whether the requirements have been adequately taken into account. BMotion Studio also helped us to discover problems with our model during its development, especially mistakes leading to liveness problems, where the HD machine cannot make any progress.

The DSV also enables domain experts to validate our formal model in terms of user interactions. This can be compared with prototyping techniques.

3.5. Validating using Co-simulation

Safety requirements **S-8**, **S-9** and **S-10** concern adjustments to the Blood Flow Rate (BF). **S-8** requires the demanded BF to be lowered if Arterial Pressure (AP) is low. We conclude that there is an inverse relationship between

BF and AP. **S-8** also states that the AP to BF relationship is affected by the fistula needle type. **S-9** indicates that low AP can result in reduced BF. Hence the achieved BF should be monitored and treatment time adjusted accordingly. **S-10** requires that BF should be optimal (presumably after consideration of **S-8** and **S-9**). We assume that this means as close to the user selected BF as possible and that a stable closed loop control of the blood pump is needed. In order to validate the specification of a suitable control system we use the continuous domain modelling tool Dymola to create a model of the environment which we co-simulate with the Event-B model of the control system (Figure 15), which is extracted from the formal model developed in Section 3.2.

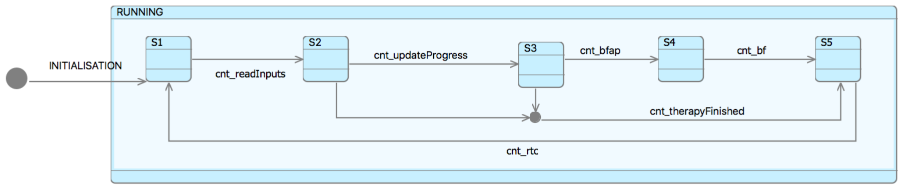


Figure 15: iUML-B model of BF and AP control cycle

Figure 16 shows the continuous domain Dymola model of the physical interaction between BF and AP. This detail of the environment being controlled was not given in the specification and we have been unable to find any reference that describes such properties. We have invented an example behaviour, based on typical pump suction properties, for the purposes of illustration. In order to validate this model we also developed a Dymola model of the control system. Once the environment model behaved as desired it was exported as a FMU which allows it to be run as a simulation outside of the Dymola tool. We then imported the FMU into the Rodin co-simulation tool, linked its I/O with our Event-B model of the control system and co-simulated the combined models.

The transition *cnt_readinputs* obtains new values provided by the Environment FMU simulation. The transition *cnt_updateProgress* subtracts the achieved BF for the cycle period from the total blood volume required to be processed. If the total has been processed, *cnt_therapyFinished* sets the demanded BF to 0. Otherwise, *cnt_bfap* calculates the demanded BF which is the user configured BF adjusted for AP (i.e. in accordance with S8). This adjustment is implemented as a simple linear interpolation function from (0,0) to (70,initial BF) which is limited outside this domain. Transition *cnt_bf* adjusts the output commanded BF to adjust for the achieved BF using a proportional error control. This final adjustment corresponds to control of the Blood Pump (BP) speed to achieve the desired BF except that we abstracted from BP units for simplicity. We chose to model AP in % of some nominal initial AP and BF in ml/min with a control cycle period of 100ms. The initial BF is set 30ml/min in the following analysis.

Our initial co-simulation results (Figure 17) showed that the AP was correctly controlled to a steady value of 72% by lowering BF to below 20ml/min.

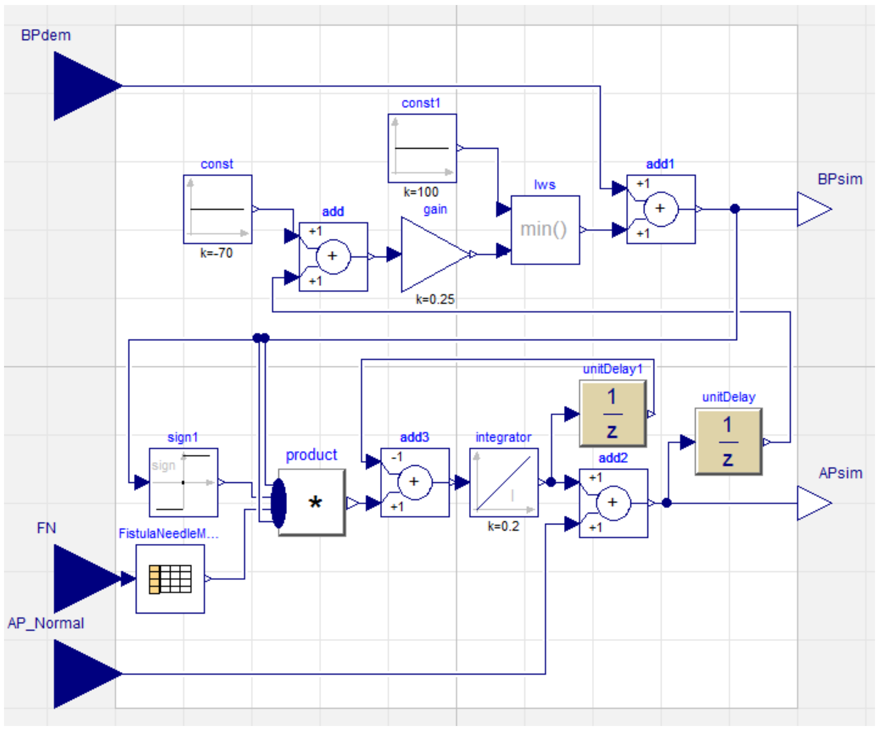
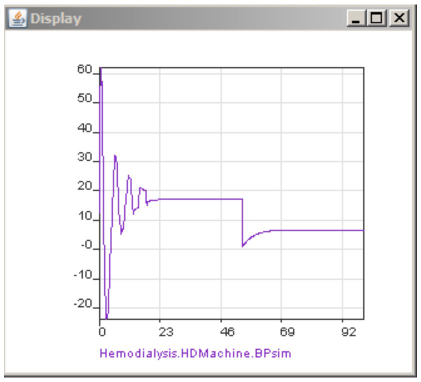


Figure 16: Dymola model of interaction between BF and AP

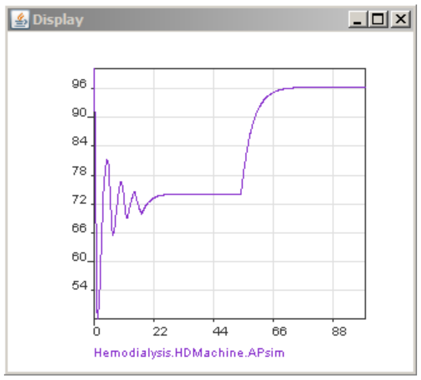
However, the initial response is very unstable. It is interesting to note that we did not see this instability when we first tested the environment model in Dymola only (i.e. using a Dymola model of the control in place of the Event-B model). We believe this is because we did not accurately model the discrete periodic cycle and therefore the response rate of the Dymola version of the control was fast enough to mask the problem. This demonstrates the advantage of testing the actual Event-B model which is inherently discrete. To improve stability we decreased the gain of the proportional control. This improved stability but resulted in a degraded AP of approx. 55%. This is due to a larger residual offset error which is an inherent problem of proportional controllers. A possible solution would be to introduce an integral term to the controller which would remove the residual offset.

3.6. Requirements Tracing with ProR

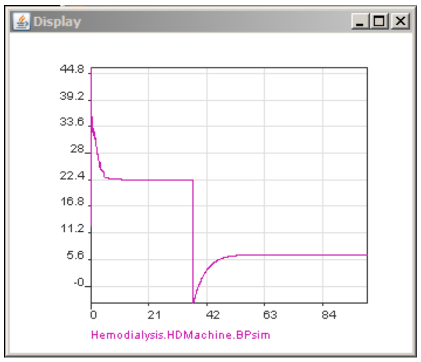
We use ProR for structuring the requirements, linking the requirements to the formal model and identifying the verification/validation status of the requirement. This provides traceability to ensure that all requirements have been addressed and verified/validated in the model. The default configuration of ProR provides a generic requirements management tool that has no support for linking to model elements such as iUML-B. However, since ProR is an EMF



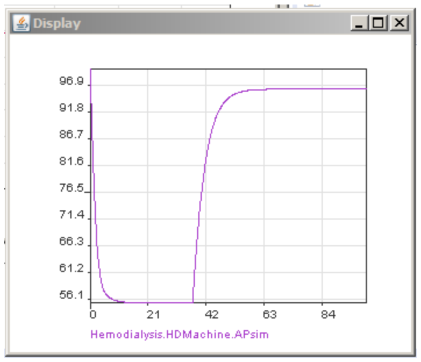
(a) Unstable control of BF



(b) Unstable control of AP



(c) Stable control of BF



(d) Stable control of AP

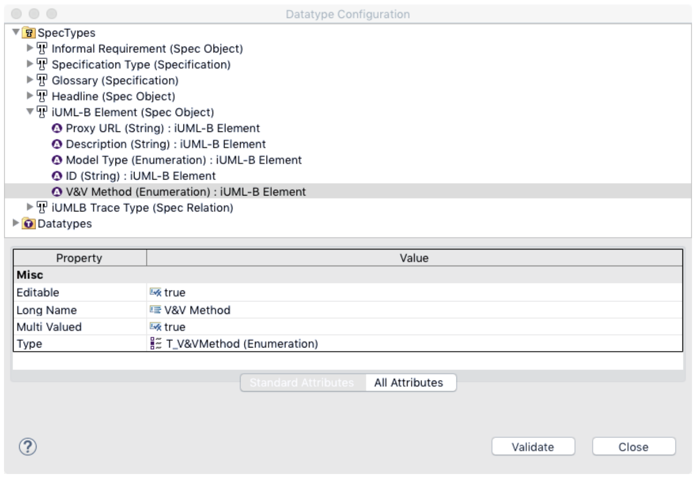
Figure 17: Co-simulation plots showing unstable and stable control of BF and AP

based implementation of ReqIF, it can be configured to support linking to any kind of object that has EMF-based tool support.[5] In order to link to iUML-B model elements we configured ProR by adding an additional specification object type called *iUML-B Element* with new attributes as shown in Figure 18.

The *iUML-B Element* spec object has the following attributes. A *Description* attribute is used to give a user readable note about the model element; the type of the modelling element is given in *Model Type*; the *Proxy URL* attribute is the machine readable reference to the model element and the methods by which the requirement has been verified and/or validated are given in the attribute *V&V Method*. New table columns are configured to display these attributes.

Figure 19 shows the complete general safety requirements being managed by ProR. *iUML-B Element* objects are attached to each requirement. A range

[5]We envisage utilising this facility, in the future, to encompass other development artifacts.

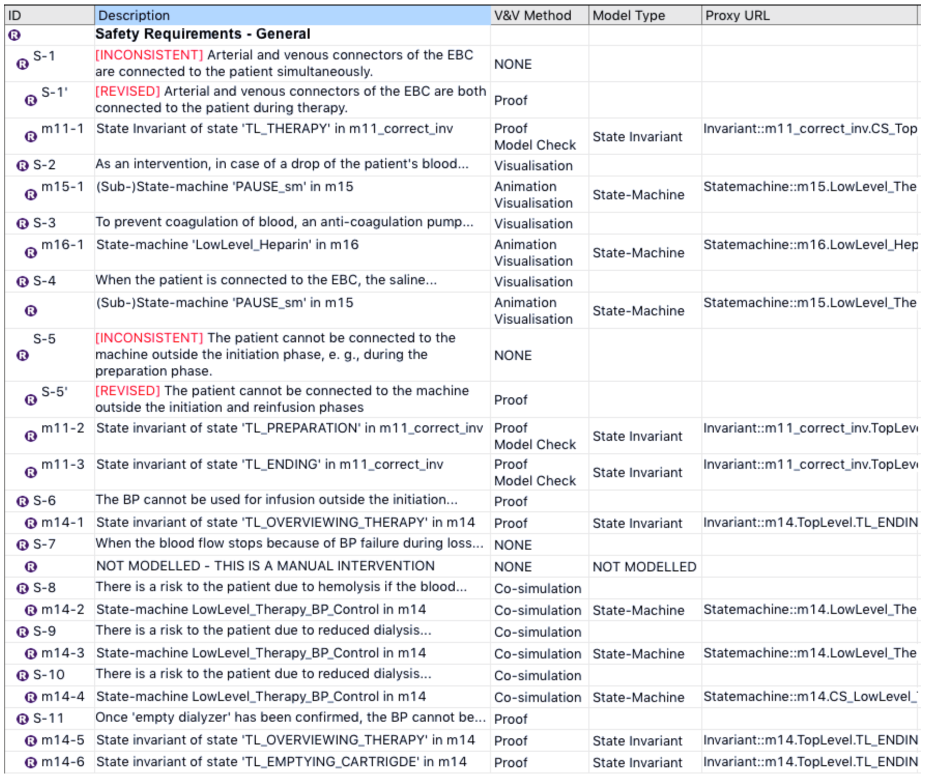


Datatype Configuration

- SpecTypes
  - Informal Requirement (Spec Object)
  - Specification Type (Specification)
  - Glossary (Specification)
  - Headline (Spec Object)
  - iUML-B Element (Spec Object)
    - Proxy URL (String) : iUML-B Element
    - Description (String) : iUML-B Element
    - Model Type (Enumeration) : iUML-B Element
    - ID (String) : iUML-B Element
    - V&V Method (Enumeration) : iUML-B Element
  - iUMLB Trace Type (Spec Relation)
- Datatypes

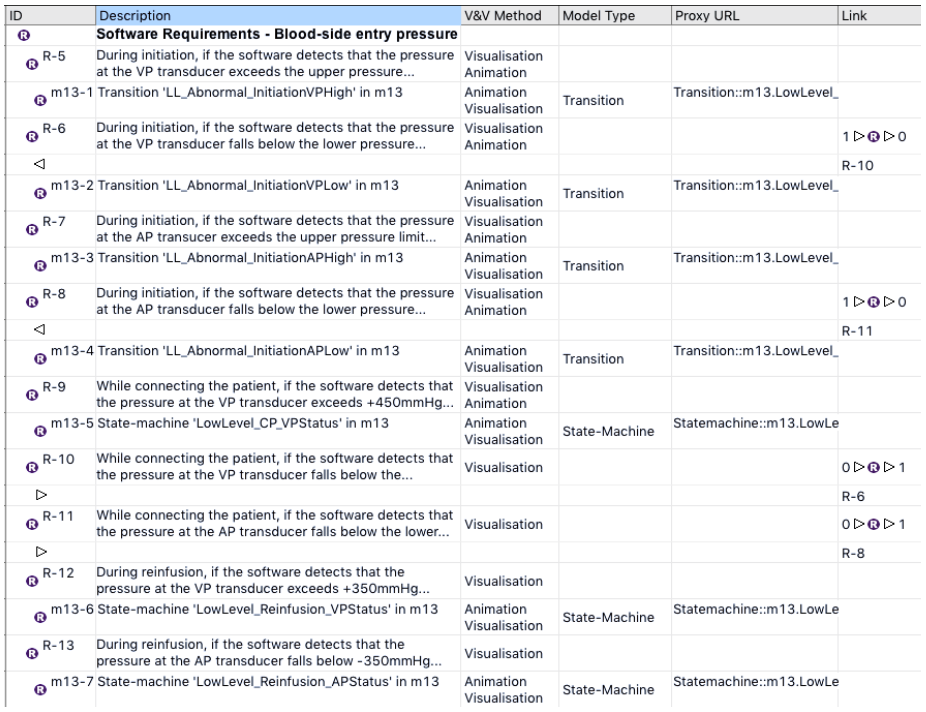
Property	Value
Misc	
Editable	true
Long Name	V&V Method
Multi Valued	true
Type	T_V&VMethod (Enumeration)

Figure 18: Configuring ProR to support new features



ID	Description	V&V Method	Model Type	Proxy URL
	Safety Requirements - General			
S-1	[INCONSISTENT] Arterial and venous connectors of the EBC are connected to the patient simultaneously.	NONE		
S-1'	[REVISED] Arterial and venous connectors of the EBC are both connected to the patient during therapy.	Proof		
m11-1	State Invariant of state 'TL_THERAPY' in m11_correct_inv	Proof Model Check	State Invariant	Invariant::m11_correct_inv.CS_Top
S-2	As an intervention, in case of a drop of the patient's blood...	Visualisation		
m15-1	(Sub-)State-machine 'PAUSE_sm' in m15	Animation Visualisation	State-Machine	Statemachine::m15.LowLevel_The
S-3	To prevent coagulation of blood, an anti-coagulation pump...	Visualisation		
m16-1	State-machine 'LowLevel_Heparin' in m16	Animation Visualisation	State-Machine	Statemachine::m16.LowLevel_Hep
S-4	When the patient is connected to the EBC, the saline...	Visualisation		
	(Sub-)State-machine 'PAUSE_sm' in m15	Animation Visualisation	State-Machine	Statemachine::m15.LowLevel_The
S-5	[INCONSISTENT] The patient cannot be connected to the machine outside the initiation phase, e. g., during the preparation phase.	NONE		
S-5'	[REVISED] The patient cannot be connected to the machine outside the initiation and reinfusion phases	Proof		
m11-2	State invariant of state 'TL_PREPARATION' in m11_correct_inv	Proof Model Check	State Invariant	Invariant::m11_correct_inv.TopLev
m11-3	State invariant of state 'TL_ENDING' in m11_correct_inv	Proof Model Check	State Invariant	Invariant::m11_correct_inv.TopLev
S-6	The BP cannot be used for infusion outside the initiation...	Proof		
m14-1	State invariant of state 'TL_OVERVIEWING_THERAPY' in m14	Proof	State Invariant	Invariant::m14.TopLevel.TL_ENDIN
S-7	When the blood flow stops because of BP failure during loss...	NONE		
	NOT MODELLED - THIS IS A MANUAL INTERVENTION	NONE	NOT MODELLED	
S-8	There is a risk to the patient due to hemolysis if the blood...	Co-simulation		
m14-2	State-machine LowLevel_Therapy_BP_Control in m14	Co-simulation	State-Machine	Statemachine::m14.LowLevel_The
S-9	There is a risk to the patient due to reduced dialysis...	Co-simulation		
m14-3	State-machine LowLevel_Therapy_BP_Control in m14	Co-simulation	State-Machine	Statemachine::m14.LowLevel_The
S-10	There is a risk to the patient due to reduced dialysis...	Co-simulation		
m14-4	State-machine LowLevel_Therapy_BP_Control in m14	Co-simulation	State-Machine	Statemachine::m14.CS_LowLevel_
S-11	Once 'empty dialyzer' has been confirmed, the BP cannot be...	Proof		
m14-5	State invariant of state 'TL_OVERVIEWING_THERAPY' in m14	Proof	State Invariant	Invariant::m14.TopLevel.TL_ENDIN
m14-6	State invariant of state 'TL_EMPTYING_CARTRIGDE' in m14	Proof	State Invariant	Invariant::m14.TopLevel.TL_ENDIN

Figure 19: Using ProR to manage requirements



ID	Description	V&V Method	Model Type	Proxy URL	Link
Ⓡ	**Software Requirements - Blood-side entry pressure**				
Ⓡ R-5	During initiation, if the software detects that the pressure at the VP transducer exceeds the upper pressure...	Visualisation Animation			
Ⓡ m13-1	Transition 'LL_Abnormal_InitiationVPHigh' in m13	Animation Visualisation	Transition	Transition::m13.LowLevel_	
Ⓡ R-6	During initiation, if the software detects that the pressure at the VP transducer falls below the lower pressure...	Visualisation Animation			1 ▷ Ⓡ ▷ 0
◁					R-10
Ⓡ m13-2	Transition 'LL_Abnormal_InitiationVPLow' in m13	Animation Visualisation	Transition	Transition::m13.LowLevel_	
Ⓡ R-7	During initiation, if the software detects that the pressure at the AP transucer exceeds the upper pressure limit...	Visualisation Animation			
Ⓡ m13-3	Transition 'LL_Abnormal_InitiationAPHigh' in m13	Animation Visualisation	Transition	Transition::m13.LowLevel_	
Ⓡ R-8	During initiation, if the software detects that the pressure at the AP transducer falls below the lower pressure...	Visualisation Animation			1 ▷ Ⓡ ▷ 0
◁					R-11
Ⓡ m13-4	Transition 'LL_Abnormal_InitiationAPLow' in m13	Animation Visualisation	Transition	Transition::m13.LowLevel_	
Ⓡ R-9	While connecting the patient, if the software detects that the pressure at the VP transducer exceeds +450mmHg...	Visualisation Animation			
Ⓡ m13-5	State-machine 'LowLevel_CP_VPStatus' in m13	Animation Visualisation	State-Machine	Statemachine::m13.LowLe	
Ⓡ R-10	While connecting the patient, if the software detects that the pressure at the VP transducer falls below the...	Visualisation			0 ▷ Ⓡ ▷ 1
▷					R-6
Ⓡ R-11	While connecting the patient, if the software detects that the pressure at the AP transducer falls below the lower...	Visualisation			0 ▷ Ⓡ ▷ 1
▷					R-8
Ⓡ R-12	During reinfusion, if the software detects that the pressure at the VP transducer exceeds +350mmHg...	Visualisation			
Ⓡ m13-6	State-machine 'LowLevel_Reinfusion_VPStatus' in m13	Animation Visualisation	State-Machine	Statemachine::m13.LowLe	
Ⓡ R-13	During reinfusion, if the software detects that the pressure at the AP transducer falls below -350mmHg...	Visualisation			
Ⓡ m13-7	State-machine 'LowLevel_Reinfusion_APStatus' in m13	Animation Visualisation	State-Machine	Statemachine::m13.LowLe	

Figure 20: Using ProR to manage requirements - software requirements

of different V&V methods are shown. Generally requirements that are modelled using state invariants are verified by proof, although the ProB model checker is often also used to check these. Requirements that are modelled by a State-machine or an individual transition are validated using (iUML-B State-machine) Animation and (BMotion Studio) Visualisation or, for **S-8** to **S-10**, co-simulation. Where requirements were found to need corrections (e.g. **S-1** and **S-5** could not be proved and required weakening as explained in section 3.2.5) the original requirement was retained with an explanation as well as the new version that is linked to a iUML-B Element. Note that the red annotations, INCONSISTENT and REVISED, were manually entered into the requirements based on the results of the v&v tools. Requirement **S-7** is a manual intervention and therefore was not modelled. For completeness we added a dummy *iUML-B Element* that explains the reason for not modelling.

Figure 20 shows some of the software requirements that are modelled by State-Machine Transitions. This figure also illustrates that some requirements have dependency links where **R-10** and **R-11** are derived (and modelled by) **R-6** and **R-8** respectively.

ProR enables us to keep track of the requirements and how they are modelled. It provides documented evidence that the requirements have been addressed and provides a record of issues that have been discovered in the requirements during the modelling process as well as recording the verification and/or validation performed for each requirement. We consider that the primary purpose of our modelling is the early discovery of problems in the requirements. Therefore it is essential to have a mechanism and good tool support for managing the requirements in a close integration with the modelling tools.

4. Related Work and Comparison

The HD machine is a form of Active Medical Device (AMD), 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 [21]. Due to the important impact of embedded AMD software, according to the latest directive 2007/47/EC of the EU concerning medical devices [22], standalone software can also be considered as an AMD. Recently safety of such software has become a challenge. Several incidents have been reported that were caused by medical software faults [23] and therefore medical software-related recalls are increasing [24].

The standards for AMD validation [25, 26, 27, 22] are mainly dedicated to the physical and electrical components of a device. Two main references concerning medical *software* development are the standard IEC 62304 [28] (International Electrotechnical Commission) and the "General Principles of Software Validation" [29] established by the the US Food and Drug Administration (FDA). These documents provide *general* guidelines for software engineering. There is a gap to address methods and techniques to ensure safety of AMD software.

Jetley and Iyer [30] highlight the role of model-based design and hence formal methods, as mathematical-based techniques, in ensuring safety and reliability of medical software to satisfy regulatory agencies such as FDA and Singh [31] highlights the benefits of using formal methods, particularly Event-B, in medical devices to ensure safety of medical protocols.

Several research efforts have been dedicated to the application of formal methods in medical devices. Singh et al. [32] present a general view of the field of medical devices and certification issues through the pacemaker challenge and include a discussion of the formalisation of a cardiac pacemaker in Event-B. Gehlot and Sloane [33] propose developing a prototype formal verification and validation toolkit to improve safety in wireless medical device networks. Some of this work is dedicated to the HD machine.

Arcaini et al. [34] present formal modelling and verification of the HD machine using Abstract State Machines (ASM). The ASM model is incrementally built through refinement. The refinement correctness proof is done by hand or, for a particular kind of refinement, using the SMT-based tool *ASMRefProver*. The model is validated by interactive simulation using the simulator *AsmetaS* and scenario-based validation which requires a manual set of scenarios using the textual notation *Avalla*. The model is verified by means of the model checker *AsmetaSMV*. Requirement traceability is given by the relation between abstract and refined models, at each refinement step.

Banach [35] examines the HD machine case study in Hybrid Event-B, an extension of Event-B that supports continuously varying behaviour as well as the usual discrete changes of state. The model, including refinement and invariant-preserving verification is presented. However tool support for Hybrid Event-B is lacking, and the work is based on hand proofs.

Fayolle et al. [36] present a specification of the HD machine by a coupling of Algebraic State-Transition Diagrams (ASTD) and B-like methods. Event-B allows the data model and the safety properties of the system to be captured whereas ASTD is used to specify the ordering of actions and to constrain the execution of the events. The model is incrementally developed using extended refinement of both methods. The authors believe that for some of the refinement steps, the B-method and ASTD refinement definitions are restrictive and they propose to use a new refinement definition in future work. The work mainly focuses on the specification of the general behaviour of the machine, safety requirements are dealt with at the last steps of the specification. Also it does not report on verification/validation techniques.

Gomes and Butterfield [37] present a formal model of some aspects of the HD machine case study using the Circus specification notation. Due to the lack of automated verification tools for Circus, the model is manually translated into machine-readable CSP (CSPm) so that the FDR3 refinement-checker can be used. The work does not include validation techniques.

UPPAAL is an integrated tool environment for modelling, simulation and verification of real-time systems [38]. Systems are modelled as networks of timed automata which would suit some of the requirements of the HD machine. The use of UPPAAL for AMD applications is illustrated by Guo et al. [39] who present an approach that transforms medical best practice guidelines to state chart models using UPPAAL and Daw et al. [40] who verify the UML model of a medical device using the UPPAAL model checker. Kamali and Petre [41] provide a comparison of UPPAAL and Event-B. UPPAAL provides clock variables to model timing behaviour of real-time systems whereas Event-B has no in-built support for timing and, consequently, it has to be dealt with by explicitly modelling a clock with variables and events. UPPAAL lacks support for refinement, particularly data refinement, compared to Event-B. UPPAAL does not support verification of continuous behaviours whereas in this paper we have reported using co-simulation with Event-B and elsewhere advances have been made in ways to model hybrid (discrete and continuous) behaviour within Event-B [42, 35]. Some attempts have been made to combine the advantages of both UPPAAL and Event-B. For example, Berthing et al. [43], Vain et al. [44] and Siavashi et al. [45] combine Event-B behaviour with UPPAAL timing and use this combination to support refinement of timed specifications.

In summary, there are many alternative modelling techniques which could or have been used for AMD. However, we have not found any that can compete in the range of techniques and strength of tool support that we are able to draw upon in the Rodin, Event-B based modelling environment. In particular, we highlight the following strengths of our approach:

- the range of Verification & Validation (V&V) methods: we used ProR for requirement tracing, ProB to verify the temporal (liveness) properties and to animate and simulate our models, theorem proving for verification and co-simulation for validation of dynamic properties.
- the range of visualisation techniques: the preparation for haemodialysis is a complex sequential process with sub-process branches, and hence highlights the need for visualisation techniques to illustrate the process. In addition, supporting graphical views can help medical experts to validate the specification. Our work benefits from different visualisation techniques: iUML-B contributes as a graphical modelling language with automatic generation of Event-B and BMotion Studio provides a more concrete visualisation of the behaviour of the model.
- integrated tool support: the Rodin toolset is an integrated platform both for modelling and proving. The V&V and visualisation techniques are integrated in Rodin as plug-ins.

5. Conclusion

The HD machine is predominantly a sequential process of user interactions with few safety properties that can be expressed as constraints on state. During the therapy stage the machine controls the dynamic properties of AP and BF. At first sight it appeared that this case study would not illustrate the strengths of our modelling tools very well since Event-B verifies the preservation of invariant properties over discrete state-changing events. However, the case study gave us an opportunity to focus on (1) verification of temporal properties using ProB and (2) validation tools that we use to develop useful models and (3) tracing of requirements into the formal model using ProR.

In order to model timing constraints, we develop modelling patterns to capture the relationship between events. The introduction of timing constraints required the model to be verified against not only safety but also liveness properties. We have utilised the capability of ProB for model checking LTL temporal properties to verify the consistency of our model with timing constraints.

Verification may result in a consistent model but we need user validation to ensure the usefulness of our models. For this case study, we therefore used the validation tools to drive a manual assessment of the model. iUML-B state-machine modelling tools map readily to the process steps of the requirements and their animation enables us to 'see' the sequential flows of the model. BMotion Studio visualisation tools link the process to a more realistic representation of the HD machine which allows us to disassociate ourselves from the model giving a stronger validation. For validation of the dynamic control of AP versus BF we use a continuous domain model of the controlled parameters to co-simulate with our iUML-B/Event-B models to provide a strong validation of the stability and effectiveness of the modelled control scheme.

The summary of the requirements (from [1]) that have been modelled and verified/validated within our development is as follows.

- *Invariant Proofs*: **S-1'**, **S-5'**, **S-6**, **S-11**
- *Simulation Validation*: **S-2**, **S-3**, **S-4**, **R-1**–**R-15**, **R-17**–**R-19**, **R-22**
- *Co-simulation*: **S-8**, **S-9**, **S-10**

Many requirements are validated using simulation/animation techniques. One exception is requirement **R-16**: "while connecting the patient, the software shall use a timeout of 310 seconds after the first start of the BP. After this timeout, the software shall change to the initiation phase ". An attempt to model this requirement leads to an invalid iUML-B state-machine. We found that the requirement is inconsistent: while connecting patient, the system is *already in the initiation phase*. It is not clear to us what the intended meaning of the requirement is.

In order to aid our modelling, we have use some Event-B modelling patterns in our development. The summary of the pattern can be seen in Table 1. The patterns provide the modelling templates that are systematically instantiated to model the HD machine. In the future, we plan to collect more modelling patterns and provide consistent descriptions including templates for both iUML-B and the corresponding Event-B translation.

Pattern name	**When to use**
Controlling equipment	The controller issues command for the equipment to perform some task and proceed accordingly to the outcome of the task.
Delay (timing)	The responsing event *can only occur after a certain delay* from the occurence of the triggering event.
Expiry (timing)	The responsing event *can only occur within a certain duration* after the occurence of the triggering event.
Deadline (timing)	The responsing event *must occur within a certain duration* after the occurence of the triggering event.

Table 1: Summary of the Event-B modelling patterns

Overall, our model has 17 levels of refinement with a total of 664 proof obligations. With the standard settings for the built-in automatic provers, 590 obligations (89%) are automatically proven. Upon inspection of the remaining proof obligations, it is clear that the standard automatic provers are unable to resolve the various state enumerations generated by the state-machines and nested state-machines. As a result, we enabled the SMT plugin for Rodin [46] which uses various SMT solvers as back-ends for discharging proof obligations. With SMT solvers, 650 obligations (98%) are automatically proven, while the remaining 14 obligations (2%) are proved manually. Note that the high percentage of automatic proofs for this development is partly due to there being no complex safety properties. Furthermore, other types of properties, such as liveness, are validated and verified using other means than proofs, e.g., model checking and simulation. The model was developed by 3 experts working on different parts of the development: modelling the process using iUML-B, developing the co-simulation, and validating the model using BMotion Studio. We estimate that a total of 2 person-months was spent on developing the model.

In order to verify liveness properties, we have utilised the ProB model checker. In particular, due to the fact that refinement in Event-B does not preserve liveness properties, the verification has to be repeated at each level of refinement. A future research direction is to investigate the use of liveness-preserving refinement approaches such as TLA+ [47], Unit-B [48]. Existing work linking Event-B and TLA+ such as [49, 50] will be investigated.

In the future, we will continue to develop the BF control using co-simulation to improve its accuracy without degrading stability and responsiveness. We plan to investigate ways to provide validation records that might be used as evidence in a safety case. For example, BMotion Studio could be enhanced to provide and replay traces of animations. We also want to extend the ProR configuration to include traceability links to Verification & Validation (V&V) evidence, i.e., link to proof obligations and/or recorded animation/visualisation trace. Finally we are interested in providing tool support for assisting the modelling of timing constraints. In particular, given some timing patterns, we could automatically generate (similar to iUML-B) the Event-B data elements, such as guards and actions, to complement the existing Event-B elements.

During the case study we have shown that the Event-B based modelling tools are able to address adequately all of the different kinds of requirements in the HD machine in appropriate ways. While other methods have strengths in particular areas, none benefit from such an integrated platform of tools. We conclude that the hypothesis is supported.

References

[1] A. Mashkoor, The hemodialysis machine case study, `http://www.cdcc.faw.jku.at/ABZ2016/HD-CaseStudy.pdf` (2015).

[2] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[3] M. Leuschel, M. Butler, ProB: An automated analysis toolset for the B method, Software Tools for Technology Transfer (STTT) 10 (2) (2008) 185–203.

[4] C. Snook, Modelling control process and control mode with synchronising orthogonal statemachines, in the Proceedings of the B2011 Workshop, Limerick (2011).

[5] V. Savicks, C. Snook, A framework for diagrammatic modelling extensions in Rodin, in: Rodin Workshop 2012, Fontainbleau, 2012, pp. 31–32, `http://deploy-eprints.ecs.soton.ac.uk/382/`.

[6] C. Snook, iUML-B statemachines, in: Proceedings of the Rodin Workshop 2014, Toulouse, France, 2014, pp. 29–30, http://eprints.soton.ac.uk/365301/.

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

[8] M. Jastram, The ProR approach: Traceability of requirements and system descriptions, Ph.D. thesis, Heinrich Heine University Düsseldorf (2012).
URL http://docserv.uni-duesseldorf.de/servlets/DocumentServlet?id=21882

[9] T. S. Hoang, C. F. Snook, L. Ladenberger, M. J. Butler, Validating the requirements and design of a hemodialysis machine using iUML-B, BMotion Studio, and co-simulation, in: Butler et al. [51], pp. 360–375.
URL http://dx.doi.org/10.1007/978-3-319-33600-8_31

[10] Object Management Group, Requirements Interchange Format (ReqIF), http://www.omg.org/spec/ReqIF/ (July 2016).

[11] T. S. Hoang, An introduction to the Event-B modelling method, in: Industrial Deployment of System Engineering Methods, Springer-Verlag, 2013, pp. 211–236.

[12] L. Ladenberger, J. Bendisposto, M. Leuschel, Visualising Event-B models with B-Motion Studio, in: Proceedings of FMICS 2009, Vol. 5825 of Lecture Notes in Computer Science, Springer, 2009, pp. 202–204.

[13] Lukas Ladenberger, BMotion Studio for ProB project website, http://stups.hhu.de/ProB/w/BMotion_Studio (Jan. 2016).

[14] Dassault Systemes, Catia Systems Engineering Dymola, http://www.3ds.com/products-services/catia/products/dymola, (accessed Jan, 2016).

[15] V. Savicks, M. Butler, J. Colley, Co-simulating Event-B and continuous models via FMI, in: Proceedings of the 2014 Summer Simulation Multiconference, SummerSim '14, Society for Computer Simulation International, San Diego, CA, USA, 2014, pp. 37:1–37:8.
URL http://dl.acm.org/citation.cfm?id=2685617.2685654

[16] FMI Steering Committee, Functional Mock-up Interface, https://www.fmi-standard.org, (accessed Jan, 2016).

[17] M. R. Sarshogh, M. J. Butler, Specification and refinement of discrete timing properties in Event-B, ECEASST 46.
URL http://journal.ub.tu-berlin.de/eceasst/article/view/701

[18] L. Lamport, Proving the correctness of multiprocess programs, IEEE Trans. Software Eng. 3 (2) (1977) 125–143.

[19] T. S. Hoang, J. Abrial, Reasoning about liveness properties in Event-B, in: S. Qin, Z. Qiu (Eds.), Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings, Vol. 6991 of Lecture Notes in Computer Science, Springer, 2011, pp. 456–471.
URL `http://dx.doi.org/10.1007/978-3-642-24559-6_31`

[20] Software Engineering and Programming Languages Group, ProB User Manual on LTL Model Checking, University of Düsseldorf, Düsseldorf, Germany, `http://www3.hhu.de/stups/prob/index.php/LTL_Model_Checking` (2017).

[21] The Council of the European Union. Council Directive 93/42/EEC of 14 June 1993 concerning medical devices., off J L 169. 1993;12 (07):143 (June 1993).

[22] The Council of the European Union. Directive 2007/47/EC of the European Parliament and of the council. (September 2007).

[23] K. Sandler, L. Ohrstrom, L. Moy, R. McVay, Killed by code: Software transparency in implantable medical devices (2010) 308–319.

[24] D. R. Wallace, D. R. Kuhn, Failure modes in medical device software: An analysis of 15 years of recall data 8 (4) (2001) 351–371.

[25] ISO 13485: medical devices - quality management systems - requirements for regulatory purposes. (2003).

[26] IEC 606011:2005 medical electrical equipment part 1: General requirements for basic safety and essential performance. (2005).

[27] ISO 14971: medical devices - application of risk management to medical devices. (2007).

[28] IEC 62304 - medical device software - software lifecycle processes. (2006).

[29] U.S. Food and Drug Administration (FDA). General principles of software valida- tion; final guidance for industry and FDA staff, version 2.0. (January 2002).

[30] R. P. Jetley, S. P. Iyer, P. L. Jones, A formal methods approach to medical device review, IEEE Computer 39 (4) (2006) 61–67.
URL `http://dx.doi.org/10.1109/MC.2006.113`

[31] N. K. Singh, Using Event-B for Critical Device Software Systems, Springer, 2013.
URL `http://dx.doi.org/10.1007/978-1-4471-5260-6`

[32] N. K. Singh, M. Lawford, T. S. E. Maibaum, A. Wassyng, Formalizing the cardiac pacemaker resynchronization therapy, in: Digital Human Modeling - Applications in Health, Safety, Ergonomics and Risk Management: Ergonomics and Health - 6th International Conference, DHM 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II, 2015, pp. 374–386.
URL http://dx.doi.org/10.1007/978-3-319-21070-4_38

[33] V. Gehlot, E. B. Sloane, Ensuring patient safety in wireless medical device networks, IEEE Computer 39 (4) (2006) 54–60.
URL http://dx.doi.org/10.1109/MC.2006.125

[34] P. Arcaini, S. Bonfanti, A. Gargantini, E. Riccobene, How to assure correctness and safety of medical software: The hemodialysis machine case study, in: Butler et al. [51], pp. 344–359.
URL http://dx.doi.org/10.1007/978-3-319-33600-8_30

[35] R. Banach, Hemodialysis machine in hybrid Event-B, in: Butler et al. [51], pp. 376–393.
URL http://dx.doi.org/10.1007/978-3-319-33600-8_32

[36] T. Fayolle, M. Frappier, F. Gervais, R. Laleau, Modelling a hemodialysis machine using Algebraic State-Transition Diagrams and B-like methods, in: Butler et al. [51], pp. 394–408.
URL http://dx.doi.org/10.1007/978-3-319-33600-8_33

[37] A. O. Gomes, A. Butterfield, Modelling the haemodialysis machine with circus, in: Butler et al. [51], pp. 409–424.
URL http://dx.doi.org/10.1007/978-3-319-33600-8_34

[38] G. Behrmann, A. David, K. G. Larsen, A tutorial on uppaal, in: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures, Vol. 3185 of Lecture Notes in Computer Science, Springer, 2004, pp. 200–236. doi:10.1007/978-3-540-30080-9_7.
URL https://doi.org/10.1007/978-3-540-30080-9_7

[39] C. Guo, S. Ren, Y. Jiang, P. Wu, L. Sha, R. B. B. Jr., Transforming medical best practice guidelines to executable and verifiable statechart models, in: 7th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2016, Vienna, Austria, April 11-14, 2016, IEEE Computer Society, 2016, pp. 34:1–34:10. doi:10.1109/ICCPS.2016.7479121.
URL https://doi.org/10.1109/ICCPS.2016.7479121

[40] Z. Daw, R. Cleaveland, M. Vetter, Formal verification of software-based medical devices considering medical guidelines, Int. J. Computer Assisted Radiology and Surgery 9 (1) (2014) 145–153. doi:10.1007/

s11548-013-0919-2.
URL https://doi.org/10.1007/s11548-013-0919-2

[41] M. Kamali, L. Petre, Uppaal vs event-b for modelling optimised link state routing, in: K. Barkaoui, H. Boucheneb, A. Mili, S. Tahar (Eds.), Verification and Evaluation of Computer and Communication Systems - 11th International Conference, VECoS 2017, Montreal, QC, Canada, August 24-25, 2017, Proceedings, Vol. 10466 of Lecture Notes in Computer Science, Springer, 2017, pp. 189–203. doi:10.1007/978-3-319-66176-6_13.
URL https://doi.org/10.1007/978-3-319-66176-6_13

[42] J. Abrial, W. Su, H. Zhu, Formalizing hybrid systems with event-b, in: J. Derrick, J. S. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, E. Riccobene (Eds.), Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings, Vol. 7316 of Lecture Notes in Computer Science, Springer, 2012, pp. 178–193. doi:10.1007/978-3-642-30885-7_13.
URL https://doi.org/10.1007/978-3-642-30885-7_13

[43] J. Berthing, P. Boström, K. Sere, L. Tsiopoulos, J. Vain, Refinement-based development of timed systems, in: J. Derrick, S. Gnesi, D. Latella, H. Treharne (Eds.), Integrated Formal Methods - 9th International Conference, IFM 2012, Pisa, Italy, June 18-21, 2012. Proceedings, Vol. 7321 of Lecture Notes in Computer Science, Springer, 2012, pp. 69–83. doi:10.1007/978-3-642-30729-4_6.
URL https://doi.org/10.1007/978-3-642-30729-4_6

[44] J. Vain, L. Tsiopoulos, J. Guin, Developing multi-view contracts using event-b and uppaal timed automata, in: H. Wang, M. Mokhtari (Eds.), 21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016, Dubai, United Arab Emirates, November 6-8, 2016, IEEE Computer Society, 2016, pp. 126–134. doi:10.1109/ICECCS.2016.024.
URL https://doi.org/10.1109/ICECCS.2016.024

[45] F. Siavashi, M. Waldn, L. Tsiopoulos, J. Vain, Modelling critical systems with time constraints in event-b, in: T. Uustalu, J. Vain (Eds.), Proceedings of the 25th Nordic Workshop on Programming Theory, NWPT '13, Tallin University of Technology, 2013, p. 13.

[46] D. Déharbe, P. Fontaine, Y. Guyot, L. Voisin, Integrating SMT solvers in rodin, Sci. Comput. Program. 94 (2014) 130–143.
URL https://doi.org/10.1016/j.scico.2014.04.012

[47] L. Lamport, Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2002.

[48] S. Hudon, T. S. Hoang, J. S. Ostroff, The Unit-B method — refinement guided by progress concerns, Software and System Modeling 15 (4) (2016)

1091–1116.
URL http://dx.doi.org/10.1007/s10270-015-0456-2

[49] D. Hansen, M. Leuschel, Translating B to TLA+ for validation with TLC, Sci. Comput. Program. 131 (2016) 109–125.
URL https://doi.org/10.1016/j.scico.2016.04.014

[50] D. Méry, M. Poppleton, Formal modelling and verification of population protocols, in: E. B. Johnsen, L. Petre (Eds.), Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings, Vol. 7940 of Lecture Notes in Computer Science, Springer, 2013, pp. 208–222.
URL https://doi.org/10.1007/978-3-642-38613-8_15

[51] M. J. Butler, K. Schewe, A. Mashkoor, M. Biró (Eds.), Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings, Vol. 9675 of Lecture Notes in Computer Science, Springer, 2016.
URL http://dx.doi.org/10.1007/978-3-319-33600-8