The original publication is available at http://dx.doi.org/10.1007/978-3-319-33600-8_31
Appeared in the Proceedings of the ABZ2016 Conference © Springer-Verlag

Validating the Requirements and Design of a Hemodialysis Machine Using iUML-B, BMotion Studio, and Co-simulation

Thai Son Hoang[1], Colin Snook[1], Lukas Ladenberger[2], and Michael Butler[1]

[1] ECS, University of Southampton, U.K.
{t.s.hoang,cfs,mjb}@ecs.soton.ac.uk
[2] University of Dusseldorf, Germany
ladenberger@cs.uni-dusseldorf.de

Abstract. We present a formal specification of a hemodialysis machine (HD machine) using Event-B. We model the HD machine using iUML-B state-machines and class diagrams and build a corresponding BMotion Studio visualisation. We focus on validation 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 co-simulation tools to validate against a continuous model of the physical behaviour.

Keywords: Hemodialysis Machine, Event-B, Validation, Visualisation, iUML-B, BMotion Studio, Co-Simulation.

1 Introduction

This paper describes our approach to formally model the requirements and design of a *hemodialysis machine* (HD machine) [8]. The HD machine is used by patients with kidney failure to remove waste products from their blood. We identify how we deal with the safety requirements that are defined for the HD machine [8].

We use Event-B [1], 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 focus on validation using (i) diagrams to aid the modelling of the sequential properties of the requirements, and (ii) ProB-based animation, visualisation and simulation tools to explore the behaviour of our models. Where appropriate we use the proof capabilities of Event-B to verify safety constraints.

Our contribution is to demonstrate how different kinds of safety requirements can be verified or validated using the tools available. To do this we provide (i) a model of the HD machine using iUML-B [11, 10, 12] 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. The graphical model and visualisation enable us to analyse and validate the behaviour of the HD machine.

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 and validation of its requirements and design using iUML-B, BMotion Studio, and co-simulation. Finally, we summarise and conclude in Section 4. For more information and resources, we refer the reader to our website: `http://stups.hhu.de/ProB/ABZ16`. The website contains the Event-B model and the BMotion Studio visualisation of the HD machine.

2 Background

Event-B. Event-B [1] 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*, *invariants* 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. 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 [5]. Event-B is supported by the *Rodin Platform* (Rodin) [2], an extensible toolkit which includes facilities for modelling, verifying about the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches.

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.

BMotion Studio. In this paper we have used the new version[3] of BMotion Studio [6] to create a *domain specific visualisation* (DSV) of our Event-B model of the

[3] Originally BMotion Studio was developed as a separate plug-in for Rodin [7].

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 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.

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. In order to validate such models a *MultiSim* plug-in [9] 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 simulated programmatically using ProB and the continuous model is a *Functional Mock-up Unit* (FMU) [4] which has been exported from a continuous domain modelling tool such as Dymola [3]. The plug-in controls the coordination and communication between the co-operating simulations.

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 [8] and we will not repeat those requirements in this paper. We suggest the readers to study this section together with the requirements document [8] and the formal model available from the web site `http://stups.hhu.de/ProB/ABZ16`. 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), how we use BMotion Studio to validate our model (Section 3.3). For dynamic properties that cannot be expressed in Event-B, we show how co-simulation helps us to validate such properties (Section 3.4).

3.1 Development Strategy

The hemodialysis process is highly sequential with several sub-processes. In the design described in [8], 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 hemodialysis process, interacting with the users, while the low-level one controls the sub-processes by monitoring and regulating the behaviour of the HD machine.

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

Refinement strategy in Event-B is often influenced by the correctness proofs. In this case, we do not find any verification difficulty, 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.

m0: Models the main phases of the hemodialysis process for the top-level control system, i.e., *PREPARATION*, *INITIATION*, and *ENDING*
m1: Models the *sub-processes* within each main phases for the top-level control system.
m2: Models the low-level control's *automatic testing of control functions*.
m3: Models the actual (physical) result of testing the HD machine's control functions.
m4: Model the *message passing communication* between the low-level control system and the HD machine for testing control functions.
m5: Models the set of *signals*.
m6: Models the signal for indication of control function testing result.
m7: Models the connection of concentrate to the HD machine.
m8: Models the *setting of rinsing parameters*.
m9: Models the sequence of connecting patient
m10: Models the physical connection of the patient (arterially and venously).
m11: Models the three pressure monitors and the system normal/abnormal states.
m12: Models various abnormal blood-side pressures.
m13: Models the blood pump, actual blood flow and abnormal situations when monitoring the blood flow.
m14: Models arterial bolus.
m15: Models heparin bolus.

3.2 Modelling using iUML-B

Modelling Sequential Processes The hemodialysis 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. Figures 1 and 2 illustrate how we model the sequential processes in **m0** and **m1**.

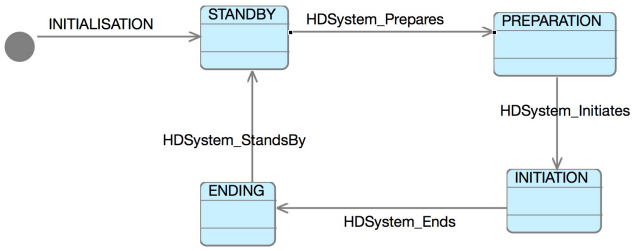


Fig. 1: State-machine *CS_TopLevel* in **m0**

Figure 1 shows the main phases of the hemodialysis process (with the additional *STANDBY* state). In **m1**, we introduce nested state-machines for states *PREPARATION*, *INITIATION*, *ENDING* to model the sequential sub-steps of each main phase. Figure 2 gives an example of the nested state-machine for the *PREPARATION* state.

The incoming/outgoing transitions of the super-state *PREPARATION* in **m0**, i.e., HDSystem_Prepares and HDSystem_Initiates, are respectively refined to HDSystem_StartsTestingCF and HDSystem_StartsConnectingPatient in **m1**. Using the encoding where each iUML-B state is represented by a constant from an enumerated carrier set, HDSystem_Prepares and HDSystem_StartsTestingCF are straightforwardly translated into Event-B as follows. Here, variable *CS_TopLevel* indicates the current state of the top-level state-machine, and the current state of the nested state-machine in state *PREPARATION* is represented by variable *Preparation_sm*.

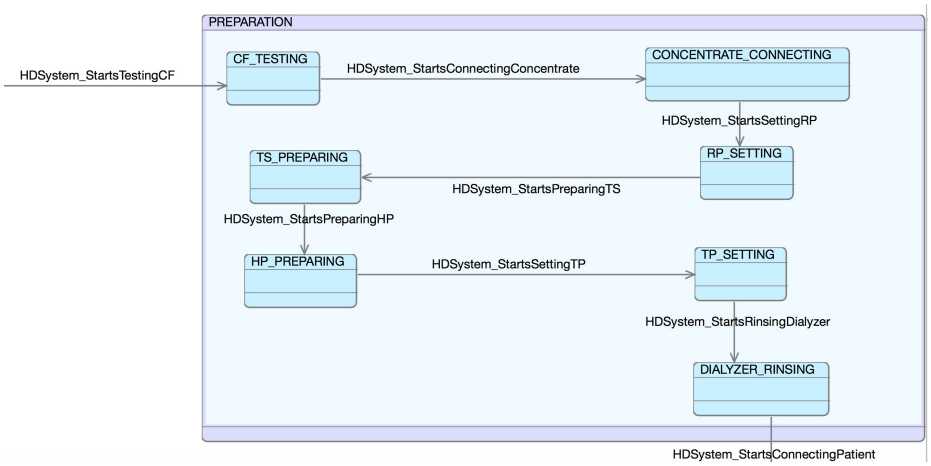


Fig. 2: Nested state-machine for state *PREPARATION* in **m1**

HDSystem_Prepares :
when
$CS\_TopLevel = STANDBY$
then
$CS\_TopLevel := PREPARATION$
end

HDSystem_StartsTestingCF :
when
$CS\_TopLevel = STANDBY$
then
$CS\_TopLevel := PREPARATION$
$Preparation\_sm := CF\_TESTING$
end

Top-level vs. Low-level Control Systems The top-level control system, which maintains the sequential hemodialysis 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 3 illustrates the state-machine for the low-level control system performing testing of *Control Functions* (CF).

The CS_LowLevel_StartsTestingCF and CS_LowLevel_StandsBy transitions are guarded by $Preparation\_sm = CF\_TESTING$ and $CS\_TopLevel = STANDBY$, respectively, to ensure that they can only be carried out in the correct phases as specified in the top-level control state-machine $CS\_TopLevel$.

The top-level control system can only move from state $CF\_TESTING$ to the next when the CF testing is successful. Using the alternative Event-B encoding for state-machine $CS\_LowLevel\_CFTesting$, where each state is represented by a Boolean variable, this is modelled by a guard on the transition elaborating HDSystem_StartsConnectingConcentrate in state-machine $CS\_TopLevel$ (Figure 2) stating that $CS\_LL\_CF\_TESTED\_OK = \text{TRUE}$.

A Pattern for Controlling Physical Equipments 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 those in [1] and we applied them to iUML-B state-machines. We show below how we incorporate the physical testing of CF into the formal model. The low-level control system for CF testing is illustrated in Figure 3. In **m3**, a variable $HDMachine\_CFTestedOK$ is introduced to denote the result of the HD machine's CF test and a new event HDMachine_CFTests is allowed to

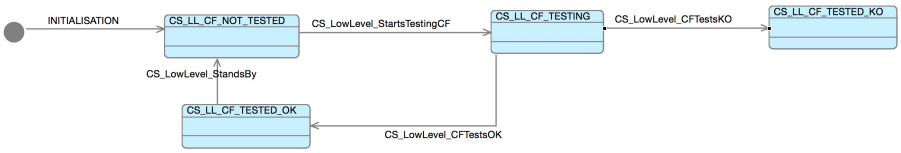


Fig. 3: Low-level control system for CF testing

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. *CS_LowLevel_CFTesting*. Transitions CS_LowLevel_CFTestsOK and CS_LowLevel_CFTestsKO of state-machine *CS_LowLevel_CFTesting* are directed by the actual result of the test: they are guarded to select a pass/fail response according to *HDMachine_CFTestedOK*, i.e., *HDMachine_CFTestedOK* = TRUE or *HDMachine_CFTestedOK* = FALSE.

In the refinement **m4**, we introduce the communication between the controller and the HD machine. Two new variables *CS_2_HD_StartsCFTesting* and *HD_2_CS_CFTestingFinished* are introduced to model the message exchange. Flag *CS_2_HD_StartsCFTesting* is set in event CS_LowLevel_StartsTestingCF and unset in HDMachine_CFTests. Invariant

$$CS\_2\_HD\_StartsCFTesting = \text{TRUE} \Rightarrow \text{CS\_LowLevel\_StartsTestingCF} = \text{TRUE}$$

allows us to refine HDMachine_CFTests's guard to *CS_2_HD_StartsCFTesting* = TRUE. Similarly, *HD_2_CS_CFTestingFinished* is set in HDMachine_CFTests and unset in CS_LowLevel_CFTestsOK and CS_LowLevel_CFTestsKO, while the guards of CS_LowLevel_CFTestsOK and CS_LowLevel_CFTestsKO are strengthened by adding the condition *HD_2_CS_CFTestingFinished* = TRUE.

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. Consider the state-machine *CS_TopLevel* in **m2**. We wish to ensure that when the system is in the *INITIATION* phase, the CF should have been successfully tested. We add an invariant CS_LowLevel_CFTestsOK = TRUE to state *INITIATION*. The translation of the state-invariant in Event-B is, as expected, i.e.,

$$CS\_TopLevel = INITIATION \Rightarrow \text{CS\_LowLevel\_CFTestsOK} = \text{TRUE} \ .$$

To prove the above invariant, an invariant is added to the *PREPARATION* state stating that $Preparation\_sm \neq CF\_TESTING \Rightarrow \text{CS\_LowLevel\_CFTestsOK} = \text{TRUE}$.

Animation/Model Checking to Validate Requirements Consider **m10**, we introduce a state-machine to model the physical connections/disconnections of the patient to the HD machine arterially and venously. The patient is connected to the machine in the first step of the *INITIATION* phase and disconnected from the machine in the first step of the *ENDING* phase. Requirements **S-1** and **S-5** from [8] 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.

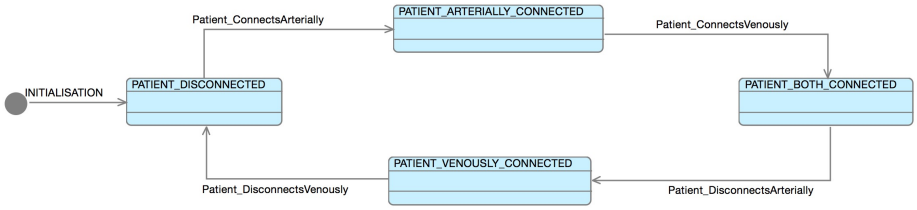


Fig. 4: Patient connections to the HD machine

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

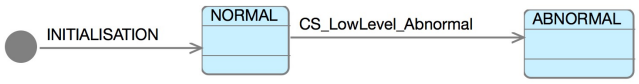
$$PATIENT\_CONNECTIONS \neq PATIENT\_DISCONNECTED \Rightarrow PATIENT\_CONNECTIONS = PATIENT\_BOTH\_CONNECTED \ .$$

and **S-5** as state-invariants for states *PREPARATION* and *ENDING* specifying that $PATIENT\_CONNECTIONS = PATIENT\_DISCONNECTED$. 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. During *PREPARATION*, the patient is connected first arterially and then venously contradicting **S-1**. The patient is still connected both arterially and venously when the reinfusion step starts, i.e., outside the initiation phase contradicting **S-2**. We therefore weaken the requirements **S-1** and **S-5** as follows.

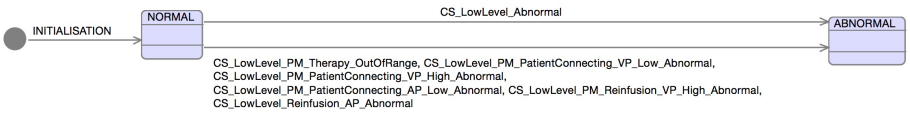
S-1' Arterial and venous connectors of the EBC are *both* connected to the patient *during therapy*.

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

Modelling Abnormal Behaviours During hemodialysis 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 **m11** (Figure 5a). When we introduced the pressure monitor in **m12**, various abnormal conditions can be detected. The events modelling such detection are a refinement of the abstract event CS_LowLevel_Abnormal (Figure 5b). Note that we still keep the abstract event CS_LowLevel_Abnormal to be able to detect more abnormal behaviours in future refinements.



(a) State-machine *CS_LowlLevel_Overall* in **m11**



(b) Nested state-machine *CS_LowlLevel_Overall* in **m12**

Fig. 5: Modelling Abnormal Behaviours

3.3 Validating using BMotion Studio

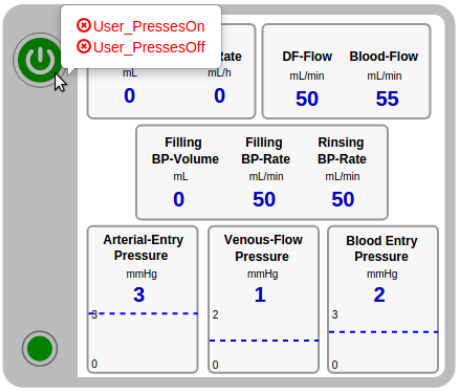
We use BMotion Studio to create a DSV of our iUML-B/Event-B model of the HD machine. The DSV consists of two views: a view of the user interface (UI) and a view of the environment of the HD machine. The description of the DSV is supported by listings in which observers and event handlers are described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

Visualising the UI display panel Figure 6a 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 6b 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. 6a),

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



(a) UI display panel visualisation

```
1  bms.observe("formula", {
2   selector: "#bloodFlow",
3   formulas: ["bloodFlow"],
4   trigger: function(e, v) {
5    e.text(v[0]);
6  }});
7
8  bms.executeEvent({
9   selector: "#bt_power",
10  events: [
11   {name: "User_PressesOn"},
12   {name: "User_PressesOff"}
13  ]});
```

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

Fig. 6: Domain specific visualisation of UI display panel

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)`.

We have used the *execute event handler* feature of BMotion Studio to add interactive components to our visualisation. As an example, Listing 6b (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. 6a.

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. 7. 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

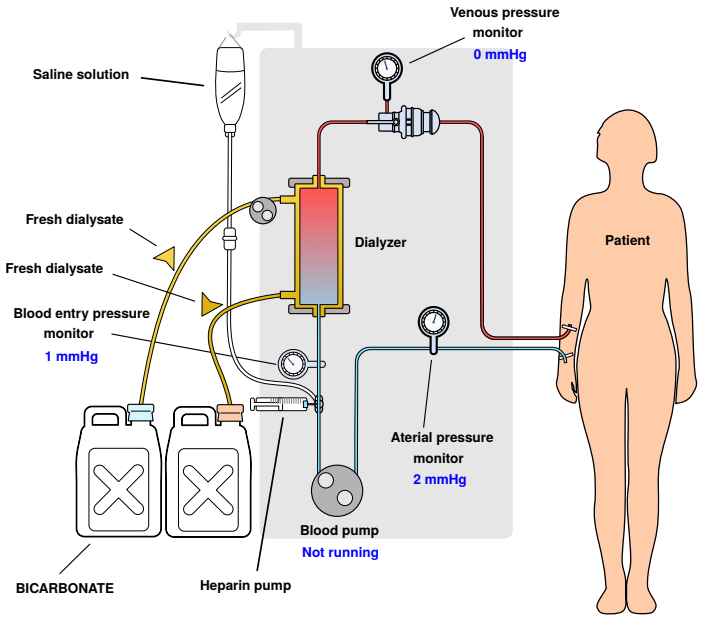


Figure CC BY 3.0 YassineMrabet (https://commons.wikimedia.org/wiki/File:Hemodialysis-en.svg)

Fig. 7: Domain specific visualisation of the environment of the HD machine

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.

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. Indeed, we plan to show our formal model and DSV to medical scientists and physicians at the university medical centre of Duesseldorf.

BMotion Studio provides a feature for sharing a DSV online. This is useful for demonstration purposes as well as for sharing the visualisation between the persons involved in developing the model. We refer the reader to the website stated in Section 1 which contains an online live visualisation of the HD machine.

3.4 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, which is extracted from the formal model developed in Section 3.2.

Figure 8 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. 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 10) showed that the AP was correctly controlled to a steady value of 72% by lowering BF to below 20ml/min. 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.

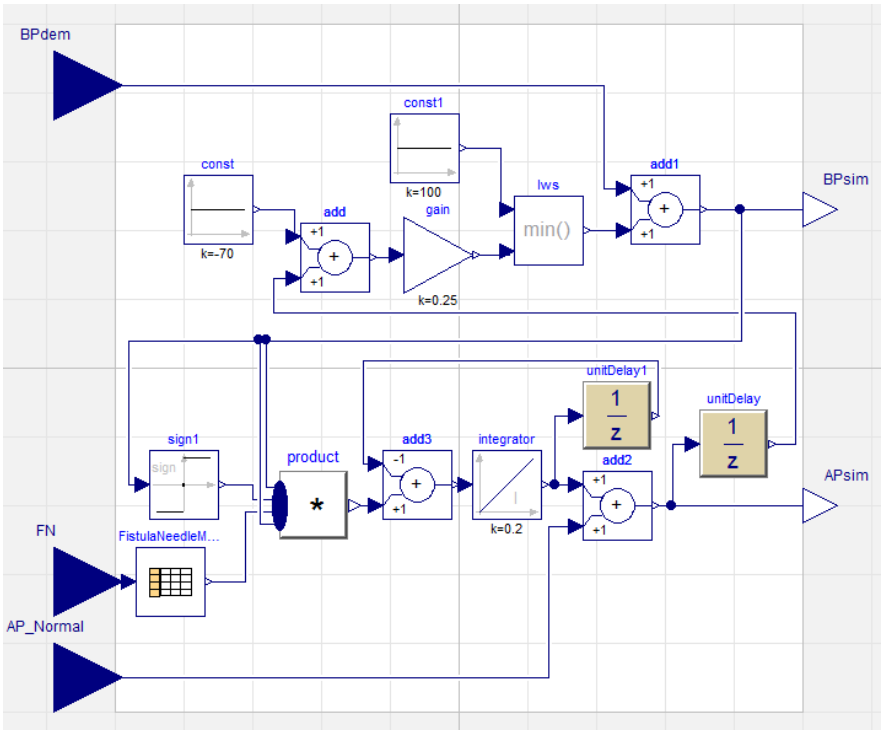


Fig. 8: Dymola model of interaction between BF and AP

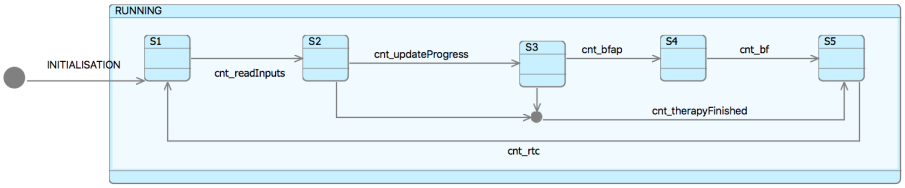
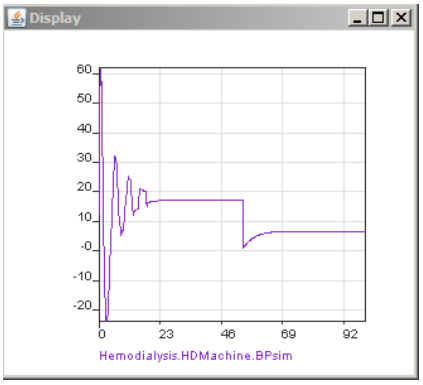
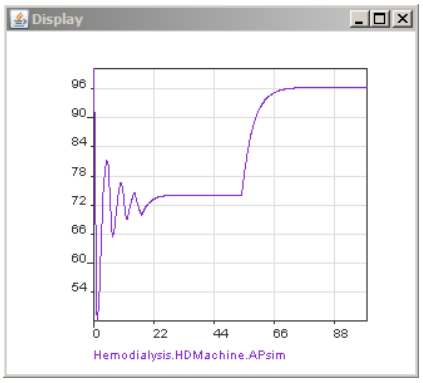


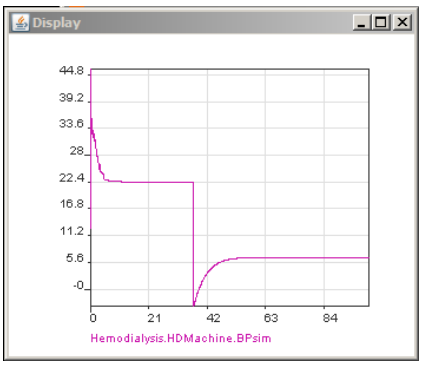
Fig. 9: iUML-B model of BF and AP control cycle



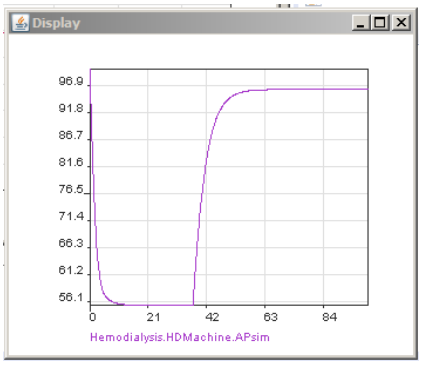
(a) Unstable control of BF



(b) Unstable control of AP



(c) Stable control of BF



(d) Stable control of AP

Fig. 10: Co-simulation plots showing unstable and stable control of BF and AP

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.

4 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 the validation tools that we use to develop useful models. Proofs may result in a correct 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 [8]) 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 the future, we plan to address the remaining requirements using similar techniques. 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.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. J-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, November 2010.
3. Dassault Systemes. Catia Systems Engineering Dymola. http://www.3ds.com/products-services/catia/products/dymola. (accessed Jan, 2016).
4. FMI Steering Committee. Functional Mock-up Interface. https://www.fmi-standard.org. (accessed Jan, 2016).
5. T.S. Hoang. An introduction to the Event-B modelling method. In *Industrial Deployment of System Engineering Methods*, pages 211–236. Springer-Verlag, 2013.
6. L. Ladenberger. BMotion Studio for ProB project website. http://stups.hhu.de/ProB/w/BMotion_Studio, January 2016.
7. L. Ladenberger, J. Bendisposto, and M. Leuschel. Visualising Event-B models with B-Motion Studio. In *Proceedings of FMICS 2009*, volume 5825 of *Lecture Notes in Computer Science*, pages 202–204. Springer, 2009.
8. A. Mashkoor. The hemodialysis machine case study. `http://www.cdcc.faw.jku.at/ABZ2016/HD-CaseStudy.pdf`, 2015.
9. V. Savicks, M. Butler, and J. Colley. Co-simulating Event-B and continuous models via FMI. In *Proceedings of the 2014 Summer Simulation Multiconference*, SummerSim '14, pages 37:1–37:8, San Diego, CA, USA, 2014. Society for Computer Simulation International.
10. V. Savicks and C. Snook. A framework for diagrammatic modelling extensions in Rodin. In *Rodin Workshop 2012, Fontainbleau*, 2012.
11. C. Snook. Modelling control process and control mode with synchronising orthogonal statemachines. In *B2011, Limerick*, 2011.
12. C. Snook. iUML-B statemachines. In *Proceedings of the Rodin Workshop 2014*, Toulouse, France, 2014. `http://eprints.soton.ac.uk/365301/`.