Hemodialysis Machine in Hybrid Event-B

Richard Banach

School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

Abstract. The hemodialysis machine case study is examined in Hybrid Event-B (an extension of Event-B that includes provision for continuously varying behaviour as well as the usual discrete changes of state). A broadly component based strategy is adopted, using the multi-machine and coordination facilities of Hybrid Event-B. Since, like most medical procedures, hemodialysis is under overall human control, it is largely a sequential process, with some branching to deal with exceptional circumstances. This makes for a relatively uncomplicated modelling framework, provided a model of the operator is included in order to capture the handling of exceptions.

1 Introduction

This paper reports on a treatment of the hemodialysis machine case study using Hybrid Event-B. Hybrid Event-B [6] is an extension of the well known Event-B framework, in which continuously varying state evolution, along with the usual discrete changes of state, is admitted. Although tool support for Hybrid Event-B is lacking at the time of writing, theoretical case studies like the present one are of great benefit in exercising the formalism, to develop the most appropriate modelling metaphors and to confirm that the formalism lives up to its expectations in terms of expressivity in the face of a wide variety of applications challenges.

The hemodialysis machine case study requirements are in [16]. As almost always happens in such situations, the requirements document itself does not completely document all aspects of the situation, and supplementary information is helpful to assist in framing the development. Additional details can be found in, e.g. [2, 17, 13, 12, 15]. While such references are certainly helpful in putting the problem in context, they actually say little about the detailed operation of hemodialysis machines, perhaps because such details are commercially sensitive. Nevertheless, they reveal that hemodialysis is overwhelmingly a sequential process, with branches corresponding to a number of exceptional circumstances that may arise along the way. What also becomes clear is that the most critical component of the process is the human operator, since the operator is relied on to steer the hemodialysis procedure through its various phases, and to initiate handling of any exceptions — in the most benign cases, operators are the patients themselves when hemodialysis is self-administered (though an assistant is normally present).[1] The sequential aspect, and the ultimate reliance on human expertise, are in

[1] In the most extreme cases, the operator or assistant undertakes a manual reinfusion of the patient when a power failure causes the equipment to halt.

fact common to most medical procedures, which take place in a series of steps, perhaps branching on the basis of the outcomes of earlier steps, and always under ultimate hands-on human control. Thus, the case study we are faced with is simplified in being essentially sequential, but needs to have the human operator as an agent in the system, since vital parts of the management of the procedure are under direct operator control.

A feature of the requirements in [16] is that they do not go into many details regarding the physical/mechanical processes involved in the operation of the machine. At a low enough level of description, it becomes clear that there are a number of subsystems in the machine, and we can infer that they enjoy a certain amount of autonomy, but without details, it is difficult to do more than guess at some interactions with the control system. The same goes for the behaviour of the operator, which is so vital. To truly model the operation of hemodialysis, we would need to understand how experienced hemodialysis nurses go about their work. All of this rather underutilises the capacity of Hybrid Event-B to model such sytems faithfully, and specifically, to write the more incisive invariants that would strengthen confidence in the design.

The rest of the paper is as follows. In Section 2 we overview the hemodialysis machine and its operation. In Section 3 we give a sketch of multi-machine Hybrid Event-B as used here. In Section 4 we give some generalities about the Hybrid Event-B development of the case study, and in Section 5 we discuss it in more detail. Section 6 discusses the role of invariants (and hence the prospects for verification), in a system like the present one, where so much depends on the operator. Section 7 concludes.

2 The Hemodialysis Machine and its Operation

In Fig. 1 we reproduce the internal structure of a hemodialysis machine from [16]. The patient's direct connection to the hemodialysis machine is via the extracorporeal blood circuit (EBC), shown in more detail in Fig.2. This monitors blood pressure on entry, infuses Heparin to prevent clotting inside the machine, and pumps the blood into the dialyser. On exit from the dialyser the blood pressure is monitored again, and passes a sensor that detects blood arrival (for filling of the EBC at procedure startup) and blood departure (for emptying the EBC of blood at procedure end). The sensor also monitors the presence of air in the blood returned to the patient to prevent dangerous amounts from being infused into the patient's bloodstream. Although not shown in Fig. 1, the EBC is also directly connected to the control system.

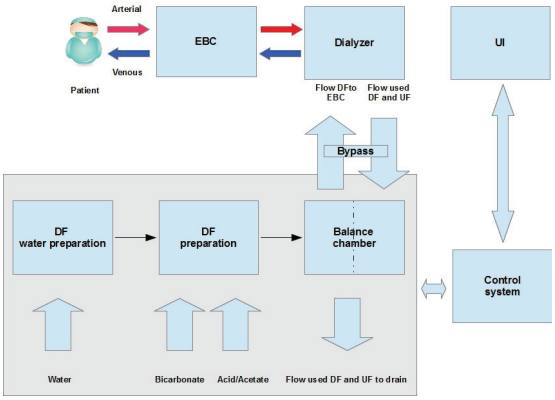

Fig. 1. Architectural overview of the hemodialysis machine, reproduced from [16].

The other main component of the hemodialysis machine prepares the dialysing fluid. This is filtered, degassed and heated water, with bicarbonate and acetate suitably added. The fluid passes through one side of the balance chamber, then into the dialyser, upon return from which it passes through the other side of the balance chamber, and then to the drain. If the properties of the dialysing fluid being supplied stray outside permitted ranges at any time during treatment, a bypass circuit isolates the dialyser from the dialysing fluid flow to prevent harm to the patient's blood.

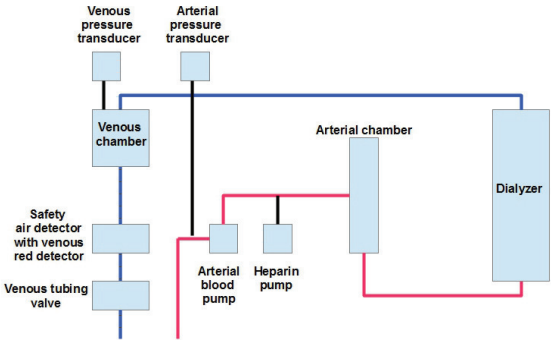

Fig. 2. Overview of the extracorporeal blood circuit and dialyser, reproduced from [16].

Both the balance chamber and dialyser are rigid structures, with a flexible membrane separating the two sides of the balance chamber. Therefore, the rate of flow of fresh dialysing fluid entering the balance chamber can be checked against the rate of flow of waste dialysing fluid exiting it (up to the latitude permitted by the flexible membrane, which has a limited freedom of movement). This being so, the rigidity of the dialyser structure then implies that the blood flow leaving the patient is strictly related to the blood flow returned to the patient (modulo the latitude permitted by the flexible membrane), this, despite the osmotic and ultrafiltration processes taking place in the dialyser which can exchange both salts and liquid between blood and dialysing fluid, the latter also altering the volume of both. This conclusion (and variations that allow for other changes, such as allowing for Heparin) is based on the assumption that both blood and dialysing fluid are *incompressible fluids*, which is true to a very high degree of accuracy.[2] Were it the case that the requirements in [16] covered such mechanical and treatment aspects of the hemodialysis machine in more detail, the insights just mentioned could be reexpressed as suitable Hybrid Event-B invariants (modulo the leeway of the flexible membrane). As noted above, this would capture a worthwhile safety property, one that was important from the patient's point of view.

3 Multi-Machine Hybrid Event-B

In this section we give a brief outline of multi-machine Hybrid Event-B for purposes of orientation. In Section 5 we cover part of the development in more detail, amplifying what we say here.

A large Hybrid Event-B system is organised syntactically in a PROJECT file. This names a number of MACHINEs and INTERFACEs. A machine is a hybrid extension of an Event-B machine (of which more shortly), while an interface declares a set of variables, their initialisations, and the invariants that involve them, so that several machines

[2] In [16] it is stated that the volumes in and out of the balance chamber are equal, but this is only true modulo the additional considerations stated here.

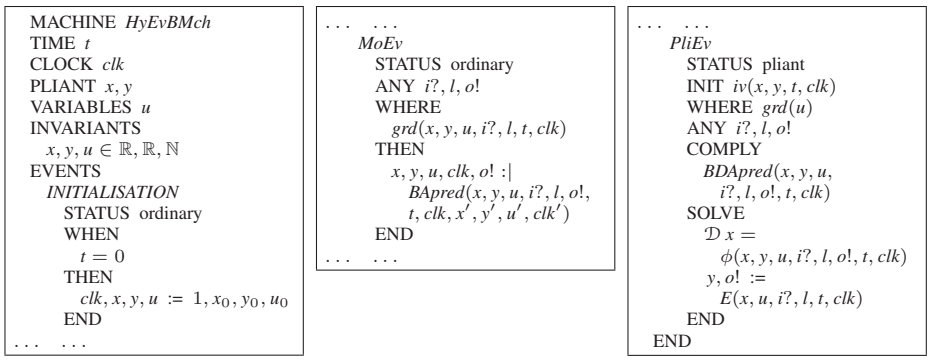

Fig. 3. A schematic Hybrid Event-B machine.

can access these variables. The project file also declares synchronisations, each of which is a set of events drawn from different machines, which must execute simultaneously at runtime. At runtime, all the machines in the project must execute concurrently, so projects must be designed in such a way that this makes sense.

Individual Hybrid Event-B machines are inspired by their Event-B predecessors. Event-B machines define discrete events that define individual changes of state, along with the supporting definitions needed to do this, and, crucially, the invariants that are claimed to hold when these events execute, thus embodying the safety properties of the system. The usual interpretation is that occurrences of the discrete events take place at isolated time points (though that is outside the actual formalism of Event-B). In a hybrid system it is natural to have discrete changes of state embodying change of 'mode', after each of which there is a period of continuously varying state change. Hybrid Event-B thus takes the individual discrete events of Event-B, naming them 'mode events', and interleaves them with periods of continuously varying state change defined in 'pliant events'. The latter may be defined in various ways: direct (continuous) assignment, (ordinary) differential equations, and implicit definition via a (time dependent) predicate expression.

Fig. 3 shows a schematic Hybrid Event-B machine. In this, the VARIABLES, INVARIANTS, *INITIALISATION* and *MoEv* are standard, as in Event-B (the latter defining discrete state change via the before-after predicate *BApred*), while TIME, CLOCK, PLIANT and the pliant event *PliEv* are new for Hybrid Event-B. The first three of these are declarations (PLIANT declaring those variables that are to be permitted to evolve contimuously). The continuous behaviour is defined in *PliEv*. Its 'STATUS pliant' proclaims it as such, and its contents describe what is needed to do this. Thus, there are two guards, *grd* concerning mode variables only, and *iv* concerning all aspects connected with initial values in continuous behaviour (when needed). The COMPLY clause is used to specify generic properties of the required behaviour (e.g. bounds for the permitted range of pliant variables, or other nondeterministic aspects), whereas the SOLVE clause is used to specify more precise behaviour. Thus the SOLVE clause contains an ordinary differential equation for the pliant variable x, i.e. $\mathcal{D}\,x = \phi(x, y, u, i?, l, o!, t, clk)$, and a direct (continuous) assignment for the pliant variable y and output $o!$, i.e. $y, o! := E(x, u, i?, l, t, clk)$.

At runtime, mode events and pliant events are required to execute alternately, in line with the intuition sketched above. With care, a semantics may be constructed that makes for a very clean interaction between the discrete and contiuous parts, and proof obligation schemas may be designed to statically ensure that the runtime behaviour is as required.

Observing that refinement in Event-B is of mode events to mode events (exclusively), in Hybrid Event-B this is preserved and extended to refinement of pliant events to pliant events (exclusively). Time is required to progress at the same rate in both abstract and concrete models. Synchronisations must be refined to synchronisations. We refer to [6] for a more detailed presentation of single machine Hybrid Event-B, and to [7] for details on the multi-machine version.

4 Hemodialysis Machine Development Generalities

Fig. 4 shows an architectural overview of the Hybrid Event-B development of the hemodialysis machine. The rectangle in the middle represents the interface *Central_IF*. This sits at the centre of the development, and holds all of the shared variables needed. Four machines connect to it: *Operator*, modelling essential elements of the operator; *Control*, modelling the control system; *BloodPump*, modelling the blood pump; and *SafetyAirDetector*, modelling the safety air detector and venous red detector. Aside from the operator and controller, the BP and SAD/VRD systems were selected for more detailed modelling in separate machines, since [16] gives just enough detail about their working to make them interesting in their own right. For the remaining subsystems in the hemodialysis machine, even if a degree of independent working might be inferred, the relatively low level of detail given in [16] suggested that more precise modelling would be guesswork, so they were represented more superficially. The practical consequence of this is that whereas, in principle, a lot more use of the capabilities of Hybrid Event-B could have been made to model the physical aspects of the system more faithfully, and hence, where appropriate, to tie these model aspects to the software control events via suitable invariants, in practice, this was not done due to the lack of the relevant low level physical detail. In fact this is a common problem in requirements targetted at software development, since overwhelmingly, software modelling formalisms are not capable of expressing, or dealing with, nontrivial physical properties.

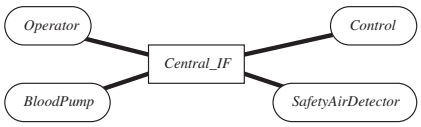

Fig. 4. Overview of the hemodialysis machine development architecture.

A further example of this is the user interface, regarding which, [16] is quite sketchy. So UI matters are represented using the environment, or recording information in suitable state variables, or simply by including the appropriate events — on the understanding that these representations could be elaborated as needed.

In the present development, it was decided to experiment with a simple component-based approach, to see how this aligned with the formal elements of Hybrid Event-B. Of course, there are many component based approaches these days, see e.g. [14, 11, 18]. The approach here was to determine the architecture of Fig. 4 at the outset, and

to then construct a development plan around it. The main issue that this threw up was squaring the desire to do the development via refinement, with the sequential nature of the description in [16]. A sequential approach to the development would complete an earlier phase before embarking on the next one, but this could easily break refinement properties, as the end of the earlier phase might have to be modified to accommodate the start of the next one. This required care in designing the top level 'umbrella' model, and in the order of introducing the other components.

5 The Development in Detail

The details of the development may be accessed at [4]. The development proceeds level by level. **Level 00** introduces the architecture of Fig. 4. There is a variable $mode \in \{NORMAL, BYPASS, ALARM\}$, and $phase \in \{PREP, INIT, TREAT, END, ENDED\}$. The *Operator* and *Control* machines move the *phase* variable through its various values, as in treatment. With an eye to refinement, a *Reset* event is introduced to enable the operator to set all system variables to arbitrary desired values when an *ALARM* occurs. In fact, the guard of the *Reset* event is weakened to TRUE, to enable its use whenever subsequent modelling is weakened due to uncertainties arising from [16].

Level 01. Level 01 introduces the blood pump, and fleshes out the *BloodPump* machine of Level 00. The Level 01 development is shown in Fig. 5, which is complete aside from a couple of small details.

We discuss Level 01 fairly fully, since it illustrates many details of multi-machine Hybrid Event-B in a small space. The PROJECT file *HemoDialysisSystemLevel_01* states that it is a refinement of the Level 00 project, and names the components, namely the *Operator*, *Control* and *BloodPump* machines, and the *Central_IF* interface, these all being at Level 01, except for *Control*, which is unchanged from Level 00.

The *Central_IF* interface refines its Level 00 counterpart. New variables relevant to the blood pump are introduced: *bpOnOff*, saying whether the blood pump is *ON* or *OFF*, and *bpMeasFlow* and *bpMeasVol*. These are respectively the measured flow rate produced by the blood pump, and the volume of blood that has passed through it. These are expected to vary continuously during operation since they are determined by the physical behaviour of the patient/machine system, so are declared pliant. The invariants now include the basic constraints upon these variables. At this level, these amount to type declarations.

There are further pump-related variables to be introduced later: the demanded pumping rate *bpRate*, and its lower and upper limits, *bpRateL* and *bpRateU*. These are data that are entered by the operator in the preparation phase using mode events (because they will not change in between occurrences of these mode events). This is covered at Level 02. The only coupling between these later data and the pump's Level 01 variables is the constraint $bpMeasFlow \leq bpRate$, which is added in a refinement from Level 01 to Level 02 of the *BloodPump* machine.[3] The remainder of the Level 01 *Central_IF* interface declares the initialisations of the variables.

[3] Obviously, the actual, measured flow rate, cannot exceed the rate that the pump is trying to achieve. To do so would contravene the laws of thermodynamics. However, in order to assert this, we must assume that the measured flow rate is *dependable*.

```
PROJECT HemoDialysisSystem_Level_01
REFINES HemoDialysisSystem_Level_00
  INTERFACE Level_01_Central_IF
  MACHINE Level_01_Operator
  MACHINE Level_00_Control
  MACHINE Level_01_BloodPump
END
```

```
INTERFACE Level_01_Central_IF
REFINES Level_00_Central_IF
VARIABLES
  mode, phase, bpOnOff
PLIANT
  bpMeasFlow, bpMeasVol
INVARIANTS
  mode ∈ {NORMAL, BYPASS, ALARM}
  phase ∈ {PREP, INIT, TREAT, END, ENDED}
  bpOnOff ∈ {ON, OFF}
  bpMeasFlow, bpMeasVol ∈ ℝ, ℝ
INITIALISATION
  BEGIN
    mode, phase, bpOnOff := BYPASS, PREP, OFF
    bpMeasFlow, bpMeasVol := 0, 0
  END
END
```

```
MACHINE Level_00_Control
CONNECTS Level_01_Central_IF
EVENTS
  PliTrue  ...  ...  ...  ...
  RaiseALARM
    STATUS asynch
    BEGIN
      mode := ALARM
    END
  PrepComplete
    STATUS asynch
    WHEN mode = BYPASS ∧ phase = PREP
    THEN phase := INIT
    END
  InitModeToNORMAL
    STATUS asynch
    WHEN mode = BYPASS ∧ phase = INIT
    THEN mode := NORMAL
    END
  InitComplete
    STATUS asynch
    WHEN mode = NORMAL ∧ phase = INIT
    THEN phase := TREAT
    END
  TreatComplete
    STATUS asynch
    WHEN mode = NORMAL ∧ phase = TREAT
    THEN phase := END
    END
  EndComplete
    STATUS asynch
    WHEN mode = NORMAL ∧ phase = END
    THEN phase := ENDED
    END
END
```

Fig. 5. Hemodialysis machine development, Level 01. Introduction of the blood pump.

```
MACHINE Level_01_Operator
REFINES Level_00_Operator
CONNECTS Level_01_Central_IF
EVENTS
  PliTrue
    STATUS pliant
    COMPLY INVARIANTS
    END
  Reset
    STATUS asynch
    BEGIN
      mode, phase, bpOnOff
      bpMeasFlow, bpMeasVol
        :| INVARIANTS
    END
  PrepStart
    STATUS asynch
    WHEN mode = BYPASS ∧ phase = PREP
    THEN skip
    END
  InitStart
    STATUS asynch
    WHEN mode = BYPASS ∧ phase = INIT
    THEN skip
    END
  TreatStart
    STATUS asynch
    WHEN mode = NORMAL ∧ phase = TREAT
    THEN skip
    END
  EndStart
    STATUS asynch
    WHEN mode = NORMAL ∧ phase = END
    THEN skip
    END
END
```

```
MACHINE Level_01_BloodPump
REFINES Level_00_BloodPump
CONNECTS Level_01_Central_IF
EVENTS
  BPStopped
    STATUS pliant
    REFINES PliTrue
    WHEN bpOnOff = OFF
    COMPLY CONST(bpMeasVol)
    SOLVE bpMeasFlow := 0
    END
  BPOn
    STATUS asynch
    WHEN bpOnOff = OFF
    THEN bpOnOff := ON
    END
  BPRunning
    STATUS pliant
    REFINES PliTrue
    WHEN bpOnOff = ON
    SOLVE 𝒟 bpMeasVol = bpMeasFlow
    END
  BPOff
    STATUS asynch
    WHEN bpOnOff = ON
    THEN bpOnOff := OFF
    END
END
```

Next, we look at the *Operator* and *Control* machines. Whereas the *Control* machine is an exact replica of the Level 00 machine, the *Operator* is a refinement of its ancestor, since the *Reset* event has to now also encompass the new blood pump variables.

Both machines have a *PliTrue* pliant event. Since any Hybrid Event-B machine runs over an extended period of time, every such machine needs at least one pliant event to be able to do so. The *PliTrue* events simply stipulate that any behaviour conforming to the invariants is acceptable. These *PliTrue* events thus embody a loose specification of the pliant behaviour of the operator and controller.

At Level 01, the operator and controller merely steer the hemodialysis machine through the phases identified earlier: $PREP, INIT, TREAT, END, ENDED$. In addition, the various modes are visible at this level: $NORMAL, BYPASS, ALARM$. For each of the phases, the operator initiates it with a *Start* mode event, and the controller terminates it with a *Complete* event, which changes the *phase* variable. Besides this, the controller can register an *ALARM*, so an event is required for this.

The observant reader will notice that, in fact, only the controller events change the phase variable, so the alternation between the operator who initiates a phase, and the controller which terminates it, is not enforced at this level of abstraction. However, at lower levels of abstraction, the controller events can have their guards strengthened to include conditions that witness the completion of intermediate activities that must be initiated by the operator. So, in order for the *Complete* events to be enabled, a collection of prerequisite events must have been triggered by the operator, enforcing the claimed sequentialisation.

All of these mode events have STATUS asynch. We explain this now. In discrete Event-B, occurrences of events, insofar as they are meant to model events in the real world, are assumed to occur at moments of time which are isolated. So 'real world' mode event execution is *lazy* (see, e.g., the many case studies in [1]).

However, because Hybrid Event-B is aimed at modelling physical behaviour, its mode events must execute *eagerly* — e.g. a ball falling onto a hard surface must bounce immediately; it does not have the option of waiting for a while. So when a mode event is enabled, it immediately preempts the ongoing pliant behaviour. Nevertheless, lazy execution of mode events is needed in modelling real systems that are capable of some spontaneous behaviour, or that respond to unpredictable stimuli from the environment. The metaphor that Hybrid Event-B provides in its semantics to handle this (see [6, 7]) is to stipulate that *inputs* to mode events arrive lazily (unless more tightly constrained in the events' guards). The 'STATUS asynch' pattern used in Fig. 5 is a syntactic shorthand introduced to replace intoducing an 'artificial' input (that would not be used in the body of the event) for each such mode event, and simply indicates that the relevant mode event is to execute lazily.

We turn to the *BloodPump* machine. At Level 00 it was specified very loosely, via a *PliTrue* pliant event alone. At Level 01 the operational details of the pump are included. There are mode events *BPOn* and *BPOff* that turn the pump on or off. Occurrences of these events interleave the pliant events *BPStopped* and *BPRunning*. These refine the Level 00 *PliTrue* specification.

BPRunning says that the derivative of the pumped blood measured volume is given by the measured blood flow rate $\mathcal{D}\, bpMeasVol = bpMeasFlow$. Note that there is no

specification of the value of *bpMeasFlow*. The earlier yet-to-be-introduced constraint $bpMeasFlow \leq bpRate$, will only amount to a loose specification of the measured blood flow rate, since the actual value of *bpMeasFlow* will depend on what takes place in the mechanical elements of the hemodialysis system, in particular on the amount of resistance they exert to pumping.

BPStopped is the analogue of *BPRunning* when the pump is not running. It stipulates that the measured flow rate *bpMeasFlow* is zero (which it must be if no flow rate is demanded), and specifies that the measured volume *bpMeasVol* remains CONSTant over the duration of the pliant event occurrence (although a differential equation could have been used). Note that this assumes that when it is switched off, the blood pump is locked in a state such that it cannot rotate backwards, which may happen at other times. If this not the case, a different specification for *BPStopped* would be needed.

Note that at Level 01, the blood pump is completely decoupled from the rest of the system (aside from the *Reset* event). The integration with other dialysis activities comes later. This remark completes Level 01.

Level 02. We treat the remaining levels of the development more briefly. Level 02 is similar to Level 01 in that it introduces the safety air detector. As for the blood pump, this is introduced in a standalone way at this stage. Thus there are mode events to turn it on and off, and pliant events to model its behaviour when on and off respectively. These behave in a way analogous to the blood pump. There is a further mode event to raise an alarm when too much air has been detected. What makes the SAD interesting to model independently, is the way that its permitted accumulated air volume varies dynamically with the current blood flow rate. In fact, this feature is only needed in the alarm event just mentioned, so that its guard has a three way test, depending on the current blood flow rate, which may be low, medium or high, corresponding to a low, medium or high permitted volume. The way that the permitted volume depends on the blood flow rate means that, in principle, it is possible for the flow rate to be high say, and for the accumulated volume to be within bounds, but then for the flow rate to decrease, crossing the boundary to the medium regime, and for the same accumulated volume to suddenly be outside the new permitted bounds and thus to cause an alarm. Whatever the merits of this possibility, it has been modelled faithfully in order to test the modelling capabilities of Hybrid Event-B. The SAD also contains the venous red detector. The only feature of this is to execute a mode event when red appears or disappears.

Level 03. Level 03 deals with the first phase of dialysis, namely the preparation of the machine. In principle, this consists of rinsing and filling the machine, and of entering a large amount of data concerned with the parameters of the desired treatment regime. The preparation phase starts with the self test. If this fails, the alarm is sounded, and it is presumed that the operator takes over. Thus the success of the self test does not figure as a guard in subsequent steps (beyond the first).

Manipulation of various items of equipment is modelled by async mode events in the *Operator* machine, following the flow of [16]. Since the rinsing of the tubing is described in a little more detail, more detailed modelling of this is attempted. The main point of interest is that the blood pump is used, necessitating the coupling of the *Operator* machine, the *Control* machine, and the *BloodPump* machine. The required synchronisations are declared in the Level 02 PROJECT file thus:

SYNCH(*PrepRinsingTestingTubingStart*)
Level_02_Operator.PrepRinsingTestingTubingStart_S
Level_02_Control.PrepRinsingTestingTubingStart_S
Level_01_BloodPump.BPOn_S
END

SYNCH(*PrepRinsingTestingTubingEnd*)
Level_02_Operator.PrepRinsingTestingTubingEnd_S
Level_02_Control.PrepRinsingTestingTubingEnd_S
Level_01_BloodPump.BPOff_S
END

Both of these state that collections of events in different machines, each event referred to in the form *MachineName.EventName_S* (where the '*_S*' suffix is a decoration for additional readability), must execute together. In *PrepRinsingTestingTubingStart*, the *Operator* machine takes the initiative as the operator starts the tubing test process. Simultaneously, the *Control* machine records the change of state, and the *BloodPump* machine starts to operate. When the required amount of pumping has taken place, in *PrepRinsingTestingTubingEnd*, the *Control* machine is preempted by the pumped volume reaching a value that triggers the guard of the *PrepRinsingTestingTubingEnd_S* event, and the synchronisation then stops the pump and alerts the operator.

Aside from this, a lot of data entry is needed, modelled by straightforward state machine controlled input techniques. One data entry task is modelled faithfully (the entry of the filling blood pump rate), and the remainder are merely indicated.

Other mechanical manipulations of the equipment are indicated more briefly in [16], and are modelled more superficially. The last of them is the rinsing of the dialyzer. The completion of this causes the transition from the preparation to the initiation phase.

The preparation phase is characterised by a couple of safety requirements in [16], namely **R-18** and **R-19** (concentrate mixup), and **R-20** (temperature alarm in heating/degassing). These fall into the first of the requirement patterns discussed more extensively in the context of Level 04 below. The concentrate mixup alarm has been modelled by an alarm covering both cases, but since so little has been included in [16] about heating/degassing, the latter has not been modelled.

Level 04. The initiation phase is modelled in the Level 04 project. Arterial connection is assumed completed via operator events, and then the filling of the blood tubing is achieved by events synchronised between the operator, control and blood pump, just as discussed above. This time the pumping process is stopped via the venous red detector. Change of phase and change of mode are handled by existing events.

The initiation/connection phase is characterised by a large number of safety requirements in [16]. There are eleven general requirements **S-1** to **S-11**. Especially given the lack of mechanical details regarding the hemodialysis machine, these appear to be requirements on the behaviour of the operator, thus outside the remit of the developed system. Further, there are software requirements **R-1** to **R-36**. These appear to be amenable to incorporation into the developed system, but are hampered by the lack of detail mentioned. Thus their treatment must be, to some extent, hypothetical.

The requirements are of two types. The first concerns 'pointlike' conditions. A physical quantity, presumed monitored in a continuous way within the machine,[4] crosses some threshold, and some response is required (usually raising an alarm). Such requirements are easy to model. Thus, provided there are enough variables whose state is

[4] In reality, such a variable will be monitored discretely, with a short sampling period. But in Hybrid Event-B terms it would correspond to a pliant variable, varying piecewise smoothly.

sufficient to define the period during which the condition is to be watched for, this state, plus the crossing of the threshold, constitute the guard of a 'raise alarm' mode event.

The second concerns 'extended' conditions. A physical quantity, again presumed monitored in a continuous way, crosses an undesired threshold and remains there too long (or until an extensive quantity has grown too large), at which point a response is required (again usually an alarm). A slightly different model is required here. Upon the threshold being crossed, a mode event as in the previous case starts a clock (or initialises the relevant extensive variable). Two further mode events complete the modelling. Since the physical quantity is assumed to vary in a *continuous* manner (a crucial assumption), it either stays on the undesired side of the threshold or it recrosses the threshold in the opposite direction. In the latter case, if it happens soon enough, the recrossing will trigger a recrossing mode event and the clock can be disabled. In the former case the clock will reach a trigger value and an alarm can be raised. Of course, once the alarm has been raised, any subsequent recrossing is of no interest.

Regarding the initiation/connection phase, **R-2** to **R-4** provide examples of both kinds of requirement. They have been modelled faithfully by the patterns just described since the blood pump has been modelled in reasonable detail.[5] Of the rest, **R-5** to **R-11** and **R-14** to **R-17** also concern initiation/connection, involve the blood pump, and fit the patterns described. Of these, **R-11** is modelled (though not the switching off of the blood pump, since it is unclear from the various parts of [16] when exactly the pump is to be switched on or off during initiation/connection). Given the lack of precision around a number of details, and the evident similarity of this set of requirements to one or other of the two patterns described, the remainder of **R-5** to **R-11** and **R-14** to **R-17** were not modelled explicitly. Requirements **R-18** to **R-21** have been mentioned already.

There are some requirements that concern the SAD in the initiation/connection phase: **R-23** to **R-32**. Of these **R-24** to **R-26** are part of its operational definition, and have been modelled directly in the safety air detector machine. The remainder specify, in the various phases of the hemodialysis procedure, that the flow, or the accumulated air, remains within permitted margins (as discussed above). Notably, among the phases mentioned, the treatment phase appears to be absent. It seems strange that there is no objection to pumping the patient full of air during treatment, whereas it is prohibited during all other phases of the hemodialysis process. Accordingly, in the present model, monitoring of the accumulated air remains enabled throughout hemodialysis, which also simplifies the modelling a little. In this manner, all of **R-23** to **R-32** are covered.

Level 05. The treatment phase is modelled in the Level 05 project. In general, is is not completely clear which of the very many parameters entered previously are active at the commencement of treatment, so this aspect was not reflected accurately in the modelling. Besides this, a number of activities take place during the treatment phase that raise questions regarding modelling. We discuss these one by one.

Venous return flow pressure. The description in [16] speaks of a dynamic limits window around the current value of the pressure, but leaves many issues uncertain. How is the dynamic limits window calculated from the current value? If it is based around the current value only, it is impossible for the current value to cross it, except when the

[5] Since the blood pump rate is continuous, the rate *must* drop below 70% of desired (**R-3**) *before* it can go into reverse (**R-4**), so one might conclude from the text of [16] that **R-4** is redundant.

machine's hard limits are crossed, so what is its purpose? If it is *not* based solely on the current value, how is it determined? What is supposed to happen when the current pressure crosses the boundary of either the dynamic limits window (assuming this is mathematically possible) or the machine's hard limits window? Owing to the lack of clarity on these points —to which it is possible to imagine many possible resolutions— modelling of the venous return flow pressure was not attempted. However, it seems apparent that whatever the truth surrounding these matters, modelling based on the same kinds of idea used in the modelling of the blood pump and of the safety air detector, would be sufficient to handle this aspect of the machine.

Arterial entry pressure. The description in [16] is much briefer here, but suggests many of the same uncertainties as in the previous case. Again, due to the large degree of doubt, modelling was not attempted.

Blood-side entry pressure at the dialyser. Again, the description in [16] raises many questions about precisely what is monitored, what is under control, and what controller actions are needed in response to what measured stimuli from the equipment. Again, this was not modelled due to the large amount of uncertainty.

Treatment at minimum UF rate. This is modelled with a simple operator command to set the UF rate.

Heparin bolus. This is modelled by an operator command to start the bolus, and a controller response when it finishes. The present study does not model the Heparin pump explicitly, due to lack of detail, but one could presume that it might work like the blood pump. In connection with this, **R-22** could be modelled like **R-4**.

Arterial bolus. In the case of the arterial (saline) bolus, a little more detail is given. The modelling is like for the Heparin bolus, but the blood pump is used. So the start and stop events are synchronised with pump starting and stopping, the latter of which is conditional on the pumped volume.

Interrupting dialysis. Dialysis can be interrupted by going to bypass mode. This has not been modelled in order to illustrate a specific point. The *mode* variable is defined at Level 00, in order to help structure the whole development. To conform with (Hybrid) Event-B refinement practices, all the relevant events involving a change in *mode* should be defined at that level. Mostly they are apparent from a high level reading of [16]. However, here, there is another *mode* transition, buried deep in the detail. It could be accommodated easily enough by reworking from Level 00 onwards, to incorporate the needed transition. But doing this would be increasingly cumbersome the larger the development, so it was not done. This raises an interesting point regarding components and refinement, which we ponder now.

In defining a family of components, we can start by writing down the components in a 'hollow' form, writing down their coordination, and leave the internal details of their working to be refined in later. Alternatively, we can start by writing the components in an independent uncoordinated 'soup', then work bottom up, filling in their details, until their coordination needs to be addressed. The first option risks not accounting for all the needed coordinations early enough (as we have seen). The second option leaves all coordination in doubt until a late point, which may put high level requirements at risk. Further discussion of such situations, including a convenient remedy in the context of the Event-B refinement approach, appears at the end of this paper.

Treatment end. An acoustic sound is modelled. However, the lamp(s) are ignored. It is not clear how many lamps there are; nor whether they are real lamps or part of a synthetic UI display; nor, if they are real, what they do when they are not displaying the colour(s) indicated in [16] at the specific moments mentioned in the description.

From the above account, it is clear that only a portion of the whole treatment regime has been captured. Nevertheless, it seems apparent that if more detail were included in [16], it would not be difficult to model it using the patterns used at various places earlier.

Level 06. Level 06 models the therapy ending phase. The last stages of this, namely emptying the dialyser/cartridge and overview seem relatively automatic, in that they simply take place in sequence. Since few details about them are provided in [16], they are modelled with a single generic command from the operator. The preceding step, reinfusing the patient, is described in a little more detail, so is modelled using the operator, control and blood pump machines. The general approach resembles similar activities earlier in the hemodialysis process, that is to say, the operator starts a portion of the reinfusion, and this is stopped when one of the terminating conditions is reached, i.e. time elapsed, volume pumped, or red detected.

Some software requirements remain unaddressed: **R-33** to **R-36**. These concern the dialyser, and fluid removal, etc. Again, there is a dearth of needed detail in [16]. It is not at all clear what components are present in the dialyser, which of them connect to the software controller, how they operate, and how this processing sits in the rest of the hemodialysis procedure. So no serious attempt was made to model these requirements. Still, if we were to conjecture on the basis of what might be guessed from [16], we may suppose that the operation might be modelled in ways similar to subsystems we have covered in detail earlier.

6 Invariants, Verification

It is very clear when considering the hemodialysis machine system, that interaction with the environment is a vital and inescapable ingredient of operating the machine. In this case the environment comprises the patient, the operator, and includes unpredictable (and undesired) facets of machine operation. There is certainly a 'desirable, default' sequence of steps to hemodialysis treatment that one expects to be the norm. But it is clear that it is not within the power of the machine's controller to *enforce* this sequence during any particular occurrence of treatment. For example, it is *desirable* that the blood pumping rate remains close to the rate specified, but the system has to cope with the possibility that it doesn't (usually by raising an alarm, and throwing the responsibility back on the operator). There are a host of similar issues noted in [16].

The preceding impacts what can be achieved using verification. Normally, a system will be characterised by a number of properties linked together during operation. The interdependencies are usually expressible via invariants, and the B-Method gains its strength by insisting on the proof of the invariants. But when almost any variation in behaviour has to be anticipated, the prospect for writing invariants that are actually provable, diminishes severely. About all that remains, is the essentially sequential structure, insisting that one step precedes another.

In the case of simple sequential precedence, the invariants that arise, do so in the form 'if $u = A$ guards an event that achieves $v' = B$, then $v = B \Rightarrow u = A$ is an invariant'. This is no more than the action/reaction pattern noted by Abrial in [1]; obviously there are variations on this in less elementary cases. Such invariants can be extracted mechanically for sequential systems, so the role of the human designer is reduced in that there are many fewer informative invariants for him to infer, in a system in which variables are allowed to misbehave in arbitrary ways. The human may test his diligence by writing the invariants and checking whether he anticipates everything that the invariant generator is able to infer, but equally, he may just get bored by doing so. In line with this observation (and noting again that Hybrid Event-B is currently not machine processable), in the present modelling exercise, these routine invariants have not been written down.[6]

Thus, what can be achieved by verification of invariants (the mainstay of the B-Method) is severely limited in cases where so much comes from the environment. The methodology is much stronger in the case of closed systems, where the robustness has to come from internal design consistency, rather than ceding responsibility to an omniscient operator. And so, to regain some of this closed system robustness, we would have to depend on more detailed knowledge of the procedures engaged in by an experienced hemodialysis nurse, so that we could couple the *Controller* and *Operator* machines in a more exacting manner.

7 Conclusions

In the previous sections, we overviewed the hemodialysis case study, and after a brief review of Hybrid Event-B, described the Hybrid Event-B treatment of the case study. Our approach was structured round a pure refinement based strategy, based on an early recognition that the essential elements of the case study were sufficiently straightforward that the treatment of faults could be included in the refinement based strategy relatively straightforwardly, i.e. without the faults derailing the strategy to an unconvincing degree. This aspects is to be contrasted with the landing gear case study treated in [3, 5], where (and especially in [5]) great benefit was derived from focusing first on the nominal development, and then incorporating the faulty regime, using retrenchment [10, 9, 8]. There, the various levels of fault tolerance would have made a flat treatment unhelpfully complex. Here though, the basic strategy of throwing responsibility back on the operator when things go wrong, made things considerably simpler, but undermined what could be accomplished using invariants, as discussed in the preceding section. Additionally, the earlier experience in modelling a complex application coming from the landing gear case study, made the process go a lot more smoothly.

As we noted in many places above, in the requirements [16], many issues regarding the dialysis treatment procedure were mentioned, without enough detail being given regarding how they might be implemented. Also, many aspects of the machine's working were alluded to without enough precision being given to make clear the degree of software involvement. In view of this, a fairly conservative approach was taken in

[6] Aside from the routine nature of the invariants, is the fact that the *actual* degree of required sequentiality in hemodialysis is not clearly delineated in [16].

the present development. What was modelled, was what related fairly directly to machine elements discussed with adequate precision in the text. Aspects that were less well described, involving features that were perhaps only hinted at, were typically not modelled, though often, it seemed that had they been better described, they would fit patterns already utilised elsewhere in the development. Since one of the main objectives of this study was to test the modelling adequacy of Hybrid Event-B in the face of challenging applications, the approach taken seemed reasonable.

On the other hand, the focus on a relatively narrow portion of the hemodialysis spectrum, loses the chance to define richer machine properties, and to connect them via suitable invariants to wider issues, whether in the machine or in the treatment regime. It is to be hoped that the introduction of more expressive formalisms such as Hybrid Event-B could contribute to more broadly based and more robust problem definitions.

In the author's opinion, a major, and worthwhile, outcome of the present work has been the opportunity to experiment with a simple component based approach to Hybrid Event-B development that was described earlier. Although it demands some forethought at the beginning, to decide on an appropriate level of modelling for the embryonic form of the various components that constitute the system-to-be, the clarity about the system architecture that it brings right at the outset, is very beneficial. Nevertheless, it demands some experience with refinement to judge how these components may be defined at the top level in order to avoid extensive rework later. Too much of the wrong sort of detail too early in the development can conflict with the rigours of refinement, when lower level detail emerges that is in conflict with more simplistic structures set down earlier. We saw an example of this in the discussion of bypass mode during the treatment phase. There we commented on the pros and cons of trying to define just enough of the coordination structure between the components early enough to provide a clear picture at a high level of abstraction, versus the 'soup of components' approach in which no attempt to define this especially early was made, at the cost of leaving coordination issues completely unaddressed till quite late on.

One convenient way of dealing with the situation is via retrenchment [10, 9, 8]. This, in its most benign form, would permit the introduction of 'new' events that modified 'old' variables, that preserved all invariants, but that were nonetheless *not* refinements of skip. Such events would evidently not upset anything that was deduced on the basis of the invariants —in our syntactic organisation of refinements of projects, all relevant invariants could be located by ascending the refinement hierarchy of interfaces until all needed variable declarations had been found— but they could upset conclusions made at a more abstract level based on model checking, since they can properly enlarge the reachability relation of the system at that level of abstraction. Postponing model checking till all such retrenchments had been completed would be an evident way of circumventing this difficulty. (Although clearly available here, the opportunity to take advantage of this technical possibility was not exploited in the present work, just to illustrate the point.) The approach described would provide the opportunity to address major coordination concerns between the components at a high level of abstration, while leaving the possibility of fine-tuning them later in the light of further detail that emerged at a lower level. With this proviso, the simple component based approach pursued here can be recommended as a useful methodological approach.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
2. Ahmad, S.: Manual of Clinical Dialysis. Springer (2009)
3. Banach, R.: Landing Gear System Case Study in Hybrid Event-B. In: Boniol, Weils, Ait Ameur, Schewe (ed.) Proc. ABZ-14. vol. 433, pp. 126–141. Springer, CCIS (2014)
4. Banach, R.: Hemodialysis Case Study in Hybrid Event-B Web Site (2015), http://www.cs.man.ac.uk/~banach/some.pubs/ABZ2016HemodialysisCaseStudy/
5. Banach, R.: The Landing Gear System in Multi-Machine Hybrid Event-B. Int. J. Soft. Tools for Tech. Trans. (2015), to appear.
6. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core Hybrid Event-B I: Single Hybrid Event-B Machines. Sci. Comp. Prog. 105, 92–123 (2015)
7. Banach, R., Butler, M., Qin, S., Zhu, H.: Core Hybrid Event-B II: Multiple Cooperating Hybrid Event-B Machines (2015), submitted.
8. Banach, R., Jeske, C.: Retrenchment and Refinement Interworking: the Tower Theorems. Math. Struct. Comp. Sci. (2015)
9. Banach, R., Jeske, C., Poppleton, M.: Composition Mechanisms for Retrenchment. J. Log. Alg. Prog. 75, 209–229 (2008)
10. Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Engineering and Theoretical Underpinnings of Retrenchment. Sci. Comp. Prog. 67, 301–329 (2007)
11. Crnkovic, I., Larsson, M.: Building Reliable Component-based Software Systems. Artech House (2002)
12. Daugirdas, J., Blake, P., Ing, T.: Handbook of Dialysis. Wolters Kluwer (2007)
13. Harris, D., Elder, G., Kairaitis, G., Rangan, G.: Basic Clinical Dialysis. McGraw Hill (2005)
14. Heineman, G., Councill, W.: Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley (2001)
15. Kallenbach, J., Gutch, C., Stoner, M., Corea, A.: Review of Hemodialysis for Nurses and Dialysis Personnel. Elsevier Mosby (2005)
16. Mashkoor, A.: The Hemodialysis Machine Case Study. In: Proc. ABZ-16. Springer, LNCS (2016), these proceedings.
17. Nissenson, A., Fine, R.: Handbook of Dialysis Therapy. Saunders Elsevier (2008)
18. Somaia, Z.: Component-Based Software Development. Lambert Academic Publishing (2014)