Safety analysis of software components of a dialysis machine using model checking

M.D. Harrison[1,4] (orcid.org/0000-0002-5567-9650), M. Drinnan[2], J. C. Campos[3,4], P. Masci[3,4], L. Freitas[1], C. di Maria[2], and M. Whitaker[2]

[1]School of Computing Science, Newcastle University, Newcastle upon Tyne NE1 7RU, UK,
[2]Regional Medical Physics Department, Royal Victoria Infirmary, Newcastle upon Tyne, NE1 4LP
[3]Dep. Informática / Universidade do Minho, Braga, Portugal
[4]HASLab / INESC TEC, Braga, Portugal

Abstract. The paper describes the practical use of a model checking technique to contribute to the risk analysis of a new paediatric dialysis machine. The formal analysis focuses on one component of the system, namely the table-driven software controller which drives the dialysis cycle and deals with error management. The analysis provided evidence of the verification of risk control measures relating to the software component. The paper describes the productive dialogue between the developers of the device, who had no experience or knowledge of formal methods, and an analyst who had experience of using the formal analysis tools. There were two aspects to this dialogue. The first concerned the translation of safety requirements so that they preserved the meaning of the requirement. The second involved understanding the relationship between the software component under analysis and the broader concern of the system as a whole. The paper focuses on the process, highlighting how the team recognised the advantages over a more traditional testing approach.

Keywords: Risk analysis, formal methods, model checking, medical devices, haemodialysis

1 Introduction

The risk analysis required to satisfy regulatory requirements (for example [4, 16]) includes an assessment of the hazards associated with a medical device. These hazards include possible hardware and software failures. Examples of hardware failure include, for example, faulty connections or pump failure. Risk analysis, as part of a submission for certification by a regulator, is an onerous task typically requiring substantial amounts of test data. Developing such a submission is essential when dealing with life-critical medical systems. Medical device standards (see for example [4]) require that measures have been taken to ensure that risks associated with use of the device are as low as reasonably practical. The required measures include careful identification of hazards and demonstration that risks

Post-print of the paper published in Formal Aspects of Component Software (FACS 2017), volume 10487 of Lecture Notes in Computer Science, pages 137-154. Springer, 2017. The final publication is available at Springer via: http://dx.doi.org/10.1007/978-3-319-68034-7_8.

associated with these hazards have been mitigated. One part of demonstrating that hazards have been mitigated is to establish requirements of the system that demonstrate that there are barriers between a hazard and its consequence. Processes that are recommended to achieve such confidence include team based scrutiny of the use of documented processes as well as testing that requirements have been satisfied.

This paper describes part of a safety analysis process. It describes how model checking analyses were used to demonstrate that a particular software component within the system satisfied requirements described in a risk log. The focus of the paper is to look at the process and to discuss how the team used the model checking analysis to consider the broader safety requirements of the system. The formal analysis that was generated as a result of the process described in the paper was submitted as evidence to the regulator.

2 The NIDUS device

Dialysis and ultrafiltration (removal of excess water) are extremely difficult procedures in small children with failing kidneys because the total volume of blood in the child's circulation is very small. The Newcastle experimental Infant Dialysis and Ultrafiltration System (NIDUS) has been used at Newcastle-upon-Tyne's Royal Victoria Infirmary (RVI) for some time. It does not use a traditional dialysis circuit, and the circuit volume of about 10 mL is suitable for treating infants with a total blood volume less than 100 mL. Before the device could be used more widely, it was necessary to identify and assess the risks of using it before the device could achieve regulatory approval.

The device is implemented using several software components. These include device drivers (for example, the motors that control the infusion pumps), components that enable the device to recognise and manage system failures (for example, the presence of bubbles in tubing) and components that provide the interface to allow the operator of the device to be aware of its status and to control the system. The component under consideration manages the drivers. Its logic of operation is organised as a control table. The table describes two aspects of the controller. It describes the attributes of the state of the device that control the dialysis process and it describes how the state of the device changes in response to events. Hence in Figure 1, *RST_InitS1* (identified in the left hand column) is a state that has attributes *Power*, *Motor1*, *Motor2* etc. (top row) with values *ALLOW12V*, *M1FWDMAX*, *M2SAFE* etc. (as described in the row labelled *RST_InitS1*). At the same time the right hand side of the table describes transitions. Hence, for example an *M1Stall* event (as identified in the top row) causes a transition from the *RST_InitS1* state to the *RST_InitS2* state.

* NAME	Power	Motor1	Motor2	Hep	Peri	Valve	Alarm	WashTimer	DialysisTimer	Flash	Mode	HardFault	Overpressure	Bubble	AgreeAir	12Voff	M1stall	
T_PowerOn	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	UNLATCH	INHIBIT	ZERO	ZERO	DISABLE	RESET	HardFault						
T_ColdStart	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	UNLATCH	INHIBIT	ZERO	HOLD	ENABLE	RESET	HardFault	ST_ColdStart	ST_ColdStart				
T_WarmStart	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	UNLATCH	INHIBIT	ZERO	HOLD	ENABLE	WASH	HardFault						
RESET stuff																		
ST_Start	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		RST_Ready		
ST_Ready	ALLOW12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors				
ST_InitS1	ALLOW12V	M1FWDMAX	M2SAFE	HEPSAFE	PERISAFE	PREP	ACTIVE	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		ST_ColdStart	RST_InitS2	
ST_InitS2	ALLOW12V	M1STOP	M2FWDMAX	HEPSAFE	PERISAFE	FLUSH	ACTIVE	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		ST_ColdStart		RST
ST_InitHep	ALLOW12V	M1STOP	M2STOP	HEPBCKMAX	PERISAFE	FLUSH	ACTIVE	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		ST_ColdStart		
ST_Relax	ALLOW12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	FLUSH	ACTIVE	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		ST_ColdStart		
ST_All	ALLOW12V	M1RESET	M2RESET	HEPRESET	PERISAFE	VALVESAFE	ACTIVE	ZERO	HOLD	ENABLE	RESET	HardFault	RST_Errors	RST_Errors		ST_ColdStart		
RESET errors																		
ST_Errors	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	VALVESAFE	WARN	HOLD	HOLD	ENABLE	RESET	HardFault	RST_Overpressu	RST_Bubble				
ST_Overpressure	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	UNLATCH	WARN	HOLD	HOLD	VALVE	RESET	HardFault	RST_Overpressure					
ST_AckOverpressure	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	VALVESAFE	WARN	HOLD	HOLD	ENABLE	RESET	HardFault	RST_Overpressure					
ST_Bubble	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	UNLATCH	WARN	HOLD	HOLD	ENABLE	RESET	HardFault		RST_Bubble				
ST_AckBubble	TRIP12V	M1UNSTALL	M2UNSTALL	HEPUNSTALL	PERIUNSTALL	VALVESAFE	WARN	HOLD	HOLD	ENABLE	RESET	HardFault		RST_Bubble				
Prime heparin																		
EP_Prime	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	FLUSH	NOTIFY	ZERO	HOLD	HEPSYRINGE	WASH	HardFault						
EP_Warn	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	UNLATCH	NOTIFY	ZERO	HOLD	ENABLE	WASH	HardFault						
WASH from fresh kit																		
VA_Start	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	ZERO	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Ready	WA_Errors	WA
VA_Ready	ALLOW12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	HOLD	HOLD	WASTEOPEN	WASHING	HardFault	WA_Errors	WA_Errors			WA_Errors	WA
WASH after any other activity																		
VA_Restart	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Decision	WA_Errors	WA
VA_Decision	TRIP12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	QUIET	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors			WA_Errors	WA
VA_Incomplete	ALLOW12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	WARN	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors			WA_Errors	WA
VA_NotReady	ALLOW12V	M1SAFE	M2SAFE	HEPSAFE	PERISAFE	VALVESAFE	WARN	HOLD	HOLD	WASTEOPEN	WASHING	HardFault	WA_Errors	WA_Errors			WA_Errors	WA
WASH start prime cycle																		
VA_S1Prime	ALLOW12V	M1BCKMAX	M2STOP	HEPSAFE	PERIPERFUSE	PREP	ACTIVE	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
VA_S1Flush	ALLOW12V	M1FWDMAX	M2STOP	HEPSAFE	PERIPERFUSE	PREP	ACTIVE	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
WASH start washing cycles																		
VA_S1Fill	ALLOW12V	M1BCKMAX	M2STOP	HEPSAFE	PERIPERFUSE	PREP	ACTIVE	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
VA_FillRlx	ALLOW12V	M1STOP	M2STOP	HEPSAFE	PERIPERFUSE	PREP	ACTIVE	HOLD	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
VA_S1S2	ALLOW12V	M1FWDMAX	M2BCKMAX	HEPSAFE	PERIPERFUSE	DIAL	ACTIVE	TICK	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
VA_FlushRlx	ALLOW12V	M1STOP	M2STOP	HEPSAFE	PERIPERFUSE	DIAL	ACTIVE	TICK	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA
VA_S2Flush	ALLOW12V	M1STOP	M2FWDMAX	HEPSAFE	PERIPERFUSE	FLUSH	ACTIVE	TICK	HOLD	ENABLE	WASHING	HardFault	WA_Errors	WA_Errors		WA_Stop	WA_Errors	WA

Fig. 1. A fragment of the control table

The controller involves 93 states and 30 events and the spreadsheet has been generated directly from the data structure that drives it. Each state attribute describes an attribute of the behaviour of the hardware system, for example:

- Attributes *Motor1*, *Motor2* and *Hep* describe the proximal, distal and heparin syringes respectively. Values of the attributes include whether the syringe pump is driving forward or backward, and whether "fast" or "slow".
- *Valve* and *Bubble* describe the valve assembly and the bubble detectors. The valve may, for example, be safe or open to the baby.
- *Flash* and *Alarm* describe features of the user interface. For example Flash shows, amongst other displays, that a clip is open or closed, and the Alarm can warn or notify or be quiet.

3 Verification Approach

The risk assessment for the device requires a multiplicity of evidence. These sources include clinical trials, software and hardware test results and documentation of the development process. The problem with trials and test results is that they do not guarantee the absence of problems. Our goal was to use formal analysis as one source of evidence that all risks have been assessed. This was made easier in the present case because the spreadsheet was an encoding of the data structure used to drive the controller. The analysis of the controller involved the following steps.

1. The developers created a risk log which described informal system safety requirements designed to mitigate hazards (Section 4). These covered the whole range of hardware and software hazards.
2. The state transition table was translated into a behavioural model that could be analysed using a model checker (Section 5).
3. Requirements, derived from the risk log, were then considered and those that related to the controller were expressed in a formal logic (Section 6).
4. These risk related properties were checked against the model that had been derived from the transition table. Where they failed, further discussion with the developers indicated either a flaw in the controller, or a situation which was considered either not to be hazardous, or to be a failure in the formulation of the property. The process of checking the properties, based on the requirements, often resulted in refinement of the properties or modifications to the control table or the controller mechanism. The results of this process were documented in the risk log (Section 7).

The process of property formulation and discussion was a significant element in the risk analysis of the controller.

4 Translating the risk log into requirements

The risk log (see Figure 2) formed the basis for the risk assessment. It described the requirements that were considered to mitigate risks and linked the

requirements to the evidence that they were satisfied. It was this document that provided material relating to the software controller component that was the source of the dialogue between developers and the formal analyst. Regulatory authorities typically require that risk control measures are included as requirements (see BS EN 62304:2006 [4] for example). Each control measure should be verified and the verification documented. As already stated, verification is typically taken to mean that some form of systematic testing has taken place. The BS EN 62304 standard requires a risk analysis path: "from hazardous situation to the software item; from the software item to the software cause; from the software cause to the risk control measure; to the verification of the risk control measure". The process of proving regulatory requirements has been discussed in more detail in [14].

Some of the requirements in the risk log were either completely or partially relevant to the software controller and these provided the basis for the analysis. Converting a software controller requirement into a property of the model involved discussion between the developers and the analyst. When a formulated property failed, examples of the failure were presented to the development team. In Figure 2 MAL.GENERROR is highlighted in red because the property that describes this requirement, at the particular stage of the risk assessment process at which the log was current, was false. As will be illustrated, this property was further refined and the red highlighting removed. Note that the spreadsheet in Figure 1 shows similar highlighting indicating changes to the control table that arose and thus required an iteration of the analysis.

In this way requirements were refined iteratively, involving the whole team. Our aim was to formulate a requirement of the controller as a property of the model and if the property failed explain why it failed. Failure of a property could mean that: (i) the model did not capture the functionality of the device; (ii) the property was not correctly formulated; (iii) there was an issue in the design that could either be dealt with in another way, for example hardware or through some mitigating process. An additional mitigating factor might be, for example, a requirement on the clinician to strictly adhere to an operating procedure. This analysis process was documented to provide evidence that all reasonable measures had been taken to ensure the safety of the device. This encouraged the developers to be confident that the device was safe as well as indicating small design changes to improve safety. When treating premature and sick infants, proper *in vivo* testing is almost impossible and the consequences of failure can be very serious. This formal process has proved invaluable. An example of a requirement in the risk log described in Figure 2, and used as illustration in Section 6, is:

> "During DIALYSIS, when the digital syringe is moving forwards then the proximal syringe is necessarily moving backwards."

The developer produced a partial translation of this in discussion with the analyst. The formulation indicates the logic without noting the temporal dimension of the property or the precise nature of the sets $\{M2Fwd\}$ and $\{M1Bck\}$.

If M2 in {M2Fwd} $\rightarrow$ M1 in {M1Bck}

Ref	Requirement	
MAL.GENINHIBIT	The alarm is only inhibited during the RESET phase.	It is always the case that wh
MAL.GENBABY	The BABY valve can only be open while the system is in DIALYSIS mode.	It is always the case that wh
MAL.GENS1MOVE	During access to the baby, the BABY valve is open.	It is always the case that wh M1 IN { M1Withdraw, M1Re
MAL.GENS2STOP	During access to the baby, the distal syringe is never running.	It is always the case that wh M1 IN { M1Withdraw, M1Re
MAL.GENERROR	For all error conditions and all system states, the next state will be an ERROR state.	For all error conditions and Note this condition logically all errors have been cleared IF ErrorCondition THEN Nex
MAL.GENS2S1	During DIALYSIS, when the distal syringe is moving forwards then the proximal syringe is necessarily moving backwards.	It is always the case that wh IF M2 IN { M2Fwd } -> M1 II
MAL.GENS1S2	During DIALYSIS, when the distal syringe is moving backwards then the proximal syringe is necessarily moving forwards.	It is always the case that wh IF M2 IN { M2Bck } -> M1 IN

Fig. 2. Risk Log in development

5 Translating the state transition table into a formal model

5.1 The specification language

The simple state transition table used to drive the controller readily lends itself to a mechanical process. We used the IVY tool [5] because it was readily available to us. It provides a front end to the NuSMV model checker [6]. IVY supports an action orientated logic language MAL (Modal Action Logic) that provides a textual structure similar to the diagrammatic structure of tools such as SRC [10]. A reason for using this particular toolset was that we were interested in the possibility of producing a tool that would be more easily understandable to an interdisciplinary team. Our goal is that IVY be used eventually without formal methods expertise. The intention was that the tool should provide a key element in communication within the team while at the same time providing the evidence that a requirement under analysis was satisfied.

MAL enables the easy description of state machines such as the table that drives the dialysis machine. Attributes are used to capture the information present in the state of the device and actions transform these states. MAL describes a logic of actions and is used to write production rules that describe the effect of actions on the state of the device. This style of specification was found easy to use by software engineers [15]. For this reason MAL was preferred to the notation which is used by the NuSMV model checker. The language also enables the expression of deontic operations, in particular permissions were used in our analysis. Non-determinism is possible when more than one action is allowed in the same state of the described model. MAL rules are a convenient way to describe the behaviour of the state table that drives the dialyser. The logic provides:

– a modal operator $[\_]\_ : [ac]expr$ is the value of $expr$ after the occurrence of action ac — the modal operator is used to define the effect of actions;

- a special reference event []: []*expr* is the value of *expr* in the initial state(s) — the reference event is used to define the initial state(s);
- a deontic operator *per*: *per*(*ac*) meaning action *ac* is permitted to happen next — to control when actions might happen;
- a deontic operator *obl*: *obl*(*ac*) meaning action *ac* is obliged to happen some time in the future. Note that *obl* was not used in this analysis.

The notation also supports the usual propositional operators. As an illustration, the following example declares two boolean attributes that describe whether the device is on (*poweredon*), whether it is dialysing (*dialysingstate*) and two actions (*start* and *pause*). It describes the effect of the action *pause* as setting the attribute *dialysingstate* to false and leaving the attribute *poweredon* unchanged. Priming is used to identify the value of the attribute after the action takes place. A permission predicate restricts the *pause* action to only happen when the system is dialysing and powered on. The *keep* function preserves the value of the attribute *poweredon* in the next state. If an attribute is not modified explicitly or is not in the *keep* list, then its value in the next state is left unconstrained.

interactor *dialyser*
attributes
poweredon, *dialysingstate* : *boolean*
actions
start pause
axioms
[*pause*] !*dialysingstate′* & keep(*poweredon*)
per(*pause*) → *dialysingstate* & *poweredon*

5.2 The Translation

The spreadsheet was translated systematically into MAL. During the analysis an automatic translator was developed based on translation patterns (explained further below) identified during the manual process. The automatic translator takes the CSV file representing the state transition model and produces its corresponding MAL representation following a translation strategy described in [8]. This ensures that the MAL model represents the finite state model, as described by the spreadsheet, accurately.

As an illustration of the translation consider the situation when an event occurs and the controller software changes the state. We consider the transition involving *M1Stall* in state *RST_InitS1* highlighted in Figure 1. Events are described in MAL as actions. These actions transition to different states depending on the current state. The controller software assumes that a pipeline of events exists, each tick of the system process causes the next event to be taken from the pipeline. If the pipeline is empty then a specified default transition is taken. The model includes no specification of a pipeline of events, rather it assumes that at any stage of the process the pipeline may become empty and as a consequence the "default" event / action is taken. When several actions are possible because

the guard for each of them is satisfied then one of the actions is taken non-deterministically. There are some circumstances where it is necessary to prove properties that assume that the pipeline is never empty. For these situations we added an additional meta-attribute that becomes false if a default action occurs in a path (*dfltchk*).

The MAL description of the effect of *M1Stall*, when the event occurs in state *RST_InitS1*, is as follows:

$$statedist = sdRSTInitS1 \rightarrow [acM1Stall]\ trRSTInitS2$$

This MAL rule describes a transformation. When the state is *RSTInitS1* the action *acM1Stall* leads to the state *RSTInitS2*. The attribute *statedist* indicates the current state. If *RSTInitS1* is the current state then *statedist* takes the value *sdRSTInitS1*. The model defines a set of transformations that change current state to specified new states. Hence *trRSTInitS2* specifies a transformation to the state *RSTInitS2* as is described below.

$$\begin{aligned} trRSTInitS2 = {} & Power' = ALLOW12V\ \&\ Motor1' = M1STOP\ \& \\ & Motor2' = M2FWDMAX\ \&\ Hep' = HEPSAFE\ \& \\ & Peri' = PERISAFE\ \&\ Valve' = FLUSH\& \\ & Alarm' = ACTIVE\ \&\ WashTimer' = ZERO\ \& \\ & DialysisTimer' = HOLD\ \&\ Flash' = ENABLE\ \& \\ & Mode' = RESET\ \&\ statedist' = sdRSTInitS2 \end{aligned}$$

This transformation specifies new values for each of the attributes, for example the value of the attribute *Motor1* becomes *M1STOP* etc. This state transition is further augmented in the model to include attributes that do not appear explicitly in the state transition table as follows:

$$\begin{aligned} & seclr' = GREEN\ \&\ !audiblealert'\ \&\ fkey1' = F1BLANK\ \& \\ & \quad fkey2' = F2BLANK\ \&\ fstop' = F3STOP \end{aligned}$$

These additional attributes deal with features of the controller that are only listed as comments in the spreadsheet, and have the following meaning, and are not currently supported by the translator.

statedist marks the current state, designed to ease the model's identification of the current state.
seclr describes the colour of the state pane on the display.
audiblealert whether the alert if any is audible.
fkey1 the function display for *key1*.
fkey2 the function display for *key2*.
fstop the function display for *stop*.

6 Requirements expressed in formal logic

6.1 Refining a sketch requirement as a CTL property

As discussed in Section 4, the risk log contains a list of requirements developed in response to known hazards. The example to be considered in more detail was initially sketched by developers as:

If M2 in {M2Fwd} → M1 in {M1Bck}

This semi-formal representation indicates that it should always be the case that if the state of the motor $M2$ is "moving forward" then the motor $M1$ should be "moving backward". Further discussion with the developers produced refinement of this sketch requirement. The two sets *M2FWD* and *M1BCK* were described as "enumerations" in MAL using the following syntax:

M2FWD = { M2FWDMAX, M2FWDUNUF }
M1BCK = { M1BCKMAX, M1BCKUF, M1WITHDRAW }

All these states involved forward and backward motion in the two motors. Having defined the relevant state attributes as specified in the spreadsheet model, the next step was to formulate a precise version of the property as a basis for the analysis. The notation used was that supported by the NuSMV model checking tool.

The property notation CTL [7] is widely used and provides two kinds of temporal operator: operators over paths and operators over states. Paths represent the possible future behaviours of the system. When p is a property expressed over paths, $A(p)$ expresses the property that p holds for all paths and $E(p)$ that p holds for at least one path. Operators are also provided over states. When q and s are properties over states, $G(q)$ expresses the property that q holds for all the states of the examined path; $F(q)$ that q holds for some states over the examined path; $X(q)$ expresses the property that q holds for the next state of the examined path; while $[qUs]$ means that q holds until s holds in the path.

CTL contains a subset of the possible formulae that arise from the combination of these operators. $AG(q)$ means that q holds for all the states of all the paths; $AF(q)$ means that q holds for at least one state in all the paths; $EF(q)$ means that q holds in at least one state in at least one path; $EG(q)$ means that q holds for all states in at least one path; $AX(q)$ means that q holds in the next state of all paths; $EX(q)$ means that there is at least one path for which q holds in the next state; $A[qUs]$ means that q holds until some other property s holds in all paths; $E[qUs]$ means there exists at least one path in which q holds until some property s.

6.2 Categories of requirements

The discussion of all the elements of the risk log led to a consideration of requirements that fall into the following categories:

P1: specified states are inaccessible in dangerous circumstances. The property described in Section 6.1 is an example of such a property. Another example is that: "it should not be possible to dialyse an infant with heparin in the blood circuit".

P2: when the dialysis machine is error free it always generates a correct dialysis sequence. This sequence includes wash and dialysis stages.

P3: when an error event occurs then the device is taken to an appropriate error state.

P4: states can only be reached if combinations of states have happened in the past. An example of such a property is that relevant reminders are always displayed to "close a clamp before the next phase of the cycle can commence".

7 Risk related properties checked of the model

The particular requirements that fall into these categories will be considered in detail in this section. An important part of the description is the discussion within the team that triggered the refinement of initial versions of properties.

P1: Unsafe Combinations of states cannot occur

The requirement (of Section 6.1) was formulated in CTL (using the MAL notation "*in*" for membership of an enumerated set) as:

$$AG(Motor2 \ in \ M2FWD \rightarrow Motor1 \ in \ M1BCK) \tag{1}$$

This property specifies that it is always the case, for all states, that when *Motor2* is in a forward state then *Motor1* is in a backward state. This property is not true of the model (as was revealed during an analysis meeting) producing a counter-example that indicates one set of circumstances in which the property fails. Figure 3 describes the sequence starting from the initial state (column 1), ending at a state where the property fails to be true (column 6). Columns indicate values held by attributes. These are named in the left hand column (i.e., column 0). For example, the attribute *Power* has value *ALLOW12V* in column 4. The colour yellow is used to indicate that a state attribute has changed value between successive states. The path indicates (as shown in the row marked *main.action*) that from the initial state the device defaults (that is it takes the action *acDefault*) because there are no events in the queue. This action is followed by *Key2*, followed by *12voff*, *12von* and *M1stall* which leads to the state where the property fails. Discussion during the risk meeting explored the implications of the sequence and came to the conclusion that this exception was acceptably safe and could therefore be excluded. A process then continued of excluding states before formulating a property that excluded all discovered exceptions:

$$\begin{array}{l} AG(Motor2 \ in \ M2FWD \rightarrow (Motor1 \ in \ M1BCK \mid \\ \quad statedist \ in \ \{sdRSTInitS2, sdWAS2Empty, \\ \qquad sdWAFlushRlx, sdWAS2Flush\}) \end{array}$$

The risk analysis team considered each of these exceptions and noted that the common property of these counter-examples was that they occurred when the device was not in dialysis mode, hence the following property was constructed:

$$\begin{array}{l} AG((Motor2 \ in \ M2FWD \ \& \\ \quad Mode \ in \ \{DIALYSE, DIALYSING\}) \\ \qquad \rightarrow Motor1 \ in \ M1BCK) \end{array}$$

The property formulated as a result of this observation is true for the model. It could be argued that visual inspection of the spreadsheet would have been sufficient to indicate the problem in this particular case. However this systematic approach to finding paths to potentially hazardous states provides an exhaustive approach.

	1	2	3	4	5	6
main.action		acDefault	acKey2	ac12Voff	ac12Von	acM1stall
Alarm	INHIBIT	INHIBIT	QUIET	QUIET	ACTIVE	ACTIVE
DialysisTimer	ZERO	HOLD	HOLD	HOLD	HOLD	HOLD
Flash	DISABLE	ENABLE	ENABLE	ENABLE	ENABLE	ENABLE
Hep	HEPUNSTAL	HEPSAFE	HEPUNSTAL	HEPSAFE	HEPSAFE	HEPSAFE
Mode	RESET	RESET	RESET	RESET	RESET	RESET
Motor1	M1UNSTALL	M1SAFE	M1UNSTALL	M1SAFE	M1FWDMAX	M1STOP
Motor2	M2UNSTALL	M2SAFE	M2UNSTALL	M2SAFE	M2SAFE	M2FWDMAX
Peri	PERIUNSTAL	PERISAFE	PERISAFE	PERISAFE	PERISAFE	PERISAFE
Power	TRIP12V	TRIP12V	TRIP12V	ALLOW12V	ALLOW12V	ALLOW12V
Valve	UNLATCH	UNLATCH	VALVESAFE	VALVESAFE	PREP	FLUSH
WashTimer	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO

Fig. 3. Counter-example to property 1

P2: Staying in the dialysis cycle

The requirement described in the risk log is expressed as follows:

> MAL.DIALCYCLE: Unless there are errors or user actions, the system stays in the 'dialysis cycle'

The requirement aims to ensure that, barring error events or user interventions, the transition table will always cause the device to complete the same haemodialysis process. To check the requirement it is first assumed that no such cycle exists. This property should fail and give, as an example of failure, one cycle that is of the appropriate form. Once the cycle is discovered, and it is the correct 'dialysis cycle', the next stage is to show that it is the only possible cycle that can be generated using the relevant actions.

The first step in demonstrating this requirement is to find a cycle, show it is the only cycle and check that is the required cycle. The cycle must begin with the action *M1out* in the state *DIA_Wdraw*. The approach therefore is to show that, once reached, this state is reached again. However it is necessary to be more precise than this. The cycle must be achieved with a subset of actions, excluding for example error actions or user interventions. Only specific actions are recognised are valid "drivers" of the cycle. A "meta"-attribute *motorsandwaits* is therefore introduced that is set true by the first action and preserved by operations : *M1in*, *M2in* and *Wait1second*. All other actions set the *motorsandwaits*

attribute to false. A counter-example to the CTL property:

$$AG(statedist = sdDIAWdraw \rightarrow AX(AG(!(motorsandwaits \ \& \ statedist = sdDIAWdraw)))) \quad (2)$$

should then generate the cycle. This is illustrated in the trace fragment in Figure 4. The sequence fragment starts with the state *DIAWdraw* (bottom row, column 23) when *motorsandwaits* is false and ends with the state *DIAWdrawRlx* (column 29). The sequence of actions that make up the cycle are shown in a row at the top of the table (*acM1out* etc.). This sequence is indeed the "dialysis cycle" as acknowledged by the domain experts during a meeting. The other rows show the values of the state attributes relating to each state as represented by a column of the table. The value of *motorsandwaits* is shown in the penultimate row and remains true throughout. This, however, is only part of the analysis because it identifies *one* cycle only. It does not exclude the possibility that there are others. It is further necessary to check that no other sequences can be produced using this subset of actions, i.e., the discovered cycle is unique. This can be achieved by using properties that require that at each step of the cycle any valid action will result in the next step of the cycle as discovered in the counter-example.

	23	24	25	26	27	28	29
main.action	ac12Von	acM1out	acWait1sec	acM1in	acM2in	acWait1sec	acM1in
Alarm	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
DialysisTimer	TICK	TICK	TICK	TICK	TICK	TICK	TICK
Flash	NOSAMPLE	NOSAMPLE	DOSAMPLE	ENDSAMPLE	NOSAMPLE	NOSAMPLE	NOSAMPLE
Hep	HEPINFUSE	HEPINFUSE	HEPINFUSE	HEPINFUSE	HEPINFUSE	HEPINFUSE	HEPINFUSE
Mode	DIALYSING	DIALYSING	DIALYSING	DIALYSING	DIALYSING	DIALYSING	DIALYSING
Motor1	M1WITHDR	M1STOP	M1FWDUNU	M1BCKUF	M1STOP	M1RETURN	M1WITHDR
Motor2	M2STOP	M2STOP	M2BCKUF	M2FWDUNU	M2STOP	M2STOP	M2STOP
Peri	PERIPERFUS	PERIPERFUS	PERIPERFUS	PERIPERFUS	PERIPERFUS	PERIPERFUS	PERIPERFUS
Power	ALLOW12V	ALLOW12V	ALLOW12V	ALLOW12V	ALLOW12V	ALLOW12V	ALLOW12V
Valve	BABY	BABY	DIAL	DIAL	DIAL	BABY	BABY
WashTimer	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO
motorsandwaits	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
statedist	sdDIAWdra	sdDIAWdra	sdDIAS1S2	sdDIAS2S1	sdDIAReturi	sdDIAReturi	sdDIAWdra

Fig. 4. Proving the 'dialysis cycle'

So for each state a property, demonstrating uniqueness is proved, for example:

$$AG(statedist = sdDIAWdraw \rightarrow AX(motorsandwaits \rightarrow statedist = sdDIAWdrawRlx))$$

Each state in the discovered cycle is considered and it is demonstrated that the only successor in each state, using the subset of events, is the next state found in the original cycle.

P3: Errors lead to error states

An important issue in the risk log was to ensure that error events would always lead to error states. This was expressed in the risk log as:

> MAL-GENERROR: For all error conditions and all system states, the next state will be an error state.

It was also required that the device would remain in an error state if further error events occur. To formulate the property a set of actions that represent the error events is first defined using MAL notation.

$$ErrorEventSet = acHardFault \mid acOverpressure \mid acBubble \mid acPeriStall$$

The set *ErrorStateSet* is defined in the model as an enumerated set that includes all the states that are determined to be error states. The required property was then agreed to be:

$$AG(AX(ErrorEventSet \rightarrow statedist\ in\ ErrorStateSet))$$

During the meeting this property was checked and found to be false. The reason for this failure, as determined by the counter-example, was that the Alarm can be inhibited and when this happens the property fails to be true. The property was therefore refined to include $Alarm\ !=\ INHIBIT$.

$$AG(Alarm\ !=\ INHIBIT \rightarrow AX(ErrorEventSet \rightarrow statedist\ in\ ErrorStateSet))$$

This property is also false. The state that offends in this case had not been counted as an error state and should have been. The set *ErrorStateSet* was therefore further augmented. The final refinement of the property further restricts to those states for which *Mode* is not *RESET*.

$$AG(Alarm\ !=\ INHIBIT\ \&\ Mode\ !=\ RESET \rightarrow AX(ErrorEventSet \rightarrow statedist\ in\ ErrorStateSet)) \quad (3)$$

Finally *STWarmStart* must also be excluded. This state is not considered to be problematic. Its occurrence is clear and will not cause confusion. These successive refinements have weakened the property and therefore a justification is required at each stage that the refined property is adequate mitigation for the possibility of an unrealised error. Discussion with the developers confirmed that this weakened formulation of the property was sufficient mitigation for possible risks and an explanation is provided in the risk log.

P4: States can only be reached if combinations of states have happened in the past

As was noted in the case of property P2, additional "meta-attributes" were introduced to the model so that it was possible to enrich the properties that could be proved using the model checker. In the case of P2, *motorsandwaits* was introduced to enable consideration of sequences of a subset of non error actions. Other requirements in the risk log could not be formalised as CTL properties using the original attributes of the model. In particular those properties that related to combinations of states that had happened in the past required such formulation. An example of a requirement of this kind is concerned with whether information is provided by the user interface to indicate to the user of the machine that a specific action should be carried out. *Flash* is a state attribute in the model that specifies the content of an information display. For example:

> "MAL.HEPCLIP: The user is instructed to close clip before changing syringe, and re-open afterwards."

Several Flash messages, specified by the attribute *Flash*, indicate dialyser warning displays. For example, $Flash = HEPCLOSE$ indicates that "close the heparin clip" has been transmitted. The attribute *hepclipopen* was included in the model specification and is set to true and continues to be true after a flash message that indicates that the heparin clip is open: $Flash = HEPOPEN$. It is made false by *Flash* taking values *HEPCLOSE* or *HEPSYRINGE*. The following fragment involving the *Hepin* specifies a transition to the state *HEPClip*. This state includes a change to the *Flash* attribute $Flash' = HEPCLOSE$ and therefore *hepclipopen* is set to false.

$$statedist\ in\ \{sdDIAReady\} \rightarrow [acHepin]$$
$$trHEPClip\ \&\ !motorsandwaits'\ \&\ keep(\dots)\ \&\ !hepclipopen'$$

We then check the property:

$$AG(Mode = DIALYSING \rightarrow hepclipopen)$$

The property asserts that you can only reach a dialysing state if a message to open the clip was the most recent flash relating to the clip and that the message had previously occurred. This property checked to be false. The state *HEPPrime* is also a clear indication to open the clip but does not involve the relevant flash. The model was changed therefore so that the meta-attribute was also set to true when visiting *HEPPrime*. The property then becomes true.

8 Discussion and Related work

The contribution of this paper is a practical demonstration of the use of formal techniques to analyse a component of a safety critical system. The approach was not novel. Similar techniques were being described and applied in the 1990s. For example, a mature set of tools have been developed by Heitmeyer's team using SRC [10]. Their approach uses a tabular notation to describe requirements which makes the technique relatively acceptable to developers. Atlee and Gannon described a similar approach in [2]. In some domains, other than medical domains, formal mathematically based methods have been effective in analysing and assessing risks systematically (see for example, [13, 3]). Despite the success of these techniques there is a continuing perception that formal methods are not easy to use and that they cannot be scaled to substantial systems. These barriers to their use have limited their uptake in medical domains. Recent research with the cooperation of the US Food and Drugs Administration (FDA) have led to increased possibilities for their potential use [12, 14]. The novelty here has been to apply this technique in a medical team where typically small teams with limited resources are involved.

The translation of the table into MAL, including meta-attributes, involved 682 lines, including 119 lines of state definitions and 152 lines of type and constant definitions. The development of the first model, by hand, took about seven hours. It was possible to make most changes to the model and show the results interactively during meetings with the development team without disturbing the flow of the meeting. Hence the refinement of requirements and the careful analysis of the hazards were facilitated by the process. The analysis involved 23 properties. On the rare occasions when it was not possible to refine a property during the meeting, for example when meta-attributes were required, this could be achieved within an hour outside the meeting. Verifying all the properties together on a MacBook Pro with Intel Core i5 clocked at 2.9GHz, with 8GB RAM and SSD memory, took 1.7 seconds. The exercise shows that, with appropriate expertise and using available artefacts (the table, safety requirements), the use of formal methods required little additional effort and supported effective discussion of the risks between the developers.

There are several ways in which it can be demonstrated that a device satisfies safety requirements using formal techniques. One way of doing this is to develop the device formally by refining the model as supported by tools such as Event B [1]. An initial model is first developed that specifies the device characteristics and incorporates the safety requirements. This initial model is gradually refined using details about how specific functionalities are implemented. This was not a realistic approach in the present case because, when the analysis was to be done, the device had already been developed. Indeed such techniques are not feasible given the typical resources available to medical device developers. Alternatively a model could be generated from the program code of an existing device, using a set of transformation rules that guarantee correctness, as discussed in [11]. This approach could have been used for other software aspects of the device, however it is unclear how well such techniques scale. Proving that the model

of this component of the software is correct with respect to the device was not a problem for the particular example because the software was driven by a table and the table was translated directly into the model. The analysis does not attempt to prove that the software drivers themselves were implemented correctly.

9 Conclusion

The risk analysis process described in the paper succeeded because the controller is table driven and it was relatively easy to generate a model from the table. It also succeeded because a mixed disciplinary team was involved. This team included one person who was able to use the formal tools and provide an explanation of the requirements and model formulations. It is standard practice to use a table to drive software that controls a multi-step process as in this case. However there are cases where this does not happen and moreover, as in this case, the software covered by the controller is only part of the software. The dialysis machine also includes user interface features, for example capacity to enter new values for thresholds relevant to the dialysis process. These are involved in the initial set-up of the machine. Other analyses, involving several of the authors, have focussed on existing IV infusion pumps [9]. In these cases such a table driven process, with the capacity to be automated, has not been possible. The analysis described in this paper therefore raises questions about the potential for extending this approach to a broader class of medical systems. The challenges raised by this analysis in the context of small-scale developments, such as this one, are:

Systematic modelling: While formal approaches to the development of software that refine safety requirements exist (see [17]), these are not yet feasible to use given the available tools and skills of existing small development teams. In this case the formal methods expertise was recruited short-term for the purpose

Mixed disciplinary teams: There was substantial benefit in recognising and using expertise from sources outside the development team. A mixed discipline approach is already in practice in the case of small companies or innovative pre-commercial developments. It would make sense therefore to add these analytical skills to the toolkit available to the device developers.

Mixed styles of analysis: As in this case a well defined and yet important software component may be analysed formally. The formal analysis of the controller table can also improve the testing coverage of the device drivers themselves although this was not done in this case. It is also good practice to have multiple independent arguments to demonstrate the safety of the system. Hence it makes sound sense to use formal techniques to improve confidence in the risk analysis.

This paper illustrates how formal techniques may be used successfully as part of the risk analysis process associated with the development of a medical device. This is part of the submission that has gone forward for regulation. The safety requirements that were formulated and proved, and improvements in the light of requirements failure, illustrate how the analysis led to improvement in the safety

of the design while providing a concise basis for evidence that part of the system is safe. The technique is readily repeatable. Tools that have been developed allow the automated development of models from control tables. The analysis approach complements testing techniques and provides a systematic solution to the safety assessment of critical devices.

Acknowledgements This work has been funded by: EPSRC research grant EP/G059063/1: CHI+MED (Computer–Human Interaction for Medical Devices). It has also been financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. J. M. Atlee and J. Gannon. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering*, 19(1):24–40, 1993.
3. J. Barnes, R. Chapman, R. Johnson, B. Everett, and D. Cooper. Engineering the tokeneer enclave protection software. In *IEEE International Symposium on Secure Software Engineering*. IEEE, 2006.
4. BSI. Medical device software - software life cycle processes. Technical Report BS EN 62304:2006, British Standards Institution, CENELEC, Avenue Marnix 17, B-1000 Brussels, 2008.
5. J. C. Campos and M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In N. Graham and P. Palanque, editors, *Interactive systems: Design, Specification and Verification, DSVIS '08*, number 5136 in Lecture Notes in Computer Science, pages 72–85. Springer-Verlag, 2008.
6. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An Open Source Tool for Symbolic Model Checking. In K. G. Larsen and E. Brinksma, editors, *Computer-Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
7. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
8. L. Freitas and A. Stabler. Translation strategies for medical device control software. Technical report, Newcastle University, August 2015.
9. M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon. Demonstrating that medical devices satisfy user related safety requirements. In Michaela Huhn and Laurie Williams, editors, *Software Engineering in Health Care: 4th International Symposium, FHIES 2014, and 6th International Workshop, SEHC 2014, Washington, DC, USA, July 17-18, 2014, Revised Selected Papers*, pages 113–128. Springer International Publishing, Cham, 2017.
10. C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj. Scr: A toolset for specifying and analyzing software requirements. In *Computer Aided Verification*, pages 526–531. Springer, 1998.

11. G. J. Holzmann. Trends in software verification. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 40–50. Springer-Verlag, 2003.
12. B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. Jetley. Safety-assured development of the GPCA infusion pump software. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT '11, pages 155–164, New York, NY, USA, 2011. ACM.
13. G. Klein, J. Andronick, K. Elphinstone, T. C. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive formal verification of an OS microkernel. *ACM Trans. Comput. Syst.*, 32(1):2, 2014.
14. P. Masci, A. Ayoub, P. Curzon, M.D. Harrison, I. Lee, O. Sokolsky, and H. Thimbleby. Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *Proceedings ACM Symposium Engineering Interactive Systems (EICS 2013)*, pages 81–90. ACM Press, 2013.
15. A.F. Monk, M. Curry, and P.C. Wright. Why industry doesn't use the wonderful notations we researchers have given them to reason about their designs. In D.J. Gilmore, R.L. Winder, and F. Detienne, editors, *User-centred requirements for software engineering*, pages 185–189. Springer, 1991.
16. US Food and Drug Administration. General principles of software validation: Final guidance for industry and FDA staff. Technical report, Center for Devices and Radiological Health, January 2002. Available at `http://http://www.fda.gov/medicaldevices/deviceregulationandguidance`.
17. S. Yeganefard and M. Butler. Structuring functional requirements of control systems to facilitate refinement-based formalisation. In *Proceedings of the 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011)*, volume 46. Electronic Communications of the EASST, 2011.