Demonstrating that medical devices satisfy user related safety requirements

M.D. Harrison[1,4], P. Masci[1], José Creissac Campos[2,3], and P. Curzon[1]

[1]Queen Mary University of London, School of Electronic Engineering & Computer Science, Mile End, London E1 4NS, UK
[2]Dep. Informática / Universidade do Minho, Braga, Portugal
[3]HASLab / INESC TEC, Braga, Portugal
[4]School of Computing Science, Newcastle University, Newcastle upon Tyne NE1 7RU

Abstract. One way of contributing to a demonstration that a medical device is acceptably safe is to show that the device satisfies a set of requirements known to mitigate hazards. This paper describes experience using formal techniques to model an IV infusion device and to prove that the modelled device captures a set of requirements. The requirements chosen for the study are based on a draft proposal developed by the US Food and Drug Administration (FDA). A major contributor to device related errors are (user) interaction errors. For this reason the chosen models and requirements focus on user interface related issues.

Keywords: Human error, formal verification, performance, medical devices, model checking, MAL, PVS

1 Introduction

Regulators in diverse domains, including Aviation, Power Generation and Medicine, are tasked to ensure that system or device developers demonstrate that risks associated with the device are minimal or "as low as reasonably practicable". This must be done before the system or device can be deployed in a safety critical context. Such a demonstration can involve proving that the device satisfies a set of safety requirements, designed to mitigate identified hazards (see [12]). The FDA has produced one such set of requirements in draft documentation [2] with a focus on medical devices, particularly infusion pumps. This paper is concerned with how to demonstrate that a device satisfies requirements, emphasising those that are *user related*. FDA guidelines propose that validation is "highly dependent upon comprehensive software testing, inspections, analyses and other verification tasks performed at each stage of the software development cycle" [19]. The data produced for such validation is usually substantial. The problem with testing (as has been noted often) is that it does not prove the absence of bugs. Formal techniques provide additional information. They are concise, precise and exhaustive. However they have the disadvantage that, with currently available tools, they are not easy to use.

This paper demonstrates the use of formal techniques to provide assurance that a user related subset of the FDA requirements are true of an IV infusion pump. A formal model of the pump is used to prove properties that capture these requirements. The paper describes the steps taken to demonstrate that the requirements are satisfied. These steps include demonstrating that: the model is a faithful representation of the device; the chosen requirements are equivalent to the proved properties; and that the properties are true of the model. While it is not possible to describe this process in detail, given space constraints, the paper aims to illustrate the approach to indicate its feasibility. Prospects for widespread adoption of formal techniques are also briefly discussed. This work progresses an agenda suggested by the FDA's use of formal techniques [11]. Section 2 sketches the infusion pump example. Section 3 to 5 are concerned with issues associated with producing a model that is a faithful representation of the device to be assessed. Section 6 suggests formalisations of the requirements that were used as a basis for proof. Section 7 briefly explores the proofs that were produced. The final section offers discussion and conclusions.

2 The medical example

The chosen device (the Alaris GP infusion pump [4] — see Figure 1) has characteristics that are common to many devices that control processes over time. The clinician user sets infusion pump parameters and monitors the infusion process using the device. A model of the particular pump had already been developed [8][1] as part of a general analysis of usability properties of the device. The Alaris GP has two basic modes (besides the off mode): infusing and holding. These each have sub-modes. In the infusing mode the volume to be infused (vtbi) is pumped into the patient intravenously according to an infusion rate. In infusing mode the vtbi can be exhausted, in which case the pump continues in KVO (Keep Vein Open) mode and sets off an alarm. In holding mode the device is not infusing and values and settings can be changed. The complexity of the interface to this device (and other members of the Alaris family of pumps) is mainly concerned with what is possible in holding mode where values and settings can be changed using a combination of function keys and chevron buttons (for the device layout, see Figure 1). A subset of the features that can be changed in holding mode can also be changed when infusing. Chevron keys are used to increase (*fup*/*sup*), or decrease (*fdown*/*sdown*), entered numbers incrementally. Holding the chevron key down may have the effect of accelerating the size of the increment or decrement. The details of this acceleration, if any, depends on the chevron key and the type of pump variable being entered. The current data entry mode governs whether the chevron buttons can be used to change infusion rate, vtbi and time, or alternatively allow the user to move between options in a menu, for example in bag mode and in query mode. Bag mode allows the user to select from a set of infusion bag options, thereby setting vtbi to a predetermined value. Query mode, invoked by the query button, generates a menu of set-up options. The

[1] The models can be found at `http://hcispecs.di.uminho.pt`.

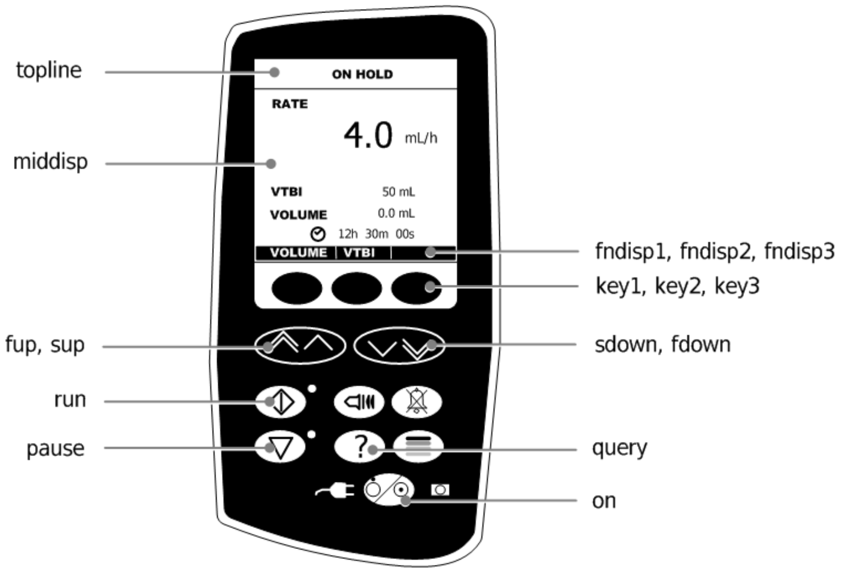

Fig. 1. Actions and attributes used to model the Alaris pump

set of available options depends on how the device is configured by the manufacturer. Options include locking the infusion rate, disabling the locking of it, setting vtbi and time rather than vtbi and infusion rate, and changing the units of volume and infusion rate. The device allows movement between display modes via three function keys (*key1*, *key2* and *key3*). Each function key has a display associated with it indicating its present function (*fndisp1*, *fndisp2* and *fndisp3*).

3 Modelling infusion devices: background and approach

3.1 Background

There are a number of ways in which requirements can be proved of a device 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 is not a realistic approach in the present case because the device had already been developed. Indeed such techniques are not currently used outside a research context. Alternatively a model could be generated from the code of an existing device, using a set of transformation rules that guarantee correctness. This is discussed in [10]. This approach was not feasible for the present study because the relevant code was not available. However it is a valuable and effective means of analysing requirements, and has been demonstrated for the number entry components of an infusion pump, see [15].

Proving that the model is correct with respect to the device can be achieved systematically through refinement and the development of models from code. Without either option being available, a model was developed *by hand*. This was achieved using a combination of user manuals, simulations and the device itself. In fact, in this case, the model had previously been developed for other purposes without the particular FDA requirements in mind. It had been used to analyse properties of the interactive modes that were described in the previous section. This analysis had considered whether modes were supported by the device without ambiguity, see [8]. For the present purpose this model was further developed to allow analysis of the details of the number entry system required to prove the FDA requirements.

Fidelity of the model to the implemented device was demonstrated by proving a range of properties. States of the modelled device were examined by comparing the traces of actions produced by the model checker, as counter-examples for these properties, with actual sequences generated by the device itself. A prototype was also produced automatically from the model to compare the "look and feel" of the actual device with the prototype, see [14] for details. The traces and simulations were produced at a level that meant they were indistinguishable from the physical device. The only difference between the simulation and the device was that no drug was infused, no connection was made to the patient and the precise timings differed. Time will be discussed briefly later. The simulations were generated with the aim that they could be explored by regulator or manufacturer. It is of course the case that they only allow an exploration of the paths that the regulator chooses to explore. Simulation can also be used to illustrate what the failure of a property means. Part of the argument to the regulator that this is acceptable may then involve a demonstration of the features of the device that fail the requirement, showing that they do not present a risk.

Proving requirements has been the focus of previous work. For example, a mature set of tools have been developed by Heitmeyer's team using SCR [9]. Their approach uses a tabular notation to describe requirements which makes the technique relatively acceptable to developers. Combining simulation with model checking is the focus of [7]. Recent work concerned with simulations of PVS specifications provides valuable support to this complementarity [14].

3.2 Modelling and analysis of the infusion pump

The process of proving FDA requirements described in this paper involves a sequence of steps. These are as follows.

1. *Developing a model of the device.* Modelling involves two stages. First a version is produced to be analysed using model checking. Second the model is transformed systematically into a form to be analysed using theorem proving.
2. *Validating the model against the physical device.* This is done using a combination of plausibility properties and simulation based on the model.

3. *Formalising the requirements.* This involves two stages. The first stage disambiguates the requirement in a form that can be transformed into a device specific property. This formalisation is described in more detail in [13]. The second stage involves refining the formalised requirement to be specifically about the Alaris device. This process is typically interactive and in principle involves discussion with both human factors specialists, checking the validity of the interpretation of the user-related requirement, and regulator to check that the property captures the spirit of the original requirement.
4. *Proving the property.* Finally prove the property by model checking or theorem proving. In the example, all the formalised requirements were proved using both technologies except those that involve full number entry which could only be proved using theorem proving.

The purpose of the paper is to show how these concrete steps were taken. Some details of the different notations and approaches are skimmed for reasons of space.

4 Developing a model of the device

The first step involved producing a model that could be analysed using model checking. This technology has the advantage that, since it is algorithmic, proof does not require a human prover. Theorem proving, on the other hand, usually requires proof guidance. The ingenuity in model checking is to formulate the properties appropriately and to interpret the results. The model under analysis had been developed in two parts. A generic "pump" component was developed that could be reused in other models. For example the BBraun infusion pump has also been studied in detail [8] and uses the same component. The Alaris version of the model focuses on mode behaviour. Proof with it is not tractable if the full number entry features of the model are also incorporated. To achieve tractability, token values were assumed for pump variables: vtbi, infusion rate, time and volume infused. They are taken to be integers in the range $0 \ldots 7$. These simplifications do not affect the mode behaviour of the device. The model used for the initial analysis was described in Modal Action Logic (MAL) [3] using the IVY tool [8]. MAL is a simple state transition language, easily translated from state transition diagrams or the SCR tabular format [9]. The notation is used because it is of a type that is more readily acceptable by developers. The MAL model is translated into NuSMV [5]. The properties that translate the requirements are expressed in CTL (see [6]). The following MAL modal axiom, involved in proving the requirement discussed in Section 7.1, describes conditions in which the button *key1* has the effect of confirming a device reset.

$$
\begin{array}{l}
topline = clearsetup \rightarrow [key1] \\
\quad topline' = holding \;\&\; middisp[drate]' \;\& \\
\quad middisp[dvtbi]' \;\&\; !middisp[dtime]' \;\& \\
\quad middisp[dvol]' \;\&\; !middisp[dbags]' \;\& \\
\quad !middisp[dkvorate]' \;\&\; !middisp[dquery]' \;\&
\end{array}
$$

$$\begin{aligned}&\mathit{fndisp1}' = \mathit{fvol} \ \& \ \mathit{fndisp2}' = \mathit{fvtbi} \ \& \\ &\mathit{fndisp3}' = \mathit{fnull} \ \& \ \mathit{entrymode}' = \mathit{rmode} \ \& \\ &!\mathit{rdisabled}' \ \& \ \mathit{onlight}' \ \& \ \mathit{pauselight}' \ \& \\ &!\mathit{runlight}' \ \& \ \mathit{effect}(\mathit{device.reset}) \ \& \ \mathit{keep}(\mathit{bagscursor}, \mathit{rlock})\end{aligned}$$

This axiom describes (to the right of [*key1*]) the effect of action *key1*. The expression to the left of the action ($\mathit{topline} = \mathit{clearsetup}$) states the condition under which the behaviour described for the action is performed. Hence, when the top line of the display shows "clear setup", and the action is invoked, then the expression on the right hand side describes the behaviour. Most of this behaviour changes the interface by defining values for attributes: *middisp*, *topline*, *fndisp1*, *fndisp2*, *fndisp3*, *onlight*, *pauselight*, *runlight*. The priming of an attribute (for example, $\mathit{topline}'$) indicates that the action changes the value of that attribute. The action *key1* also changes the mode of the device (*entrymode*) to allow entry of infusion rate (*rmode*). It also invokes a generic pump action (*device.reset*) that initialises all the pump variables. This action *reset* is accessed in the reusable pump component which is identified as *device* in the MAL specification. The $\mathit{keep}(\ldots)$ expression specifies which attributes are not affected by the action and remain unchanged.

The MAL model focuses on interface features and the modes of the device, describing concretely how actions change the display and modes of the device. It has a simple discrete model of time. An action *tick* increments time as the infusion process continues, or while the device is paused. In the latter case the value of time is used to determine how long the pause has been. This model, even without full number entry, requires substantial processing for analysis – between one and two hours per property on a typical desktop computer.

A second model was developed by translating the MAL systematically into PVS [18] (a theorem proving system). The PVS specification allows the analysis in principle of properties involving infinitely many states. The equivalent specification for the fragment described above is:

```
key1_case_clearsetup(st: (per_key1)):alaris =
st WITH [ topline := holding,
middisp := LAMBDA(x: imid_type):
COND x = drate -> TRUE,
x = dvtbi -> TRUE, x = dvol -> TRUE,
x = dtime -> FALSE, x = dbags -> FALSE,
x = dkvorate -> FALSE, x = dquery -> FALSE ENDCOND,
device := reset(device(st)),
fndisp1 := fvol, fndisp2 := fvtbi, fndisp3 := fnull,
entrymode := rmode ]
```

The PVS theory captures all the characteristics of the MAL model, including time, but also includes a full number entry model. The PVS features that correspond to MAL elements can be clearly seen in the specification. This function `key1_case_clearsetup` is invoked in the more general `key1` function when the condition `topline(st) = clearsetup` is true. The function has domain `(per_key1)` hence it is only permitted when `key1` is accessible to the user. The reason for having two models was that the counter-example approach supported by model checking facilitated analysis of the plausibility of the model and also refinement of the requirements in the early stages of development.

5 Validating the model against the physical device.

Plausibility was investigated first by checking properties and then by exploring the model through simulation. This simulation process is described in [14]. A typical example of a property checked that, once relevant pump variables had been entered, infusion would lead to a state in which the volume infused was equal to the vtbi.

$$\begin{aligned}&AG(\mathit{device.infusionrate} = 1 \ \& \ \mathit{device.vtbi} \ = \ 7 \rightarrow\\ &\quad AG(\mathit{device.volumeinfused} \ != \ 7))\end{aligned}$$

Properties such as these are expressed as negations in order to construct a counter-example that has the required properties. This property asserts that it is always the case for all paths that if infusion rate is set to 1 (a token) and vtbi is set to 7 then a state cannot be reached in which volume infused is 7. This property does not depend on the details of the device interface, depending only on the generic pump model, but produces results that enable an analysis of the interface, making possible a comparison between alternative interfaces. As expected, the property fails when checked and produces a trace of steps in which the infusion rate is set to 1 and vtbi is set to 7. It indicates that once this has happened eventually the device is set to infuse and then after more steps a state is reached where the volume that has been infused becomes 7. The trace can be compared with the actual device, thereby providing a visualisation of one possible path in the model.

The model checker (NuSMV [5]) accepts a finite state model and analyses it exhaustively to prove or disprove a property. Other model checkers exist that are not limited to finite state models but these do not significantly improve performance. They can complete the analysis in a reasonable time only if the models are not too large or complicated [7].

6 Formalising Requirements

Five FDA requirements described in [11] mitigate user related hazards:
R1 *Clearing the pump settings and resetting of the pump shall require confirmation.*
R2 *The pump shall issue an alert if paused for more than t minutes.*
R3 *If the pump is in a state where user input is required, the pump shall issue periodic alerts/indications every t minutes until the required input is provided.*
R4 *The flow rate for the pump shall be programmable.*
R5 *To avoid accidental tampering of the infusion pump's settings such as flow*

rate/vtbi, at least two steps should be required to change the setting.
Further requirements were added based on templates supported by the IVY tool [8]. They mitigate various use or "interaction" hazards for infusion pumps identified in the hazard analysis presented in [16].
R6 *Whenever a pump variable is being entered, the variable should be clearly identified and its current value visible to the user.*
R7 *The current mode should be clearly identified. Changes in mode therefore should have perceivable feedback to that effect.*
R8 *Confirmation of number entry should be achieved using a consistent action.*
R9 *Any data entry action can be reversed.*
Before these requirements can be proved of the model it is necessary to consider *precisely* how the requirements should be interpreted. For reasons of space a subset of the requirements (R1, R2, R4 and R9) is considered. The formalisation uses a PVS like specification that combines a functional notation similar to that used in programming languages, typically used by engineers, with logic connectives such as `AND` (conjunction), `OR` (disjunction) `=>` (implication). Precision is achieved by defining abstractions that can be more readily understood and expressed as properties that can be proved of either the MAL or PVS models. The formalisation must capture the essence of the requirements as understood both by the regulator who developed it in the first place and the human factors specialists who can comment on the user aspects of the requirements and whether they are fulfilled by the specific properties of the device. The formalisation must not be biased towards a particular make of device.

6.1 R1: Clearing the pump settings and resetting of the pump shall require confirmation.

This requirement aims to prevent users changing infusion settings inadvertently. The state in which the particular pump variable is ready to clear is described by the predicate: `pumpvariable_ready_to_clear`. The `clear_setting` action for the device does not update the value until a `confirm_action` has taken place. Any other action (`no_confirm`), permitted by the device when `clear_setting` has occurred, has no effect on the pump variable if taken. Each pump setting is dealt with individually and, for pump variable vtbi, can be expressed as follows:

```
vtbi_ready_to_clear(st, x) AND x /= 0 =>
(clear_setting_vtbi(st)‘vtbi = x AND
confirm_action(clear_setting(st))‘vtbi = 0 AND
no_confirm(clear_setting(st))‘vtbi = x)
```

where: `st` is the current state of the device; `vtbi` is the state attribute that correspond to the considered pump variable; and `x` is the value of the considered pump variable before resetting.

6.2 R2: The pump shall issue an alert if paused for more than t minutes.

R2 requires that the user is alerted if the device is left unattended during data entry, as might occur if the clinician is interrupted. This requirement can be formulated as:

```
user_input_strictly_overdue(st) => alert(st)
```

where `user_input_strictly_overdue` is true if the device has been paused without activity for a specified period. This predicate can be expressed in more detail as:

```
user_input_strictly_overdue(st) = paused(st) AND
elapsed(st) > timeout
```

`paused` and `elapsed` will have specific meanings for the particular infusion pumps. `alert(st)` describes an appropriate alert produced by the device.

6.3 R4: The flow rate for the pump shall be programmable.

This safety requirement ensures that any value for the flow rate can be programmed. The formalisation of the requirement indicates an inductive argument that proves there is always a sequence of actions to reach a particular value of the infusion rate. If the device is ready to enter the rate then there is always an action that will take the flow rate closer to the expected rate, and eventually the intended rate will be reached. For the illustrative purposes of the paper a simpler version of the requirement is adopted that demonstrates only that the flow rate gets closer to the expected rate (`e`) expressed as

```
FORALL (st: State, e: infusion_rate):
rate_entry_ready(st) =>
EXISTS (a: State -> State): (current_display_rate(st) > e =>
(current_display_rate(st) - e > current_display_rate(a(st)) - e))
AND (current_display_rate(st) < e =>
(current_display_rate(st) - e < current_display_rate(a(st)) - e))
```

The attribute `current_display_rate` is a visible representation of the current infusion rate.

6.4 R9: Any data entry can be reversed

It seems desirable that for any number entry action there is an action that will reverse the action. This can be expressed as:

```
ready_to_enter_pump_variable =>
FORALL (act1: State -> State): EXISTS (act2: State -> State):
act2(act1(st))‘pump_variable = st‘pump_variable
```

This formulation reveals interesting features of many of the number entry schemes for infusion pumps, because, as discussed below, in general this simple notion of reversability is not possible.

7 Proving Requirements

Theorem proving uses natural deduction to do proof. Induction can be used to prove general properties over very large numbers of states. For this reason properties can be proved that are beyond the capacity of readily available computers using model checking. Setting up the induction and guiding the proof requires skill. When a proof fails it can be difficult to see why it has gone wrong and what must be done to remedy it — a process that is relatively straightforward using a model checker through available counter-examples.

Proof by model checking provides clear counter-examples that aid diagnosis and reformulation of models and properties. Proof by theorem proving is faster but failure requires more skill to interpret. In the present case the two safety requirements (R4 and R9), that require precise modelling of the number entry system of the device, could not be proved using model checking. R1 and R2 could be proved using both model checking and theorem proving. Model checking required approximately 90 minutes using an Intel 2.4 GHz i5 with 8GB of RAM (1333 MHz DDR3) when the properties are taken individually.

The first stage in proving a requirement is to take the formalisations discussed in Section 6 and to further refine them as properties of the particular device.

7.1 Proving R1: Clearing the pump settings and resetting of the pump shall require confirmation

The Alaris device is ready to clear vtbi when the device is powered on and in the holding state. The pump variable needs clearing so should be non-zero. Relevant state attributes for expressing whether the pump is infusing and switched on are `infusing?` and `powered_on`. The `vtbi_ready_to_clear` predicate is therefore:

```
vtbi_ready_to_clear(st: alaris, x: ivols): bool =
NOT device(st)‘infusing? AND device(st)‘powered_on? AND
device(st)‘vtbi=x AND x \= 0
```

`clear_setting(st: alaris): alaris = on(on(st))` because *on* switches the device off if switched on and vice versa. `clear_user_confirm(st: alaris): alaris = key1(st)`. That `key3` is the only other available action is checked by the predicate `no_confirm` expressed as:

```
no_confirm(st: alaris): alaris =
NOT (per_sup(st) OR per_fup(st) OR per_sdown(st) OR per_fdown(st) OR
per_key2(st) OR per_query(st) OR per_run(st) OR per_pause(st))
```

Here `per_action` is true if `action` is permitted to have an effect. The theorem combines these elements:

```
R1vtbi: THEOREM
FORALL (st: alaris, x: ivols):
LET stprime=clear_setting(st) IN (vtbi_ready_to_clear(st,x) =>
(topline(stprime) = clearsetup AND device(stprime)`vtbi=x
AND no_confirm(stprime) AND device(key1(stprime))`vtbi=0
AND device(key3(stprime))`vtbi=x))
```

The assertion captured in the theorem is required to be proved of all states and is easy to prove in PVS. The general purpose proof command `grind` proves the theorem. In most other theorems it is necessary to restrict the proof to reachable states. This must be done explicitly when theorem proving but is checked automatically by the model checker.

7.2 Proving R2: The pump shall issue an alert if paused for more than t minutes

The `user_input_strictly_overdue` predicate is defined as:

```
user_input_strictly_overdue(st: alaris): bool =
   device(st)`powered_on? AND NOT device(st)`infusing? AND
      (device(st)`elapse > timeout)
```

The attribute `elapse` specifies the time since the user last used the device when in holding mode. `elapse` is incremented when the device is paused each time the `tick` action is invoked. The alert is specified as: `alert(st: alaris): boolean = topline(st) = attention`. The assumption is that the alert is only indicated by an appropriate top line on the display. In reality there is also an audible alarm. These additional features are easily modelled. The assertion that is to be proved refines the property of Section 6.2 as follows.

```
R2assertionwithouttick(st: alaris): boolean =
      user_input_strictly_overdue(st) => alert(st)
```

The theorem is formulated as a structural induction. It requires that, over all states that can be reached from the initial state by an Alaris action, the assertion is true. The predicate `alaris_transitions` expresses this reachability property:

```
alaris_transitions
(pre, post: alaris): boolean =
(per_sup(pre) & post = sup(pre)) OR (per_fup(pre) & post = fup(pre)) OR
(per_sdown(pre) & post = sdown(pre)) OR
(per_fdown(pre) & post = fdown(pre)) OR
(per_tick(pre) & post = tick(pre)) OR (per_key1(pre) & post = key1(pre)) OR
(per_key2(pre) & post = key2(pre)) OR (per_key3(pre) & post = key3(pre)) OR
(per_query(pre) & post = query(pre)) OR post = on(pre) OR
(per_run(pre) & post = run(pre)) OR (per_pause(pre) & post = pause(pre))
```

The state `pre` is associated with the state `post` by an action that is permitted by the device. The appropriate permission predicate is omitted when the action is always permitted. The theorem that uses the structural induction is as follows.

```
R2withouttick: THEOREM
FORALL (pre, post: alaris):
(init?(pre) => R2assertionwithouttick(pre)) AND
((R2assertionwithouttick(pre) AND
alaris_transitions(pre, post)) => R2assertionwithouttick(post))
```

7.3 Proving R4: The flow rate for the pump shall be programmable

The components required in the formalisation are as follows.

```
rate_entry_ready(st: alaris): boolean =
switchedon?(st) AND NOT rlock(st) AND
(((entrymode(st) = rmode) AND (topline(st) = holding)) OR
((entrymode(st) = infusemode) AND (topline(st) = infusing)))
```

The device is ready to accept a rate value when: the device is switched on; infusion rate is not locked; and the top line shows holding or infusing. Further redundant constraints are included relating to entry mode to focus the states under consideration in the proof.

Two possible actions are dealt with in the theorem. If the expected rate exceeds the current display rate then the single chevron up key moves the current display rate closer to the expected rate. On the other hand if the current display rate exceeds the expected rate the single chevron down will get the current display rate closer. This theorem could be extended by using the size of the difference to use double or single chevron keys. However a simpler version of the requirement is adopted that nevertheless fulfils the requirement. The current display rate is defined for the Alaris as: `current_display_rate(st: alaris): irates = device(st)‘infusionrate` The theorem is based on the formulation of Section 6.3.

```
R4: THEOREM
FORALL (st: alaris, expected_rate: irates):
(rate_entry_ready(st) =>
(((expected_rate > current_display_rate(st)
AND per_sup(st) =>
(expected_rate - current_display_rate(sup(st))) <
(expected_rate - current_display_rate(st)))) AND
((expected_rate < current_display_rate(st)
AND per_sdown(st) =>
(current_display_rate(sdown(st)) - expected_rate) <
(current_display_rate(st) - expected_rate)))))
```

An element of the theorem, not expressed in the formulation of Section 6.3, checks that the action is available to the user before applying it.

7.4 Proving R9: Any data entry action can be reversed.

Reversability of number entry actions is only true in limited situations. For example:

- Applying double chevron up to 99 and then applying double chevron down produces 90.
- Applying double chevron down to 100 and then applying double chevron up produces 91.
- Applying single chevron up to 99.9 and then applying single chevron down produces 99.
- Applying single chevron down to 100 and then applying single chevron up produces 99.9.

These anomalies arise because there are thresholds in which the effect of the chevron action changes. The effect of the action depends on the current value within that threshold. Hence because 99.9 is less than 100 single chevron up increments by 0.1. But now it is greater than or equal to 100, so single chevron down decrements by 1. These anomalies were "discovered" through a process of trial and error, successively reformulating the theorem until it was proved true. One of the several theorems proved in this category, which elaborates the formulation of Section 6.4, is:

```
R9ratesdownsupqpt1: THEOREM
FORALL (st: alaris):
(rate_entry_ready(st) AND
per_sup(st) AND per_sdown(release_sup(sup(st))) AND
(device(st)‘infusionrate >= 0) AND
((device(st)‘infusionrate + small_step/10) <100) AND
(floor(device(st)‘infusionrate*10) =
device(st)‘infusionrate*10) AND
(ceil_rate(device(st)‘infusionrate*10) =
device(st)‘infusionrate*10))
=> device(release_sdown(sdown(
release_sup(sup(st)))))‘infusionrate =
device(st)‘infusionrate)
```

`rate_entry_ready` is as specified in the case of R4 and `small_step` is defined to be `1`. Further clauses in the theorem are concerned with (1) affirming that the appropriate number entry action is available to the user; (2) specifying that the number is expressed with no greater precision than one decimal place and (3) including the release action that is required after pressing any chevron button to indicate that the button has been released (note that any chevron button can be held down, thereby increasing the size of the increment or decrement achieved by the button). This theorem is true for all states without the need for induction. However constraint of the numbers to one decimal place is required in the proof. This is a property of the number entry system on the Alaris that should be proved and would require a structural induction to prove it. Similar parts of the theorem have been proved for the ranges 100 to 1000 and greater than 1000.

Proof of the last requirement, with all its qualifications, is of little value as an assurance that number entry actions are easily reversible by users. The qualifications that were developed in attempting to prove the theorem were however valuable in understanding the characteristics of the number entry system. Indeed formulating and proving the theorem raises questions as to whether the device is acceptable or whether it is likely to lead to interaction error. It should be noted that the latest release of Alaris firmware has fixed these inconsistencies.

8 Discussion and Conclusions

Demonstrating that the design of a medical device has been constructed to reduce the probability of interaction error to "as low as reasonably practicable" is a serious issue. It is estimated that there were 56,000 adverse event reports relating to infusion pumps between 2005 to 2009 in the United States including at least 500 deaths. This has resulted in 87 infusion pump recalls to address identified safety concerns, according to FDA data. Of these adverse event reports interaction error has been a significant factor. The documentation provided by manufacturers to regulators as part of a safety argument is usually substantial. However, the scale of the argument inevitably makes it difficult for regulators to comprehend them and to be confident that the evidence provided is of satisfactory quality. The use of formal techniques has a number of potential advantages. (1) It is precise and concise. (2) Tools like model checkers and theorem provers provided to support them enable mechanical and exhaustive verification. (3) The use of simulation techniques combined with the specification can clearly demonstrate how potential problems are addressed.

There are however obstacles to the immediate take-up of these facilities. They are not currently part of a typical developer's suite of tools. They are not currently used in product development. However there are signs of interest in these techniques. For example the FDA has developed generic PCA models [17] using simulink. A number of universities have demonstrated the role of a variety of formal verification techniques in relation to medical devices.

This paper has described one experience using formal verification technologies to verify draft FDA safety requirements for a commercial medical device. Examples were illustrated in detail to explore three key verification challenges: how to validate a model and show that it is a faithful representation of the device; the benefits of formalising requirements given in natural language; how to prove requirements on realistic models of devices. Our main contribution is to demonstrate how to achieve the FDA's agenda of using formal methods that can support the approval process for real medical devices. In particular requirements related to interaction issues have been formalised and verified.

An important issue that has not been fully resolved is how to check that a model is a faithful representation of the device. In this case model checking of properties combined with simulation was used to support this process by generating traces to be validated on the device. However, manufacturers have access to source code and therefore they can create faithful models systematically. An important issue not addressed here is how to validate that the considered set of safety requirements correctly address the usability and safety of the device for the context in which the device is to be used. This problem is orthogonal to the techniques presented in this paper. Safety aspects can be addressed with a human factors emphasis using a hazard analysis such as the one presented in [14].

Formalising the requirements provides benefits in addition to the ability to prove them. It led to much more detailed thinking about the precise nature of the requirements, both in general and for a specific device under test, than was possible in the informal natural language version. The pragmatic and informal combination of model checking and theorem proving provided powerful tools for analysis. By using each flexibly for requirements they were suited to, rather than ideologically favouring one for all requirements, or trying to combine them into a single tool applying both, it was possible to prove the requirements with minimum effort. One potential drawback of this approach is the need to master the two verification techniques. Indeed, both verification methods currently require significant skills for analysis. However, we have observed recurrent patterns in the structure of the formal models of devices from different manufacturers, and in the strategies needed to complete verification of several types of safety requirements. Therefore there are clear opportunities to create a reference template that can be used to automate the approach and thus reduce the analysis effort.

Acknowledgements This work has been funded by the EPSRC research grant EP/G059063/1: CHI+MED (Computer–Human Interaction for Medical Devices). J. C. Campos was funded by project NORTE-07-0124-FEDER-000062.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
2. David Arney, Raoul Jetley, Paul Jones, Insup Lee, Oleg Sokolsky, Arnab Ray, and Yi Zhang. Generic infusion pump hazard analysis and safety requirements. Technical Report MS-CIS-08-31, University of Pennsylvania, February 2009.
3. J. C. Campos and M. D. Harrison. Interaction engineering using the IVY tool. In G. Calvary, T.C.N. Graham, and P. Gray, editors, *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 35–44. ACM Press, 2009.
4. Cardinal Health Inc. Alaris GP volumetric pump: directions for use. Technical report, Cardinal Health, 1180 Rolle, Switzerland, 2006.
5. 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.

6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
7. G.E. Gelman, K.M. Feigh, and J. Rushby. Example of a complementary use of model checking and agent-based simulation. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 900–905, Oct 2013.
8. M.D. Harrison, J.C. Campos, and P. Masci. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering*, pages 1–17, April 2013.
9. Heitmeyer, J. C., Kirby, and B. Labaw. Applying the SRC requirements method to a weapons control panel: an experience report. In *Proceedings of the Second Workshop on Formal Methods in Software Practice (FMSP '98)*, pages 92–102, 1998.
10. 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 Berlin Heidelberg, 2003.
11. R. Jetley, S. Purushothaman Iyer, and P.L. Jones. A formal methods approach to medical device review. *Computer*, 39(4):61–67, 2006.
12. N. G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety (Engineering Systems)*. MIT Press, 2011.
13. 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.
14. P. Masci, A. Ayoub, P. Curzon, I. Lee, O. Sokolsky, and H. Thimbleby. Model-based development of the generic PCA infusion pump user interface prototype in PVS. In F. Bitsch, J. Guiochet, and M. Kaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 228–240. Springer Berlin Heidelberg, 2013.
15. P. Masci, Y. Zhang, P. Jones, P. Curzon, and H. W. Thimbleby. Formal verification of medical device user interfaces using PVS. In *ETAPS/FASE2014, 17th International Conference on Fundamental Approaches to Software Engineering*, Berlin, Heidelberg, 2014. Springer-Verlag.
16. Paolo Masci, Yi Zhang, Paul Jones, Harold Thimbleby, and Paul Curzon. A Generic User Interface Architecture for Analyzing Use Hazards in Infusion Pump Software. In Volker Turau, Marta Kwiatkowska, Rahul Mangharam, and Christoph Weyer, editors, *5th Workshop on Medical Cyber-Physical Systems*, volume 36 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–14, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
17. A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. P. E. Heimdahl. Compositional verification of a medical device system. In *Proceedings ACM High Integrity Language Technologies HILT'13*. ACM Press, 2013.
18. N. Shankar, S. Owre, J. M. Rushby, and D. Stringer-Calvert. PVS System Guide, PVS Language Reference, PVS Prover Guide, PVS Prelude Library, Abstract Datatypes in PVS, and Theory Interpretations in PVS. Computer Science Laboratory, SRI International, Menlo Park, CA, 1999. Available at `http://pvs.csl.sri.com/documentation.shtml`.
19. 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`.