Electronic Communications of the EASST
Volume 69 (2013)

Proceedings of the 5th International Workshop on Formal Methods for Interactive Systems (FMIS 2013)

Developing and Verifying User Interface Requirements for Infusion Pumps: A Refinement Approach

Rimvydas Rukšėnas, Paolo Masci, Michael D. Harrison, and Paul Curzon

12 pages

Guest Editors: Judy Bowen, Steve Reeves
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Developing and Verifying User Interface Requirements for Infusion Pumps: A Refinement Approach

Rimvydas Rukšėnas[1], **Paolo Masci**[1], **Michael D. Harrison**[1],[2] **and Paul Curzon**[1]

[1] School of Electronic Engineering and Computer Science
Queen Mary University of London, London, UK

[2] School of Computing Science
Newcastle University, Newcastle upon Tyne, UK

Abstract: It is common practice in the description of criteria for the acceptable safety of systems for the regulator to describe safety requirements that should be satisfied by the system. These requirements are typically described precisely but in natural language and it is often unclear how the regulator can be assured that the given requirements are satisfied. This paper is concerned with a rigorous refinement process that demonstrates that a precise requirement is satisfied by the specification of a given device. It focuses on a particular class of requirements that relate to the user interface of the device. For user interface requirements, refinement is made more complex by the fact that systems can use different interaction devices that have very different characteristics. The described refinement process recognises an input/output hierarchy.

Keywords: safety, human reliability, medical devices, refinement, user requirements, Event-B

1 Introduction

Demonstrating that interactive devices are acceptably safe is a significant and important element in their development. For example, design errors in medical devices have an impact on patient safety and contribute to health-care costs. Because of this, medical device regulators require manufacturers to provide sufficient evidence that the risks associated with the device are as low as reasonably practicable as well as being fit for purpose before entering the market. This process is known as the premarket review process.

The level of scrutiny in the pre-market review generally depends on the risks inherent in the use of the device. For new medical devices it involves submitting sufficient engineering and clinical evaluation evidence that the device can be safely deployed in the field. To expedite the premarket review process, faster routes exist for devices providing functionalities that are similar to those of already legally marketed products. For such devices, manufacturers need to demonstrate "substantial equivalence to a predicate device", that is they need to demonstrate that the new device has the same intended use of and is as safe and effective as an already legally marketed device (the predicate device). In the US, for instance, this process is defined in the Premarket Notification document known as 510(k) [LF09].

Regulators and manufacturers depend on these faster routes for cost reasons. Recent figures suggest that the majority of devices are approved this way. In the US alone, over five thousand new devices require 510(k) review each year. In its current form, the pre-market approval process involves the analysis of tens of thousands of printed pages [MAC+13] rather than a direct evaluation of the product. The structure and content of the provided documents are not standardised, which makes the review process hard for regulators since they must substantively review the documents within a relatively short time frame (e.g., 90 calendar days of the filing date for 510(k) applications).

The US Food and Drug Administration (FDA), the regulator for medical devices in the US, is promoting approaches based on the use of formal methods as a means to reduce the amount of paperwork and enable the submission of more succinct and rigorous evidence. For instance, at the FDA's Office of Science and Engineering Labs (OSEL), engineers are experimenting with *usage models* [JPJ06] for the verification of software. A usage model is a formal representation that describes the common characteristics and behaviour of software for broad classes of devices. The approach is based on the idea of developing usage models that satisfy core sets of safety requirements that can mitigate against typical hazards. This way, usage models can be used as a reference by manufacturers – it they are able to show that their product is compliant with the behaviours of the usage models, then regulators have evidence that the manufacturer's device meets minimum safety conditions.

The FDA currently specifies usage models as state machines. These models are developed manually starting from safety requirements, verifying the models against these requirements through model checking techniques. This paper shows how stepwise refinement and the Event-B/Rodin platform can be conveniently used to develop usage models that are correct by construction. The first problem with the FDA's approach is to express requirements so that they are sufficiently precise to be effectively operationalised. The second is to provide, by operationalising requirements, the means for encompassing the range of input/output technologies that are likely to be encountered in interacting with the systems. Event-B is used here to express the high level requirements such as those proposed by the FDA. Refinement is used to demonstrate that the requirement can be cascaded into a hierarchy that encompasses potential input/output technologies.

To illustrate the approach, we focus specifically on infusion pumps. We take as a starting point a particular sample set of requirements specified by the FDA. These are specified by the FDA in natural language. We give abstract formal specifications of these requirements. We then show how they can be refined to a more concrete version. This version can then be verified against the formal specification of specific pump designs. Here we concentrate on a particular infusion pump design based on a commercially available pump.

2 Outline of the Approach

The proposed approach is based on three layers: requirements hierarchy, interface hierarchy and concrete interfaces, each described below.

The requirements hierarchy layer, which is directly relevant to regulators, concerns the development of user interface requirements. The regulator will be interested in the satisfaction of these requirements to assure them of the device's safety. A minimal set of such requirements, relevant to some usability aspect of device interfaces, is developed. The aim is that these requirements should be sufficiently abstract to encapsulate the behaviour of the largest class of possible devices. Refinements are then used to detail these requirements in a sequence of steps. It is also possible that refinement can lead to alternative interface requirements that also provide assurance of the safety of the device. These modified requirements would be developed as a contract between regulator and manufacturer. The requirements hierarchy layer is discussed in Section 5.

The concrete interface layer focuses on the user interfaces of specific devices. This layer is most relevant to manufacturers as they demonstrate that the user interfaces of their devices satisfy the requirements developed in the requirements hierarchy layer. This is discussed in Section 6.

The middle layer, the interface hierarchy, aims to facilitate the dialogue between regulators and manufacturers in order to demonstrate that a specific user interface adheres to the relevant set of user requirements. It develops a refinement based hierarchy (classification) of user interfaces.

The aim is that user requirements are verified once for most abstract classes of interfaces. More concrete classes of interfaces at the lower levels of this hierarchy are then guaranteed to satisfy the requirements by construction. This simplifies the process of demonstrating that a specific interface satisfies the relevant user requirements. Instead of directly verifying the interface against the requirements it suffices simply to demonstrate that it is an instance of some concrete class of user interfaces. This approach, discussed in Section 7, correlates with the FDA pre-market approval process.

3 Sample User Interface Requirements from FDA

The regulator's aim is to be assured that risks associated with the use of a device are as low as reasonably practicable. As previously discussed part of this assurance is achieved through a credible demonstration that safety requirements are true of the device. Before showing how this demonstration can be achieved in the proposed framework a set of safety requirements developed by the FDA is described. These requirements relate to the usability of the data entry systems for infusion pumps. They will form the basis for the illustration contained in this paper. The safety requirements are taken from a larger set produced by the FDA (see Safety Requirements for the Generic PCA pump, obtained from rtg.cis.upenn.edu/gip.php3 on 4th April 2013). This set is intended specifically for PCA (Patient Controlled Analgesic) pumps. As a result there is more emphasis on patient tampering than clinician errors, and therefore the focus is slightly different than is relevant to the volumetric infusion pump used by clinicians that forms the basis of the example contained in this paper. The aim is to show how these independently determined properties can be framed in our framework.

The requirements in the FDA document related to data entry interfaces are listed below:

R1 *The flow rate for the pump shall be programmable.* This safety requirement aims to mitigate hazards due to incorrectly specified infusion parameters (e.g., flow rate is too high or low).

R2 *The VTBI (Volume to be infused) settings shall cover the range from v_{min} to v_{max} ml.*

R3 *The user shall be able to set the VTBI in j ml increments for volumes below x ml.*

R4 *The user shall be able to set the VTBI in k ml increments for volumes above x ml.*

4 Background

4.1 Interface refinement approaches

Several previous projects on formal refinement for interface design had different foci to our work. For example, the main focus of Bowen and Reeves [BR09] is on a description of the actions that the user can engage with and how these actions can be refined. The refinement process involves actions being replaced by more concrete actions in terms of more concrete structures. The refinement described by them is more akin to trace refinement. Although they argue that their interest is in ensuring that requirements are true of the more refined system, there is less concern with how the requirements are transformed through the levels of refinement. Duke and Harrison [DH95] are concerned with data refinement. They note that abstract representations of objects can be refined in two directions, into what is perceivable and into the architecture of the device. Darimont and van Lamsweerde [DL96] are concerned with requirements described in terms of the refinement of goals using the KAOS language. The interesting innovation in their proposal is that the formal refinement process may be achieved through a set of patterns. The approach we take here has most in common with the work of Yeganefard and Butler [YB11] who demonstrate a similar refinement process, in this case for control systems, using Event-B.

4.2 Event-B/Rodin framework

Event-B specifications are discrete models that consist of a state space and state transitions. A state includes constants and variables that describe the system. State transitions are specified as *events*. A specification of an event consists of two parts. The first one is a list of *guards*. Each guard is a predicate over the state variables and constants. The guards define the necessary conditions for the event to occur. The second part is a list of *actions* which describe how the state variables are modified as a result of event execution.

Specifications are structured in terms of *machines* and *contexts*. Machines specify the dynamic aspects of systems, whereas contexts specify its static aspects. A machine includes state variables and events. Invariant properties are expressed as machine invariants, i.e., predicates that must hold in all machine states. A context includes constants defined by a set of axioms. A machine may reference constants from the contexts it 'sees'.

Intuitively, machine execution means that one of the events, with all guards being true, is chosen. The machine variables are modified as specified by the actions of that event. The basic syntactic form of an event is given below, other features of Event-B are introduced when needed.

$$\boxed{\mathbf{Event}\quad E \mathrel{\widehat{=}}\quad \mathbf{when}\ \ G(v)\ \ \mathbf{then}\ \ T(v)\ \mathbf{end}}$$

Here v is a list of variables. $G(v)$ denotes the guards of E and $T(v)$ denotes the actions associated with E. A detailed description of Event-B can be found in [Abr10].

5 The requirement hierarchy

The informal requirements **R1** and **R2** from Section 3 provide a basis for the abstract specification of user requirements relevant to data entry. **R3** and **R4** are introduced in a later refinement.

5.1 Requirements R1 and R2

The requirement **R1** (*The flow rate for the pump shall be programmable*) is expressed as the following machine in Event-B. This abstract description simply requires that a variable called *data* has the attribute that it is programmable. The requirement asserts that *data* commences with a value named *source* and describes the event *programmable* as changing the value to *target*. The possible values of *data* are given as the set *Numbers*. All three constants, *Numbers*, *source* and *target*, are defined in the context ReqParams1 below. Nothing is contained in the requirement to indicate that it relates to flow rate. The requirement as specified could be applied to, e.g., VTBI.

MACHINE Reqs1 **SEES** ReqParams1
VARIABLES *data* **INVARIANTS** $data \in Numbers$
EVENTS
Initialisation **begin** $data := source$ **end**
Event *programmable* $\widehat{=}$ **begin** $data := target$ **end**
END

The invariant of Reqs1 simply gives typing of *data*. The initialisation event assigns the *source* value to it. Since the *programmable* event expresses an abstract requirement, its guard is assumed to be always true, and the **when** clause is omitted in the above specification.

The requirement **R2** (*The VTBI settings shall cover the range from v_{min} to v_{max} ml*) is specified in the context ReqParams1 which defines the corresponding constants *Min*, *Max*. It is assumed that *Max* exceeds *Min* and that *Min* is non-negative. The set constant (type) *Numbers* is assumed to be the interval $0 \,..\, Max$. The context defines a number of other constants: *RefValues*, *source* and *target*. It is assumed that the *source* value belongs to the interval *Numbers* and it is assumed that *target* is a member of the set of reference values (*RefValues*) that covers the required range of settings. At this stage, no other assumptions are made as to what these values are.

CONTEXT ReqParams1
CONSTANTS *Min* *Max* *Numbers* *RefValues* *source* *target*
AXIOMS

$Min \geq 0 \qquad Max > Min \qquad Numbers = 0 \,..\, Max$

$RefValues \subseteq Numbers \cap \{x | x \geq Min\} \qquad source \in Numbers \qquad target \in RefValues$

END

Because the **R1** requirement is specified in a non-operational form it is necessary to refine the machine. Informally, machine refinement means verifying three constraints. The first concerns event refinement: a concrete event must refine the corresponding abstract one (new events must refine an implicit event that does nothing). The second constrains new events: they must 'converge' (i.e., not run forever on their own). The third states that the concrete machine must not deadlock before the machine it refines.

The following refinement of Reqs1 provides guidance about how **R1** can be implemented. The operational version of **R1** has a number of new characteristics. Two new variables are introduced: *entry* and *disp*. Whether a number is being entered is indicated by *entry*, whereas *disp* gives the displayed value of the number entered. The initial state requires that *data* and *disp* are both initialised to the *source* value and *entry* is false, indicating that entry of the target number has not commenced. The new requirement decomposes the event representing **R1** into three events. The first one (*choose*) is used to elect to enter the target value, while the second one models the modification of the display value (this is not necessarily the data value). The final event is triggered when the display and target values are equal. At this step the data value is set to be equal to the display value and entry becomes false. This operational requirement indicates more about the programming process but says little about how the value is entered.

MACHINE Reqs11 **REFINES** Reqs1 **SEES** ReqParams1

VARIABLES $data \quad disp \quad entry$ **INVARIANTS** $disp \in Numbers \quad entry \in BOOL$

EVENTS

Initialisation **begin** $data := source \quad disp := source \quad entry := FALSE$ **end**

Event $choose \mathrel{\widehat{=}}$ **Status** anticipated

  when $entry = FALSE$ **then** $disp := data \quad entry := TRUE$ **end**

Event $modify \mathrel{\widehat{=}}$ **Status** anticipated **when** $entry = TRUE$ **then** $disp :\in Numbers$ **end**

Event $set \mathrel{\widehat{=}}$ **refines** *programmable*

  when $disp = target \quad entry = TRUE$ **then** $data := disp \quad entry := FALSE$ **end**

END

The machine Reqs11 specifies that *set* refines the abstract event *programmable* (intuitively, both events assign *target* to *data*). The other two events, *choose* and *modify*, are new. Rather than requiring their convergence, the specification assumes, as indicated by the keyword ‘anticipated’, that *choose* and *modify* will not run forever. If necessary, this assumption can be proven later.

5.2 Requirements R3 and R4

In the case of **R3** (*The user shall be able to set the VTBI in j ml increments for volumes below x ml*) and, similarly, **R4**, the requirements are expressed in a sufficiently concrete form to proceed directly to their operationalised versions. They are captured in the following context ReqParams11 which extends ReqParams1 by adding three relevant constants—*Threshold* (*x* in **R3** and **R4**), *j* and *k*—with three associated axioms:

CONTEXT ReqParams11 **EXTENDS** ReqParams1 **CONSTANTS** $Threshold \quad j \quad k$

AXIOMS

$Threshold \in Min+1\,..\,Max-1 \qquad j < Threshold \qquad k \leq Threshold$

$RefValues \subseteq \{x \cdot x > 0 \land j * x \leq Threshold \mid j * x\} \cup \{x \cdot x > 0 \mid Threshold + k * x\}$

END

The fourth axiom restricts the reference set (*RefValues*) to the values obtained using the increments *j* and *k*. This context is used by Reqs111 which is the same machine as Reqs11 otherwise:

MACHINE Reqs111 **REFINES** Reqs11 **SEES** ReqParams11 ....

The last step in the refinement of requirements has a more technical nature. It decomposes Reqs111 so that the assumptions about to the user behaviour are removed from the requirements for the pump interfaces. In particular, one guard ($disp = target$)) in the event *set* encompasses the notion of a target. Though the latter is relevant to the user behaviour, it would be meaningless to apply it to the pump interface. The decomposition introduces the machine Reqs111_Pump11 which replaces the constant *target* by a variable that represents the display value 'passed' to the user. The details are omitted here, since this does not affect the actual data entry.

6 The interface hierarchy

Having produced an operational but abstract definition of the requirements, the next stage is to make sense of the requirement in terms of the particular device that the developer wishes to certify. This section develops a refinement-based classification of user interfaces that is relevant for various modes of data entry in infusion pumps. Each refinement step introduces specific features, thereby creating a hierarchy of user interface classes. The aim is to verify safety requirements for the classes at the top of the hierarchy. If those requirements are satisfied at that level, then the interface classes at the lower levels are guaranteed to preserve them by construction.

There are a number of different data entry systems that are already used in infusion pumps [OTC11] and there is future scope for many more. To illustrate the approach, two types are considered in this paper: chevron based interfaces and five-key interfaces.

6.1 Chevron interfaces

In chevron based interfaces, the current data value is updated by pressing the 'up' (increase) and 'down' (decrease) chevron keys. These interfaces always include at least one 'up' and one 'down' chevron, however more chevrons could be used to speed up data entry. For example, a fast 'up' chevron may increase the current value by a larger amount compared to a slow 'up' one.

Interface specification. An abstract specification of the chevron based interface, machine Chevron_Entry1 defines two events for updating data values: *increase* and *decrease*. The first increases the current value (*disp*) by an unspecified (non-deterministically chosen) amount, while the second similarly decreases it. Both events are only enabled when the pump is in data entry mode ($entry = TRUE$). The *increase* event is specified as follows:

Event *increase* $\widehat{=}$ ... **when** $entry = TRUE$ **then** $disp :| disp' \in Numbers \land disp' \geq disp$ **end**

Verifying requirements. The abstract specification of the chevron based entry, Chevron_Entry1, is easily verified against the set of interface requirements. The verification is formally encoded as an assertion that the machine Chevron_Entry1 refines Reqs111_Pump11. In particular, the events *increase* and *decrease* both refine the *modify* event specified in the abstract machine relating to the operationalised requirement **R1**:

MACHINE Chevron_Entry1 **REFINES** Reqs111_Pump11 **SEES** ReqParams11 ...
Event *increase* $\widehat{=}$ **Status** anticipated **refines** *modify* ...
Event *decrease* $\widehat{=}$ **Status** anticipated **refines** *modify* ...
END

Interface refinement. As an example of how more concrete layers can be added to the interface hierarchy, we consider a chevron entry interface with two ‘up’ and ‘down’ keys. The slow ‘up’ and fast ‘up’ chevrons are modelled by the *up* and *UP* events, respectively. The *up* event (specified below) increases the current value by *delta* at least, whereas *UP* increases it by *Delta*.

```
Event  $up \widehat{=}$ ...
       when  $entry = TRUE$  then  $disp :| disp' \in Numbers \land disp' \geq min(\{disp + delta, Max\})$  end
```

The *dn* and *DN* events are specified similarly. In each case, *delta*/*Delta* is the minimum allowed update amount. This permits implementations of this interface where the actual update depends on the current data value. It is assumed that *Delta* is greater than *delta* to guarantee that the fast ‘up’ and ‘down’ chevrons are indeed faster than the slow ones:

```
CONTEXT  ChevronDefinitions11  EXTENDS  ReqParams11 ...
AXIOMS   $delta \in Numbers$   $Delta \in Numbers$   $delta > 0$   $Delta > delta$
END
```

This specification of a chevron based interface is a refinement of the interface with single ‘up’ and ‘down’ chevrons. In particular, the *up* and *UP* events refine the more abstract *increase* event, whereas *dn* and *DN* (omitted here) refine *decrease*:

```
MACHINE  Chevron_Entry11  REFINES  Chevron_Entry1  SEES  ChevronDefinitions11 ...
Event  $up \widehat{=}$  Status anticipated  refines $increase$ ...
Event  $UP \widehat{=}$  Status anticipated  refines $increase$ ...
END
```

6.2 Five-key interfaces

In the case of five-key interfaces, numbers are modified by combining up and down keys with movement of the cursor keys. The size of the increment or decrement is measured by the position of the cursor that can be manipulated using the left and right keys. The up and down keys normally operate on the single digit indicated by the cursor (e.g., up key modifies 5 to 6). However, there is a lot of variation [CGT^{+}12] in the behaviour of five-key interfaces at the ‘edges’ (e.g., when pressing up on the digit 9). Two main variations are to simply wrap the digit (e.g., 9 is modified to 0) and to modify the whole number according to the rules of arithmetic (e.g, 9 is modified to 0 and the digit to the left is increased by 1).

Interface specification. The context specified here describes the properties relevant to an abstract cursor and up/down keys. The constant *wrapCursor* indicates whether the cursor is wrapped at the edges (positions 0 and *maxPos*) when the corresponding cursor key is pressed. The axioms capture that the maximal position (*maxPos*) is greater than 0 and the limits in the cursor movement (*min_cursor* and *max_cursor*) which generally may depend on the current value. They also define the starting position of the cursor (*startPos*), the size of the increment or decrement (*delta*), and the signatures and behaviour of the *left* and *right* cursor keys. Further axioms concerning variations in the behaviour of the up and down keys are discussed below.

CONTEXT FiveKeyDefinitions **EXTENDS** ReqParams11

CONSTANTS

$maxPos \quad min\_cursor \quad max\_cursor \quad startPos \quad wrapCursor$

$delta \quad left \quad right \quad round\_up \quad round\_dn \quad mem\_up \quad mem\_dn$

AXIOMS

$maxPos > 0$

$min\_cursor \in Numbers \rightarrow 0\,..\,maxPos \qquad max\_cursor \in Numbers \rightarrow 0\,..\,maxPos$

$\forall x \cdot x \in Numbers \Rightarrow min\_cursor(x) \leq max\_cursor(x)$

$startPos \in min\_cursor(source)\,..\,max\_cursor(source) \qquad wrapCursor \in BOOL$

$delta \in 0\,..\,maxPos \rightarrow 1\,..\,10^{maxPos} \qquad \forall i \cdot i \in 0\,..\,maxPos \Rightarrow (delta(i) = 10^{i})$

$left \in 0\,..\,maxPos \rightarrow 0\,..\,maxPos \qquad right \in 0\,..\,maxPos \rightarrow 0\,..\,maxPos$

$\forall i, x \cdot i \in 0\,..\,maxPos \wedge x \in Numbers \Rightarrow$

$\qquad (i < max\_cursor(x) \Rightarrow left(i) = i + 1) \wedge$

$\qquad (i = max\_cursor(x) \wedge wrapCursor = TRUE \Rightarrow left(i) = 0) \wedge$

$\qquad (i = max\_cursor(x) \wedge wrapCursor = FALSE \Rightarrow left(i) = i)$

$round\_up \in Numbers \rightarrow BOOL \qquad round\_dn \in Numbers \rightarrow BOOL$

$\forall x \cdot x \in Numbers \Rightarrow (round\_up(x) = (x = Max \vee x = Min))$

$\forall x \cdot x \in Numbers \Rightarrow (round\_dn(x) = (x = Min))$

$mem\_up \in Numbers \rightarrow BOOL \qquad mem\_dn \in Numbers \rightarrow BOOL$

$\forall x \cdot x \in Numbers \Rightarrow (mem\_up(x) = (x < delta(maxPos)))$

$\forall x \cdot x \in Numbers \Rightarrow (mem\_dn(x) = (x > Max - delta(maxPos))) \quad \ldots$

END

The behaviour of the up/down and left/right keys is specified as the machine FiveKey_Entry1. It defines the relevant events *left*, *right*, *up* and *down*, all possible when the pump is in data entry mode (*entry* is true). The specifications of the *left* (given below) and *right* events are based on the functions defined above. They both modify *cursor* which represents the position of the cursor:

Event $left \mathrel{\widehat{=}} \ldots$ **when** $entry = TRUE$ **then** $cursor := left(cursor)$ **end**

The specifications of the *up* (given below) and *down* events can potentially be refined into both types of five-key interfaces described earlier: arithmetic and wrapping. They also take into account rounding behaviour at the minimal and maximal values allowed by the interface as specified by the predicates *round_up* and *round_dn*. For example, *round_up* states that such rounding can yield either the maximal value *Max* or the minimal value *Min*. Furthermore, both events permit the implementations of five-key interfaces with memory. For that the predicates *mem_up* and *mem_dn* define ranges of values that can be recalled from memory by pressing the up and down keys.

Event $up \mathrel{\widehat{=}} \ldots$ **when** $entry = TRUE$ **then**

$disp :| \, disp' \in Numbers \wedge$

$(disp' = disp + delta(cursor) \vee disp' = disp - 9 * delta(cursor) \vee round\_up(disp') \vee mem\_up(disp'))$

end

Verifying requirements. The FiveKey_Entry1 machine refines the set of requirements developed. In particular all four events, *left*, *right*, *up* and *down*, refine the same abstract event *modify* in the requirements:

MACHINE FiveKey_Entry1 **REFINES** Reqs111_Pump11 **SEES** FiveKeyDefinitions1 ...
Event *left* $\widehat{=}$ **Status** anticipated **refines** *modify* ...
END

The final stage of the process is to show how these interface classes, still abstract in the sense that they have potential to be implemented as a number of different chevron or five key interfaces, can be refined into a particular design.

7 Verification of concrete interfaces

The aim of this section is to show how an interface specification of a specific device is shown to satisfy user requirements. Ideally, such a specification would be provided by the manufacturer. Alternatively, it can be reverse engineered by interactively exploring the actual device [Thi07].

Having an interface hierarchy already shown to satisfy the user requirements, there is no need to verify a specific interface against those requirements. It suffices to show that the interface is an instance of some class in the hierarchy. This section demonstrates the principle for a concrete device with four chevrons as its number entry mechanism (Alaris GP Volumetric Pump [Car06]).

A specification of the Alaris number entry mechanism has been reverse engineered in PVS and SAL [MRO^{+}11]. The specification given below is its direct translation to Event-B. The purpose of using this translation is to demonstrate that our interface hierarchy can be used to verify the relevant user requirements for the independently developed specifications of concrete interfaces.

In the PVS and SAL versions, the behaviour of the Alaris chevrons (slow and fast up/down keys) is captured using functions that specify how the current value is modified by pressing each chevron. In Event-B, the corresponding functions, *alaris_up*, *alaris_dn*, *alaris_UP* and *alaris_DN*, are defined in the following context. It extends RealDefinitions which provides an Event-B model for the real numbers supported by the Alaris pump. The definitions of *alaris_dn*, *alaris_UP* and *alaris_DN* (omitted here) are similar to that of *alaris_up*:

CONTEXT AlarisDefinitions **EXTENDS** RealDefinitions
CONSTANTS *trim* *alaris_up* *alaris_dn* *alaris_UP* *alaris_DN* *init*
AXIOMS

$$\begin{aligned}
&trim \in \mathbb{Z} \rightarrow realq \quad alaris\_up \in real \rightarrow realq \quad init \in real\\
&\forall x \cdot (x < minAlaris \Rightarrow trim(x) = minAlaris) \land\\
&\quad (x > maxAlaris \Rightarrow trim(x) = maxAlaris) \land\\
&\quad (x \geq minAlaris \land x \leq maxAlaris \Rightarrow trim(x) = x)\\
&\forall x \cdot x \in real \Rightarrow (x < r100 \Rightarrow alaris\_up(x) = trim((floor(x*10)+r1)/10)) \land\\
&\qquad\qquad (x \geq r100 \land x < r1000 \Rightarrow alaris\_up(x) = trim(x+r1)) \land\\
&\qquad\qquad (x \geq r1000 \Rightarrow alaris\_up(x) = trim((floor(x/10)+r1)*10)) \quad \ldots
\end{aligned}$$

END

The machine Alaris_Rate1 specifies the behaviour of the four chevrons when entering infusion rates. This behaviour is described by the events *up*, *dn*, *UP* and *DN*. E.g., *up* is specified below:

Event *up* $\widehat{=}$ **Status** anticipated **when** $rmode = TRUE$ **then** $display := alaris\_up(display)$ **end**

Here the *display* variable represents the displayed rate and *rmode* indicates whether the pump is in the rate entry mode.

Now, the demonstration that the Alaris rate entry interface is an instance of the class of chevron interfaces specified in Section 6.1 boils down to proving refinement between Chevron_Entry11 and Alaris_Rate1. For such a proof, the generic parameters (such as *delta* or *Threshold*) specified in ChevronDefinitions11 must be instantiated with the concrete values from the Alaris specification (context AlarisDefinitions). This instantiation is specified as the folowing context:

```
CONTEXT  ChevronAlarisParams  EXTENDS ChevronDefinitions11    AlarisDefinitions
AXIOMS
           Min = minAlaris      Max = maxAlaris      Threshold = r100
           j = r01      delta = j      k = r1      delta = j      source = init
END
```

Furthermore, the four chevron events in Alaris_Rate1 must refine the corresponding events in Chevron_Entry11, and the invariants of Alaris_Rate1 must include a ‘glueing’ invariant that specifies the connection between the state spaces of both machines:

```
MACHINE  Alaris_Entry1  REFINES Chevron_Entry11  SEES ChevronAlarisParams    ...
INVARIANTS  (data = rate) ∧ (disp = display) ∧ (entry = rmode)    ...
EVENTS
Event  up ≙  Status anticipated  refines up
when  rmode = TRUE   with  disp′ : disp′ = display′   then  display := alaris_up(display)  end ...
END
```

8 Conclusions

The paper has demonstrated how Event-B can be used to support manufacturers as they aim to demonstrate that regulator requirements are satisfied by their products. All the refinements described have been proven using the Rodin platform. The refinement hierarchies thus developed for requirements and user interfaces enable developers to trace the regulator requirements down to the specialised classes that match the physical characterisation of their device. Such an approach fits well with the FDA pre-market review process which involves providing evidence that a new device is ‘substantially equivalent’ to already approved and legally marketed medical devices.

This work is in its early stages but it would be envisaged that demonstration that the device satisfies these requirements would involve the development of specification fragments that provably demonstrate that the requirements are satisfied. It then remains as an open question as to how it could be demonstrated that these components are consistent with each other and how they might fit into a larger specification. This is future work. It would aim to explore work on composition [SB09] and product lines [GPS09] in Event-B being carried out at Southampton. The advantage of using Event-B is that the approach is tool supported. It is feasible to consider the possibility that standard refinement processes such as these can be made easier for developers to use.

Acknowledgements: This work was partly funded by the CHI+MED research project on the design and safe use of interactive medical devices (UK EPSRC Grant EP/G059063/1).

Bibliography

[Abr10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[BR09] J. Bowen, S. Reeves. Refinement for user interface designs. *Formal Aspects of Computing* 21:589–612, 2009.

[Car06] Cardinal Health Inc. Alaris GP Volumetric Pump: directions for use. Technical report, Cardinal Health, 1180 Rolle, Switzerland, 2006.

[CGT+12] A. Cauchi, A. Gimblett, H. Thimbleby, P. Curzon, P. Masci. Safer ”5-key” number entry user interfaces using differential formal analysis. In *Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers*. BCS-HCI ’12, pp. 29–38. British Computer Society, Swindon, UK, 2012.

[DH95] D. J. Duke, M. D. Harrison. Mapping user requirements to implementations. *Software Engineering Journal* 10(1):13–20, 1995.

[DL96] R. Darimont, A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings 4th ACM Symposium on the Foundations of Software Engineering (FSE’03)*. Pp. 179–190. ACM Press, 1996.

[GPS09] A. Gondal, M. Poppleton, C. Snook. Feature composition - towards product lines of Event-B models. In *1st International Workshop on Model-Driven Product Line Engineering (MD-PLE’09)*. CTIT Workshop Proceedings, 2009. http://eprints.soton.ac.uk/267547/

[JPJ06] R. Jetley, S. Purushothaman Iyer, P. Jones. A formal methods approach to medical device review. *Computer* 39(4):61–67, 2006. doi:10.1109/MC.2006.113

[LF09] W. Lin, X. Fan. Software Development Practice for FDA-Compliant Medical Devices. In *International Joint Conference on Computational Sciences and Optimization, 2009. CSO 2009.* Volume 2, pp. 388–390. 2009. doi:10.1109/CSO.2009.191

[MAC+13] P. Masci, A. Ayoub, P. Curzon, M. Harrison, I. Lee, O. Sokolsky, 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).* ACM Press, 2013.

[MRO+11] P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, H. Thimbleby. On formalising interactive number entry on infusion pumps. *Electronic Communications of the EASST* 45, 2011.

[OTC11] P. Oladimeji, H. Thimbleby, A. Cox. Number entry and their effects on error detection. In Campos et al. (eds.), *Interact 2011*. Lecture Notes in Computer Science 6949, pp. 178–185. Springer Verlag, 2011.

[SB09] R. Silva, M. Butler. Supporting reuse mechanisms for developments in Event-B: Composition. Technical report, University of Southampton, 2009.

[Thi07] H. Thimbleby. Interaction walkthrough: evaluation of safety critical interactive systems. In Doherty and Blandford (eds.), *Interactive Systems: Design, Specification and Verification*. Lecture Notes in Computer Science 4323, pp. 52–66. Springer Verlag, 2007.

[YB11] S. Yeganefard, 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.