Model-based development of the Generic PCA infusion pump user interface prototype in PVS

Paolo Masci[1], Anaheed Ayoub[2], Paul Curzon[1],
Insup Lee[2], Oleg Sokolsky[2], and Harold Thimbleby[3]

[1] Queen Mary University of London, UK
{paolo.masci,paul.curzon}@eecs.qmul.ac.uk

[2] University of Pennsylvania, PA, USA
{anaheed,lee,sokolsky}@cis.upen.edu

[3] Swansea University, UK
harold@thimbleby.net

Abstract. A realistic user interface is rigorously developed for the US Food and Drug Administration (FDA) Generic Patient Controlled Analgesia (GPCA) pump prototype. The GPCA pump prototype is intended as a realistic workbench for trialling development methods and techniques for improving the safety of such devices. A model-based approach based on the use of formal methods is illustrated and implemented within the Prototype Verification System (PVS) verification system. The user interface behaviour is formally specified as an executable PVS model. The specification is verified with the PVS theorem prover against relevant safety requirements provided by the FDA for the GPCA pump. The same specification is automatically translated into executable code through the PVS code generator, and hence a high fidelity prototype is then developed that incorporates the generated executable code.

Keywords: Formal methods; Model-based development; Medical devices; User interface prototyping.

1 Introduction and motivation

Infusion pumps are medical devices used to deliver drugs to patients at precise rates and in specific amounts. The current infusion pumps incorporate sophisticated software, of around tens of thousands of lines of program code [9]. This complexity may make infusion pumps flexible and configurable, but it introduces new risks as software correctness is hard to verify. Traditional manual verification and validation activities based on manual inspection, code walkthroughs and testing are insufficient for catching bugs and design errors in such complex software. Unfortunately there are currently no widely accepted techniques for development and verification of software for medical devices, nor standard guidelines to ensure that a device meets given safety requirements [8].

Numerous adverse events have been reported that are associated with infusion pumps. Reports from the US Food and Drug Administration (FDA) show that some of these incidents are due to use errors and software failures caused by poor software design [5]. Because of this, several device recalls have been

issued: for instance, 87 models of infusion pump, affecting all infusion pump manufacturers, were recalled over 2005 to 2009 in the US [5].

The FDA is promoting the development of so-called Generic Infusion Pump (GIP) models and prototypes as a way to demonstrate how rigorous development methods can substantially improve confidence in the correctness of software. For instance, in [1], the FDA presents a research prototype for generic design of Patient Controlled Analgesia (PCA) pumps, called the Generic PCA (GPCA) pump. We explain PCA pumps more fully below.

The GPCA itself is not yet a real medical device. However, because its functionalities and details closely resemble those of a real medical device, it can be used as a realistic workbench. Successful application of methods and tools to the GPCA prototype should indicate that they are viable for commercial devices.

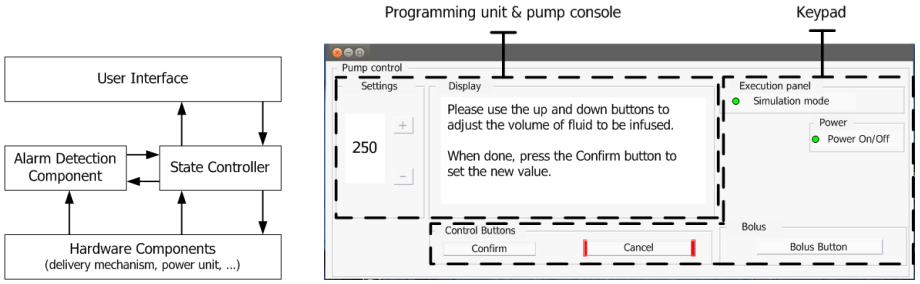
The importance of user interface design is well understood by regulators [21]. However, hardly any concrete examples of model-based development of user interfaces have been explored that take account of human factors or human factors engineering. In our previous work, we illustrated how verification tools could be used to verify the design of commercial infusion pump user interfaces against properties that capture human factors concerns [4, 6, 13, 14] and safety requirements [12]: potential issues were identified precisely, and verified design solutions that could *fix* the identified issues were presented. This work builds on our previous work, and extends it by introducing a model-based development approach for rapid prototyping of medical device user interfaces that are verified against given safety requirements. The approach presented in this paper is illustrated through the development of core parts of the user interface of the GPCA.

Contributions. The main contribution of this paper is the detailed model-based development of a data entry system for the GPCA user interface within Prototype Verification System, PVS [18]. The specification of the data entry system of the GPCA user interface incorporates safety features that can mitigate use errors, and the specification is formally verified against safety requirements provided by the FDA within PVS, a standard system commonly used for this purpose. The verified model is then automatically transformed into executable code through the PVS code generator, and the generated code is then incorporated in a prototype that can be executed.

2 Related work

In this paper, the model-based approach is implemented using the Prototype Verification System (PVS) [18]. PVS is a state-of-the-art verification tool that provides an expressive specification language based on higher-order logic, a language mechanism for theory interpretation [19], a verification engine based on automated theorem proving, and a code generator for automatic translation of PVS specifications into executable code [23].

PVS is only one approach of course, and other tools could have been used to develop the prototype. Our choice was guided by pragmatics linked to best current development practices — the need to specify safety requirements inde-



(a) GPCA architecture (b) The original GPCA user interface in Matlab

Fig. 1. Schematic of the FDA's Generic PCA infusion pump prototype.

pendently, expressiveness of the specification language (here, PVS), and automated code generation from verified specifications. In [10], different tools are used to generate a prototype of the GPCA pump controller with a model-based approach: safety requirements are formalised as properties of a Uppaal model of the GPCA controller, and the Times code generator is used for translating the model into platform-independent C code. In [27], a model of the Generic Insulin Infusion pump controller are developed in Event-B using the Rodin platform. The model is verified against selected safety requirements related to timing issues. The development of a prototype from the verified model was not in the scope of that work. In [3], model-based development is used to generate a runtime software monitor that validates the behaviour of an insulin infusion pump against safety requirements. Petri Nets are used to specify the behaviour of the monitor, and then the specification is manually translated into Java code.

No prior work addresses model-based development of realistic user interfaces for infusion pumps.

3 The Generic PCA (GPCA) pump

PCA pumps are a class of infusion pump used to administer pain relief medication. They are employed for self-administration where a patient is able to request pain relief in controlled amounts when they need it. The patient interacts with the PCA pump using a single button, which is used to request additional predefined doses of drug. The intended infusion parameters are programmed in advance by clinicians. In the current generation of infusion pumps, clinicians program infusion parameters by interacting with buttons on the user interface.

The FDA has developed a Matlab Simulink/Stateflow model of the Generic PCA (GPCA) pump that captures the core functionalities of PCA pumps in general. The GPCA model has a layered architecture (see Figure 1(a)). The top layer is the user interface, which presents the state of the infusion pump and allows users to program infusion parameters. The software controller is the middle layer in the architecture, and includes two components: a state controller and alarm detection component. The state controller drives the drug administration process and supervises communication among the modules of the GPCA

pump. The alarm detection component handles alarms and warnings. The lowest layer models the hardware components, such as the delivery mechanism (peristalitic motors and air bubble sensors, etc) and power unit (includind the battery charger, etc).

The GPCA user interface, as provided with the original model [21], has the layout shown in Figure 1(b). The user interface includes the following elements: a *programming unit and pump console*, which renders information about the pump state and allows users to set infusion parameters; and a *keypad*, which allows users to send commands to the pump. Human factors were not considered when developing this original user interface [21], as the user interface was used primarily for development and debugging purposes.

3.1 GPCA safety requirements

The FDA has released an initial set of 97 GPCA safety requirements [1]. They are formulated in natural language, and grouped into 6 main categories: *infusion control*, which are dedicated to safety features and constraints that can mitigate hazards resulting from incorrectly specified infusion parameters (e.g., flow rate too high or too low); *user interface*, which describe constraints on user interface functionalities that can help avoid accidental modification of infusion parameters; *error handling*, which are dedicated critical alarming conditions; *drug error reduction*, which define drug library functionalities; *power and battery operations* and *system environment*, which are dedicated to constraints on operating conditions.

The GPCA safety requirements describe essential safety features and constraints that guarantee a minimum level of pump safety. The requirements were obtained by reasoning about mitigation actions that could contrast identified hazards associated with PCA pumps, as well as related causes of the identified hazards. For instance, an identified hazard of PCA pumps is overinfusion, and one of the causes is that the programmed flow rate is too high. A suggested mitigation for this hazard is to make the flow rate programmable within given rate bounds only. Starting from this suggested mitigation, corresponding GPCA safety requirements are then formulated that can help check the mitigation barrier in the pump.

The GPCA safety requirements were designed on the basis of a preliminary hazard analysis for the controller of the pump. We found that almost half of the requirements can be related to user interface functionalities, and correctly capture basic human factors concerns. However, a hazard analysis specifically addressing user interface functionalities is needed to cover a more complete set of aspects related to human factors. We are currently starting this hazard analysis. Some examples of safety features and constraints that are currently *not* considered in the GPCA safety requirements and can potentially make the user interface design safer follows.

Illegal keying sequences shall be blocked during interactive data entry. An illegal keying sequence is a sequence of key clicks resulting in an illegal value (e.g., a value out of range) or illegal number format (e.g., a number with two

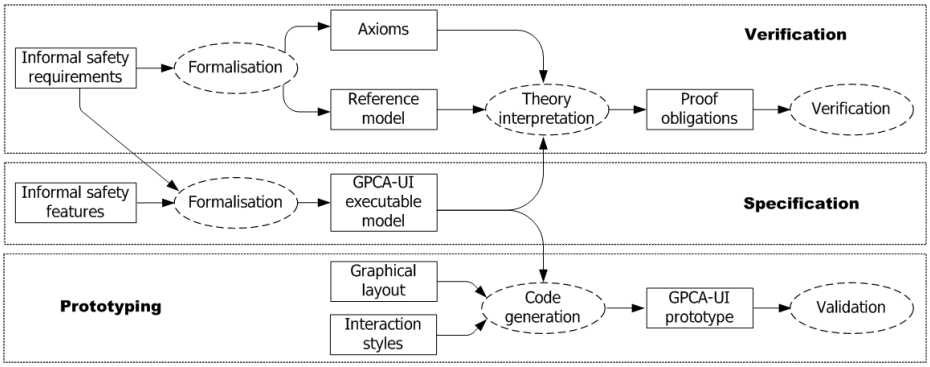


Fig. 2. The adopted model-based development approach.

decimal dots). Blocking an illegal keying sequence means that interaction is halted when a key click results in an illegal keying sequence, and feedback is provided to the user. Results presented in [4] and [25] show that this safety feature can create useful mitigation barriers against keying slip errors.

Numbers rendered on the display shall follow the ISMP rules [7]. The Institute for Safe Medical Practices (ISMP) promoted the adoption of two basic rules to design correctly formatted dosage values: leading zeros (e.g., 0.1 mL) are always required for fraction dose values; and trailing zeros (e.g., 1.0 mL) are always avoided. These rules are distilled from best practice (e.g., 1.0 may be misread as 10) and aim to reduce medication errors through standardised formatting and clear presentation.

Numbers rendered on displays shall follow the NPSA recommendations [26]. The UK National Patient Safety Agency (NPSA) recommends adherence to the following guidelines to facilitate a correct identification of dosage values, including: digits after the decimal point are rendered in smaller font size; the visual salience of the decimal point is increased; using "TallMan" lettering for names and units.

4 Development of the GPCA user interface

A GPCA user interface (hereafter, GPCA-UI) prototype is now developed using a model-based approach. Within model-based development approaches, models are used as primary artefacts during the whole development cycle: they present a design specification that can be checked against given requirements, and then code generation techniques are used to transform the model into a concrete implementation for a specific platform. The adopted model-based development approach consists of the following phases (as shown in Figure 2):

Specification. A GPCA-UI model is specified that defines the interactive behaviour of the GPCA user interface. The specification is given in the form of an *executable model*, that is a model whose specification is rich enough that it can be systematically translated into a concrete code implementation.

Verification. The developed GPCA-UI model is verified against a formalisation of selected GPCA safety requirements. This is done through an approach based on *theory interpretation*. It is based on the idea of formalising safety requirements as axioms of an abstract model (which we call the *reference model*); proof obligations that need to be verified on the GPCA-UI model are automatically generated by mapping functionalities of the abstract model into functionalities of the GPCA-UI model; a formal verification is then performed to discharge the generated proof obligations. With this approach, safety requirements can be developed independently from the GPCA-UI model.

Prototyping. A user interface prototype is developed. The prototype incorporates software code automatically generated from the verified GPCA-UI model through code transformation. The prototype has a back-end that defines the functionalities of the GPCA-UI, and a front-end that defines the visual appearance of the GPCA-UI. The back-end is executed within a verified execution environment to ensure that correctness properties verified on the formal specification are preserved at run-time when executing the generated code. The front-end just renders information returned by the back-end. The prototype can be used for validation purposes. The prototype implementation in PVS is further elaborated in the following sub-sections.

4.1 Specification

The model is developed as a finite state machine. The state of the state machine defines information observable on the GPCA-UI (e.g., values shown on a display) and internal variables (e.g., values held by timers). State transitions of the state machine define interactive functionalities activated by the operator (e.g., button clicks) and internal events generated by the GPCA-UI (e.g., timer events).

The GPCA-UI model includes the typical functionalities provided by the current generation of commercial PCA pump user interfaces. Due to space limitations, only a qualitative description of the functionalities included in the model is provided here without going into the specific details of the PVS model. The full PVS model can be found at [2].

The GPCA-UI programming unit specifies the behaviour of a "5-key" number entry [4, 17], as widely used in commercial infusion pumps. A different choice could have been made (chevron keys, number pad, or others). Two functions (*up* and *down*) edit the entered value by an increment step. The increment step is proportional to the position of a cursor. Two functions (*left* and *right*) edit the position of the cursor. The accuracy of the entered value is limited to two decimal digits, and legal values are below between 0 and 99999. These limits reflect those of commercial PCA pumps. Whenever these limits are violated, interaction is halted and an alert message displayed.

The GPCA keypad specification defines the basic behaviour of typical commands made available to the operator to control the state of the pump: turn the pump on and off; start and stop an infusion. Additional functionalities not implemented in this first version of the model include: edit infusion parameters; view pump status; deliver an additional limited amount of drug upon demand.

The model developed includes a specification of the communication protocol with the GPCA controller developed by Kim *et al* in [10]. In the current version, the protocol specification includes the sequence of commands to boot-strap the pump controller.

4.2 Verification

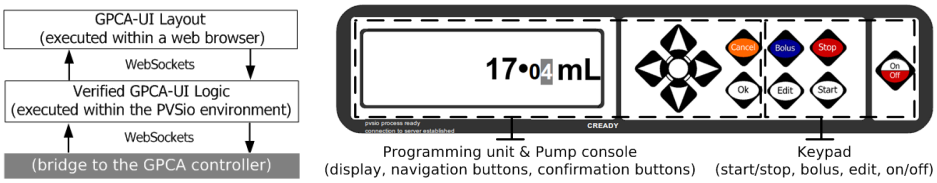
Within the verification approach, safety requirements formulated in natural language are formalised as predicates (see Figure 2). These predicates define the functionalities of a logic-based model, which we call the *reference model*, which encapsulates the semantics of the safety requirements by construction. The reference model is used for the verification of the GPCA-UI model by means of a technique called *theory interpretation* [19], which is a verification approach based on the idea of establishing a mapping relation between an abstract model and a concrete model. The mapping relation is used to systematically translate properties that hold for the abstract model into proof obligations that need to be verified for the concrete model. In our case, the abstract model is the reference model, and the concrete model is the GPCA-UI model. Hence, safety requirements encapsulated in the specification of the reference model are systematically translated into proof obligations for the GPCA-UI model. Being able to discharge the generated proof obligations through formal proof is a demonstration that the GPCA-UI model meets the safety requirements. The GPCA-UI specification developed is then formally verified against the following GPCA requirements that are relevant to the data entry system.

GPCA 1.1.1 *The flow rate of the pump shall be programmable.*
GPCA 1.1.2 *At a minimum, the pump shall be able to deliver primary infusion at flows throughout the range f_{min} and f_{max} mL per hour.*
GPCA 1.3.1 *The volume to be infused settings shall cover the range from v_{min} to v_{max} mL.*
GPCA 1.3.2 *The user shall be able to set the volume to be infused in j mL increments for volumes below x mL.*
GPCA 1.3.3 *The user shall be able to set the volume to be infused in k mL increments for volumes above x mL.*

Example. Requirement 1.3.1 is formalised and verified to exemplify the verification approach. A logic expression is created by extracting the relevant concepts presented in the textual description: *VTBI settings range* (where VTBI means volume of drug to be infused), v_{min} and v_{max}. As shown in Listing 1.1, these concepts are used to define an uninterpreted predicate `vtbi_settings_range` in PVS higher-order logic, and two symbolic constants `v_min` and `v_max` of type non-negative real numbers. The state of the reference model is specified with a new uninterpreted type, `ui_state`.

Listing 1.1. Part of the Reference Model

```
ui_state: TYPE
vtbi_setting_range(vmin,vmax: noneg_real)(st:ui_state): boolean
vmin,vmax: noneg_real
```



(a) Architecture (b) GPCA-UI layout (HTML & JavaScript)

Fig. 3. The GPCA-UI prototype.

The predicate and constants are then encapsulated in the reference model by formulating a property that always holds for the reference model (i.e., an *axiom*). In PVS, axioms are boolean expressions annotated with the `AXIOM` keyword. Axioms can be formalised in a way that facilitates proof by structural induction. For the considered example, the formalisation is as follows (see Listing 1.2). The initial state of the reference model (predicate `init?`) satisfies `vtbi_settings_range` (the *induction base*); given a state `st0` of the reference model that satisfies `vtbi_settings_range`, any state `st1` reachable from `st0` through a transition function (`trans`) of the reference model satisfies `vtbi_settings_range` (the *induction step*).

Listing 1.2. Axiom used to specify requirement R1

```
R1_Axiom: AXIOM
FORALL (st, st0, st1: ui_state):
(init?(st) IMPLIES vtbi_settings_range(vmin,vmax)(st)) AND
 ((vtbi_settings_range(vmin,vmax)(st0) AND trans(st0, st1))
     IMPLIES vtbi_settings_range(vmin,vmax)(st1))
```

A relation is then defined that specifies how `vtbi_settings_range` is mapped into the GPCA-UI model. In this case, the relation maps `vtbi_settings_range` into a function that checks the vtbi range supported by the GPCA-UI model (the second `LAMBDA` function in Listing 1.3). Through this mapping, PVS is able to automatically generate proof obligations that must be verified on the GPCA-UI model in order to demonstrate compliance with the reference model (and, hence, show that the safety requirement is met). The syntax for specifying a theory interpretation in PVS is that of a PVS theory importing clause (keyword `IMPORTING` followed by the model name, `reference_model_th` in this case) with actual parameters specifying the mapping relation (a list of substitutions provided within double curly brackets). Listing 1.3 gives a snippet of the PVS theory interpretation specified for the considered requirement: it states that the uninterpreted state of the reference model (`ui_state`) is mapped onto the state of the GPCA-UI model (`gpcaui_state`). The uninterpreted predicate that recognises the initial state of the reference model (`init?`) is mapped into the interpreted predicate that recognises the initial GPCA-UI concrete model state (`gpcaui_init?`). The uninterpreted predicate that identifies the set of transitions of the reference model (`trans`) is mapped into a function that enumerates the transition functions of the GPCA-UI concrete model (the first `LAMBDA` expres-

sion in the specification snippet shown in Listing 1.3; in the expression, `st_prime` identifies the next state obtained after applying a transition function).

Listing 1.3. Theory interpretation

```
IMPORTING reference_model_th
 {{ ui_state := gpcaui_state,
    init? := gpcaui_init?,
    trans := LAMBDA (st, st_prime: gpcaui_state):
               st_prime = click_up(st) OR %...
    vmin := 0, vmax := 99999,
    vtbi_settings_range
          := LAMBDA (vmin, vmax: nonneg_real)(st: gpcaui_state):
               vmin <= display(st) AND display(st) <= vmax
                AND vmin <= vtbi(st) AND vtbi(st) <= vmax }}
```

Given this theory interpretation, PVS automatically generates the proof obligation in Listing 1.4, which then needs to be discharged. The proof obligation requires we show that, for all reachable states, it is always true that the display and the VTBI range have values between `v_min` and `v_max` (0–99999 in this case).

Listing 1.4. Proof obligation

```
IMP_reference_model_th_R1_Axiom_TCC1: OBLIGATION
  FORALL (st, st0, st1: gpcaui_state):
    (gpcaui_init?(st) IMPLIES
      0 <= st`display AND st`display <= 99999
          AND 0 <= st`vtbi AND st`vtbi <= 99999)
   AND ((0 <= st0`display AND st0`display <= 99999 AND 0 <= st0`vtbi
           AND st0`vtbi <= 99999 AND st1 = click_up(st0) OR %...)
       IMPLIES 0 <= st1`display AND st1`display <= 99999
                   AND 0 <= st1`vtbi AND st1`vtbi <= 99999);
```

The generated proof obligation can be discharged within the PVS theorem prover thanks to using implicit subtype constraints [24] declared for the `vtbi` type and the `display` type in the GPCA-UI model. (In PVS, implicit subtype constraints are made explicit by using the command `typepred`.) After making subtype constraints explicit, the proof can be completed in less than a second with *assert*, a predefined decision procedure of the PVS theorem prover that simplifies expressions using decision procedures for equalities and linear inequalities. Alternatively, PVS can perform the proof automatically in seconds with its command `grind`, a powerful predefined decision procedure that repeatedly applies definition expansion, propositional simplification, and type-appropriate decision procedures.

4.3 Prototyping

An interactive GPCA-UI prototype is now presented that incorporates the verified PVS specification. The utility of the prototype is that it allows validating the behaviour of the generated code, and verifying aspects of the UI that are not formalised in the specification (e.g., the guidelines illustrated in Section 3.1). Additionally, the prototype can be used by formal methods experts to engage with domain experts such as human factors specialists.

The GPCA-UI prototype can be downloaded at [2]. The prototype architecture is split into a front-end and a back-end, as shown in Figure 3(a). The front-end is deployed on a tablet, which makes it possible to do realistic interaction with the buttons on the user interface. The back-end is deployed on a server with the PVSio [15] prototyping environment. Code automatically generated from the PVS specification is executed exclusively on the back-end within the Lisp execution environment of PVS. This gives us confidence that the safety requirements verified for the GPCA-UI specification are preserved when executing the Lisp code automatically generated from the verified GPCA-UI specification.

The design choices for the front-end and back-end are valid for the illustrative purpose of this work, that is to generate a realistic user interface for a research prototype from a verified model:

The GPCA-UI front-end is responsible for the visual appearance of the GPCA-UI. A "5-key" number entry layout based on navigation buttons has been chosen because it is widely used in the current generation of commercial PCA pumps. A different choice could have been made (chevron keys, number pad, or others). The front-end is executed within a web browser, which very conveniently allows using HTML code to render the GPCA-UI layout and using JavaScript to capture user interactions with buttons and translate them into function calls for the PVSio environment executed on the back-end. This translation from user actions to commands is performed on the basis of mappings between interactive areas of the GPCA-UI and function names in the PVS specification of the GPCA-UI. An example mapping that has been defined is the following: a button click of the *up* arrow key triggers a call to function `click_up` in the PVS specification. JavaScript is used to render the user interface state returned by PVSio: it renders numbers in the GPCA-UI display in a way that is compliant with the ISMP and NPSA recommendations given in Section 3.1. This can be validated through visual inspection. Note that the developed HTML and JavaScript code do not add new behaviours to the GPCA-UI — they are just used to send commands and render the state returned by the back-end.

The GPCA-UI back-end is responsible for the interactive behaviour of the GPCA-UI. The core of the back-end is the PVSio [15] prototyping environment. It provides an interactive command prompt that accepts higher-order logic expressions. The expressions are evaluated in the Lisp execution environment of PVS: Lisp code is generated on-demand, and then executed. A result is returned symbolically every time an expression is evaluated. The returned expression is a GPCA-UI model state, in this case. For instance, writing the expression `click_up(init)` in the PVSio command prompt results in the evaluation of function `click_up` of the GPCA-UI specification starting from state `init`. Lisp code is automatically generated, the function is executed, and a new state returned. A web-server presents the PVSio command prompt as a service of the GPCA-UI back-end. WebSockets, a standard protocol for bidirectional low-latency communication between two endpoints over a TCP connection, are used to enable communication between the front-end and the back-end.

5 Conclusions

Making medical devices safer involves a constructive dialogue among stakeholders (manufacturers, regulators, clinicians), and a verification approach based on these generic models can help to make this dialogue precise, as well as having the advantages of being computerized and runnable.

We have presented a model-based development approach for building a realistic user interface for the GPCA pump prototype. Although the user interface is a research prototype and not a real medical device, the functionalities and level of detail used in the specification are very similar to those of commercial PCA pumps. Because of this, it is evident that the specification can be used as a realistic workbench, and the model-based developed approach used can in principle be used as part of the development of real medical device user interfaces.

The model-based approach incorporates several concepts promoted by medical device regulators and which should be directly applicable to the development of real medical devices. For instance, in [21] and [9], the FDA Office of Science and Engineering Lab (OSEL) engineers have promoted the formalisation of safety requirements as generic models that can be used for verification of real devices.

The model-based approach introduced here has some limitations that need to be considered and should be the subject of further work: the formalisation of safety requirements as predicates does not allow a formal verification of the consistency of the safety requirements (e.g., contradictory safety requirements can be formalised); the verification technique based on theory interpretation allows the creation of mappings that are syntactically correct but semantically wrong (e.g., visible display elements of the reference model can be mapped into state variables of the concrete model that are not rendered on the display); code generation is limited to Lisp code (new code generators that translate PVS models into C [20] and Java [11] are still under development). Further work is needed to demonstrate the approach for the entire user interface (we have illustrated the approach just for the data entry system). We have started to explore solutions to these limitations in [22] and [16].

Acknowledgements. This work is supported in part by the EPSRC (CHI+MED, EP/G059063/1), NSF CNS-1035715, and NSF CNS-1042829.

References

1. GPCA Hazards and Safety Requirements. `http://rtg.cis.upenn.edu/gip.php3`.
2. The GPCA-UI Prototype. `http://tinyurl.com/QMUL-GPCA-UI`.
3. S. Babamir. Constructing a model-based software monitor for the insulin pump behavior. *Journal of Medical Systems*, 36, 2012.
4. A. Cauchi, A. Gimblett, H. Thimbleby, P. Curzon, and P. Masci. Safer "5-key" number entry user interfaces using differential formal analysis. In *BCS-HCI '12*.
5. Center for Devices and Radiological Health, U.S. Food and Drug Administration. *White Paper: Infusion Pump Improvement Initiative*, 2010.
6. M.D. Harrison, J. Campos, and P. Masci. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering, Springer-Verlag London*, 2013.

7. Institute for Safe Medication Practices (ISMP). Guidelines for standard order sets. `http://www.ismp.org/tools/guidelines`.
8. R. Jetley, C. Carlos, and S. Purushothaman Iyer. A case study on applying formal methods to medical devices. *International Journal on Software Tools for Technology Transfer*, 5(4):320–330, 2004.
9. R. Jetley and P. Jones. Safety requirements based analysis of infusion pump software. *IEEE RTSS/SMDS*, 2007.
10. 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 *ACM international conference on Embedded software*, EMSOFT '11. ACM, 2011.
11. L. Lensink, S. Smetsers, and M. van Eekelen. Generating Verifiable Java Code from Verified PVS Specifications. *NASA Formal Methods*, pages 310–325, 2012.
12. P. Masci, A. Ayoub, P. Curzon, M.D. Harrison, I. Lee, and H. Thimbleby. Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *EICS2013*. ACM Digital Library, 2013.
13. P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby. On formalising interactive number entry on infusion pumps. *ECEASST*, 45, 2011.
14. P. Masci, R. Rukšėnas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby. The benefits of formalising design guidelines: A case study on the predictability of drug infusion pumps. *Innovations in Systems and Software Engineering, Springer-Verlag London*, 2013.
15. C. Muñoz. Rapid prototyping in PVS. Technical Report NIA Report No. 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, 2003.
16. P. Oladimeji, P. Masci, P. Curzon, and H. Thimbleby. PVSio-web: a tool for rapid prototyping device user interfaces in PVS. To appear in *FMIS2013*, 2013.
17. P. Oladimeji, H. Thimbleby, and A. Cox. Number entry interfaces and their effects on error detection. In *INTERACT'11*, Berlin, Heidelberg, 2011. Springer-Verlag.
18. S. Owre, S. Rajan, J. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In *CAV96*, volume 1102 of *LNCS*. Springer Berlin Heidelberg, 1996.
19. S. Owre and N. Shankar. Theory Interpretations in PVS. Technical Report SRI-CSL-01-01, Computer Science Lab, SRI International, Menlo Park, CA, 2001.
20. S. Owre and N. Shankar. A brief overview of PVS. In *TPHOLs 2008*, volume 5170 of *LNCS*, pages 22–27, Montreal, Canada, August 2008. Springer-Verlag.
21. A. Ray, R. Jetley, P. Jones, and Y. Zhang. Model-based engineering for medical-device software. *Biomedical Instrumentation & Technology*, 44(6):507–518, 2010.
22. R. Rukšėnas, P. Masci, M.D. Harrison, and P. Curzon. Developing and verifying user interface requirements for infusion pumps: a refinement approach. To appear in *FMIS2013*, 2013.
23. N. Shankar. Efficiently Executing PVS. Technical report, Computer Science Laboratory, SRI International, Menlo Park, 1999.
24. N. Shankar and S. Owre. Principles and pragmatics of subtyping in PVS. In *WADT '99*, volume 1827 of *LNCS*, pages 37–52. Springer-Verlag, 1999.
25. H. Thimbleby and H. Cairns. Reducing number entry errors: Solving a widespread, serious problem. *Journal Royal Society Interface*, 7(51), 2010.
26. UK National Patient Safety Agency. Design for patient safety: A guide to the design of electronic infusion devices, 2010.
27. H. Xu and T. Maibaum. An Event-B Approach to Timing Issues Applied to the Generic Insulin Infusion Pump. In *Foundations of Health Informatics Engineering and Systems*, volume 7151 of *LNCS*. Springer Berlin Heidelberg, 2012.