Stepwise Formal Modelling and Reasoning of Insulin Infusion Pump Requirements

Neeraj Kumar Singh[1(✉)], Hao Wang[2], Mark Lawford[1], Thomas S.E. Maibaum[1], and Alan Wassyng[1]

[1] McMaster Centre for Software Certification, McMaster University, Hamilton, Canada
{singhn10,lawford,wassyng}@mcmaster.ca, tom@maibaum.org
[2] Faculty of Engineering and Natural Sciences, Aalesund University College, Alesund, Norway
hawa@hials.no

Abstract. An insulin infusion pump (IIP) is a critical software-intensive medical device that infuses insulin satisfying patient needs under safety and timing constraints that are appropriate for the treatment of diabetes. This device is used by millions of people around the world. The USA Food and Drug Administration (FDA) has reported several recalls in which IIP failures were responsible for a large number of serious illnesses and deaths. The failures responsible for this harm to people who are dependent on external insulin were caused by the introduction of hardware or software design errors during the system development process. This paper presents an incremental proof-based development of an IIP. We use the Event-B modelling language to formalize the given system requirements. Further, the Rodin proof tools are used to verify the correctness of functional behaviour, internal consistency checking with respect to safety properties, invariants and events.

Keywords: Insulin Infusion Pump (IIP) · Event-B · Refinement · Formal methods · Verification · Validation

1 Introduction

Patient safety is a major concern and an always challenging goal in the design and manufacture of medical devices. It requires knowledge and skill in both the medical and engineering domains, especially in human factors and systems engineering (HFE). The main reasons for recalls related to medical systems are the lack of attention to HFE in the design and implementation of technologies, processes, and usability. The primary use of HFE is to enhance system performance, including patient safety and technology acceptance [1].

An insulin pump is a small, complex, safety-critical software-intensive medical device that allows controllable continuous subcutaneous infusion of insulin to patients for diabetes treatment. This device is used by millions of people around the world. The

Partially supported by: The Ontario Research Fund, and the National Science and Engineering Research Council of Canada.

© Springer International Publishing Switzerland 2015
V.G. Duffy (Ed.): DHM 2015, Part II, LNCS 9185, pp. 387–398, 2015.
DOI: 10.1007/978-3-319-21070-4_39

safety of IIPs has been a major concern in health care for a number of years. The USA Food and Drug Administration (FDA) has reported several recalls in which IIP failures are responsible for a large number of serious illnesses and deaths. According to the FDA, 17000 adverse-events were reported during 2006–2009, where 41 deaths were found to be due to malfunctioning IIPs. The FDA found that these deaths and adverse-events were caused by product design and engineering flaws including firmware problems [2,3].

Formal methods can, and should play a significant role in verifying the system requirements, and in guaranteeing the correctness, reliability and safety of developed system software. Since software plays an important role in the medical domain, regulatory agencies, like the FDA, need effective means to evaluate the software embedded in the devices in order to certify the developed systems, and to assure the safe behaviour of each system [2,4–6]. Regulatory agencies are striving for rigorous techniques and methods to provide safety assurance. Many people believe that formal methods have the potential to develop dependable, safe and secure systems that are also more amenable to certification with required features that can be used to certify dependable medical systems [6–9]. We also note that many formal techniques need to be much better targeted at practical software development and certification than they seem to be at present [10].

This paper contributes to the formalization and verification of an IIP using incremental refinement in Event-B [11]. We previously formalized pacemakers using Event-B [8], and this work now helps us to formulate more general strategies for developing medical devices using formal techniques. It should be noted that in the pacemaker case study, we investigated only a refinement strategy that was used to formalize required behaviours and operating modes incrementally by adding various safety properties. In the IIP case study, we are verifying functional behaviours including various system operations, which are required to maintain insulin delivery, user profile management, and the calculation of required insulin. The complete formal development builds incrementally-refined models of IIP formalizing the required functional behaviour by preserving its required safety properties. The primary use of the models is to assist in the construction, clarification, and validation of the IIP requirements. We use the Rodin [12] tool to develop the formal models. This tool provides an Event-B integrated development environment, automated proof strategies, model checking and code generation.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 gives preliminary information about an IIP including informal system requirements. The modelling framework is presented in Sect. 4. Section 5 explores an incremental proof-based formal development of an IIP. Section 6 concludes the paper along with an indication of our intended future work.

2 Related Work

Masci et al. [13] presented the model-based development of user interface behaviour of an infusion pump in the Prototype Verification System (PVS). The developed model was verified against relevant safety requirements provided by the FDA. Finally, the PVS code generator was used to produce executable code from the verified specifications.

In [14], a prototype of the Generic Patient-Controlled Analgesic (GPCA) infusion pump controller was formalized using the UPPAAL model checker, and then this model was used to generate platform-independent C code. In [15], Structured Object-Oriented Formal Language (SOFL), a formal software engineering method was applied to develop a prototype of an insulin pump, in which the prime motivation was to use SOFL's data-driven, comprehensible, graphical notations for describing the specifications. A generic model for an IIP was developed in the Event-B modelling language to verify the safety requirements related to timing issues [16]. An insulin pump has been used as a case study in [17] for formalizing system behaviours using the Z modelling language.

3 The Insulin Infusion Pump

An insulin pump is a small, complex, software-intensive medical device that allows controllable, continuous subcutaneous infusion of insulin to patients. It delivers physiological amounts of insulin between meals and at meal times. An insulin pump consists of the physical pump mechanism, a disposable reservoir, and a disposable infusion set. The pump system includes a controller, and a battery. The disposable infusion set includes a cannula for subcutaneous insertion, and a tubing system to interface the insulin reservoir to the cannula. At present, open-loop and closed-loop insulin pumps exist. A closed-loop insulin pump is also known as an *artificial pancreas*, which automatically monitors and controls the blood glucose level of a patient. For an open-loop insulin pump, patients need to monitor the blood glucose level manually. An insulin pump can be programmed to release small doses of insulin continuously (basal), or one shot dose (bolus) before a meal, to control the rise in blood glucose.

3.1 Informal IIP Requirements

In this section, we describe briefly the high-level informal functional system requirements of an IIP, that forms the basis of our formal model described in Sect. 5. As far as we know, there are no published system requirements for an IIP, but several research publications provide informal requirements [13, 14, 16]. We used such informal descriptions as a basis for this work to identifying the system requirements by applying *use case* and *hazard analysis*. These system requirements focus on the functional behaviour of an IIP without addressing design requirements, and human computer interaction (HCI) requirements. Our prime objective is to use formal methods to check consistency and required safety properties of the IIP requirements. The informal IIP requirements are described as follows:

REQ1: *The device must suspend all active basal delivery or bolus deliver during pump refilling and in the case of system failure.*
REQ2: *The device must undergo a power-on-self-test (POST) whenever device power is turned on.*
REQ3: *The device shall allow the user to manage system functionalities related to: stopping insulin delivery; validating basal profiles parameters; reminder management; and validating bolus preset parameters.*
REQ4: *The device shall allow the user to define a basal profile that consists of an ordered set of basal rates, ordered over a 24 hour day, as well as a temporary basal,*

that consists of a basal rate for a specified duration of time within a 24 hour day.
REQ5: *The device can contain several basal profiles, but only one basal profile can be active at any single point in time.*
REQ6: *The device must allow the user to override an active basal profile with a temporary basal, without changing the existing basal profile.*
REQ7: *The device shall resume the active basal profile after the temporary basal terminates.*
REQ8: *The device shall enforce a maximum dosage for the normal bolus or extended bolus.*
REQ9: *The user shall be able to stop the active normal or extended bolus.*
REQ10: *The device must maintain an electronic log of every operation associated with an user alert, such as an audio alarm.*
REQ11: *The device shall maintain a history of basal and bolus dosages over the past n days. The n always differs among brands, though most store up to 90 days of data.*
REQ12: *The device shall enable the user to create a food database that can be used to store food or meal descriptions and the carbs associated with them.*
REQ13: *The device shall allow to the user to change parameter setting basal profile, bolus preset, and temporary basal.*
REQ14: *The device shall provide feedback to the user regarding system and delivery status.*

4 The Modelling Framework

In this section, we summarize the Event-B modelling language [11]. The Event-B language has two main components: *context* and *machine*. A *context* describes the static structure of a system, namely *carrier sets* and *constants* together with *axioms* and *theorems* stating their properties. A *machine* defines the dynamic structure of a system, namely *variables, invariants, theorems, variants* and *events*. Terms like *refines, extends*, and *sees* are used to describe the relation between components of Event-B models. Events are used in a *machine* to modify state variables by providing appropriate *guards*.

4.1 Modelling Actions over States

The event-driven approach of Event-B is borrowed from the B language. An Event-B model is characterized by a list of *state variables* possibly modified by a list of *events*. An invariant $I(x)$ expresses required safety properties that must be satisfied by the variable x during the activation of events. An event is a state transition in a dynamic system that contains *guard(s)* and *action(s)*. A *guard*, predicate built on the state variables, is a necessary condition for enabling an event. An *action* is a generalized substitution that describes the ways one or several state variables are modified by the occurrence of an event. There are three ways to define an event e. The first is BEGIN $x : |(P(x, x'))$ END, where the *action* is not guarded and the action is always enabled. The second is WHEN $G(x)$ THEN $x : |(Q(x, x'))$ END, where the *action* is guarded by G, and the *guard* must be satisfied to enable the *action*. The

last is ANY t WHERE $G(t,x)$ THEN $x : |(R(x,x',t))$ END, where the *action* is guarded by G that now depends on the local state variable t for describing non-deterministic events.

The proof obligations (POs) are generated by the Rodin platform [12]. Event-B supports several kinds of POs like invariant preservation, non-deterministic action feasibility, guard strengthening in refinements, simulation, variant, well-definedness etc. Invariant preservation (INV1 and INV2) ensures that each invariant is preserved by each event; non-deterministic action feasibility (FIS) shows the feasibility of the event e with respect to the invariant I; guard strengthening in a refinement ensures that the concrete guards in the refining event are stronger than the abstract ones; simulation ensures that each action in a concrete event simulates the corresponding abstract action; variant ensures that each convergent event decreases the proposed numeric variant; and well-definedness ensures that each axiom, theorem, invariant, guard, action, or variant is well-defined.

$$
\begin{array}{l}
INV1 : Init(x) \Rightarrow I(x) \\
INV2 : I(x) \wedge BA(e)(x,x') \Rightarrow I(x') \\
FIS \ : I(x) \wedge \mathsf{Grd}(e)(x) \Rightarrow \exists y.BA(e)(x,y)
\end{array}
$$

4.2 Model Refinement

A model can be refined to introduce new features or more concrete behaviour of a system. The Event-B modelling language supports a stepwise refinement technique to model a complex system. The refinement enables us to model a system gradually and provides a way to strengthen invariants thereby introducing more detailed behaviour of the system. This refinement approach transforms an abstract model to a more concrete version by modifying the state description. The refinement process extends a list of state variables (possibly suppressing some of them) by refining each abstract event to a corresponding concrete version, or by adding new events. These refinements preserve the relation between an abstract model and its corresponding concrete model, while introducing new events and variables to specify more concrete behaviour of the system. The abstract and concrete state variables are linked by *gluing invariants*. The generated POs ensure that each abstract event is correctly refined by its concrete version. For instance, an abstract model AM with state variable x and invariant $I(x)$ is refined by a concrete model CM with variable y and gluing invariant $J(x,y)$. e and f are events of the abstract model AM and concrete model CM respectively. Event f refines event e. $BA(e)(x,x')$ and $BA(f)(y,y')$ are predicates of events e and f respectively. This refinement relation generates the following PO:

$$
I(x) \wedge J(x,y) \wedge BA(f)(y,y') \Rightarrow \exists x' \cdot (BA(e)(x,x') \wedge J(x',y'))
$$

The new events introduced in a refinement step are viewed as hidden events, that are not visible to the environment of the system being modelled. These introduced events are outside the control of the environment. Newly introduced events refine *skip* and are not observable in the abstract model. Any number of executions of an internal action may occur in between each execution of a visible action. This refinement relation generates the following PO:

$$I(x) \ \wedge \ J(x,y) \ \wedge \ BA(f)(y,y') \ \Rightarrow \ J(x,y')$$

The refined model reduces the degree of non-determinism by strengthening the guards and/or predicates. The refinement of an event e by an event f means that the event f simulates the event e, which guarantees that the set of traces of the refined model contains (up to stuttering) the traces of the resulting model. The Rodin platform provides rich tool support for model development using the Event-B language. The tool support includes project management, model development, proof assistance, model checking, animation and automatic code generation.

5 Formalizing the Insulin Infusion Pump

To cope with the inherent complexity of an IIP, we will use the stepwise refinement approach mentioned in Sect. 4. In our work, we will use this stepwise incremental approach to specify the IIP requirements by introducing new safety properties at each refinement level. The complete development of the IIP using this approach, required eight phases (the initial abstract model followed by seven refinement steps): power status (initial abstract model); basal profile management; temporary basal profile management; bolus preset management; bolus delivery; reminder management; and insulin output calculator. It should be noted that there is no specific order required in which to apply the refinements. Any order can be chosen after developing the initial abstract model. The abstract model can be further refined by introducing new components, enriching the existing behaviours or strengthening the guards.

Since the length of this paper is limited to 12 pages, we only include a brief description of the model development and refinements. We invite readers to use a detailed version of this work [18] to understand the formal development and related refinements of the case study including formally proved Event-B models.

5.1 Abstract Model: Power Status

An abstract model of an IIP specifies only power status and related functionality that controls the power status, i.e. turning *on/off* the system (REQ2). In order to start the formalization process, we need to define static properties of the system. An Event-B context declares three enumerated sets *e_pwrStatus*, *e_basicResp*, and *e_postResult* defined using axioms ($axm1$ - $axm3$) for power status.

$$\begin{array}{l} axm1 : partition(e\_pwrStatus, \{Standby\_pwrStatus\}, \{POST\_pwrStatus\}, \\ \qquad \{Ready\_pwrStatus\}, \{OffReq\_pwrStatus\}) \\ axm2 : partition(e\_basicResp, \{Accept\_basicResp\}, \{Cancel\_basicResp\}) \\ axm3 : partition(e\_postResult, \{Pass\_postResult\}, \{Fail\_postResult\}) \end{array}$$

An abstract model declares a list of variables defined by invariants (*inv1* - *inv5*). A variable *POST_Res* is used to state the result of *power-on-self-test (POST)*, where the result 'pass' ($Pass\_postResult$) means system is safe to turn on, and the result 'fail' ($Fail\_postResult$) means system is unsafe to start. The next variable *post_completed* is used to show successful completion of POST of an IIP.

The variable *c_pwrStatus* shows the current power status of the system. The variable *M_pwrReq* is used to model a request for power *on/off* from the user, and the last variable *M_pwrResp* is used for modelling user responses to system prompts.

$inv1 : POST\_Res \in e\_postResult$
$inv2 : post\_completed \in BOOL$
$inv3 : c\_pwrStatus \in e\_pwrStatus$
$inv4 : M\_pwrReq\_A \in BOOL$
$inv5 : M\_pwrResp \in e\_basicResp$

We introduce 10 events for specifying the desired functional behaviour for controlling the power status of an IIP. These events include guard(s) for enabling the given action(s), and the actions that define the changes to the states of the power status (*c_pwrStatus*) and power-on-self-test (*POST_Res*). Here, we provide only two events related to the power status and power-on-self-test in order to demonstrate the basic formalization process. An event *POST_Completed* is used to assign pass result (*Pass_postResult*) to *POST_Res*, when *post_completed* is *TRUE*. Similarly, another event *PowerStatus1* is used to set $POST\_pwr - Staus$ to *c_pwrStatus*, when power status is *standby*, and there exists a power request from the user. The remaining events are formalized in a similar way.

EVENT POST_Completed
WHEN
grd1 : $post\_completed = TRUE$
THEN
act1 : $POST\_Res := Pass\_postResult$
END

EVENT PowerStatus1
WHEN
grd1 : $c\_pwrStatus = Standby\_pwrStatus$
grd2 : $\exists x \cdot x \in BOOL \wedge x = M\_pwrReq$
THEN
act1 : $c\_pwrStatus := POST\_pwrStatus$
END

We now present summary information about each refinement step in the IIP development, since we do not have space for the detailed formalization and proofs.

5.2 A Chain of Refinements

First Refinement: User Operations. This refinement introduces a set of operations that is performed by the user to operate/program the system for delivering insulin. These user operations create, remove, activate and manage the basal profile, bolus profile, and reminders (REQ3, REQ12, REQ13). These system operations are allowed when an IIP is *on* and we want to deliver an insulin amount in a controlled manner according to the physiological needs of a patient. This refinement formalizes the possible interactions for each system operation to make sure that the given requirements are consistent. For example, if no basal profile exists in the system, then the user will not be allowed to perform any operation other than to create a basal profile, and a notification will appear on the screen to direct the user. This step implements all user interactions with the system, including user initiated commands and system responses. This also includes all safeguards generated by safety constraints resulting from the hazard analysis.

In this refinement, we define an enumerated set and a list of variables to formalize user operations. This refinement step introduces 35 events to specify all the possible user operations related to the given requirements. All these new events refine *skip*. For example, an event *CurrActiUserOper_Idle1* refines *skip* that allows a user to create a new basal profile. The guards of this event state that power status is ready, system is in idle state (means no user operation is currently being performed) and there exists an operation requested by a user to create a new basel profile. We omit the formalization of the rest of the events, which are formalized in a similar way.

EVENT CurrActiUserOper_Idle1
WHEN
grd1 : $c\_pwrStatus = Ready\_pwrStatus$
grd2 : $c\_operation = Idle\_operations$
grd3 : $\exists x \cdot x \in BOOL \wedge x = M\_basCreateReq$
THEN
act1 : $c\_operation := CreateBasProf\_operations$
END

Second Refinement: Basal Profile Management. This refinement introduces basal profile management (REQ4, REQ5) to maintain a record and to store basal profiles defined by the user. The operations of interest are: create a basal profile; remove a basal profile; check the validity of a selected basal profile; activate a basal profile; and deactivate a basal profile. The basal activation process must allow activation only of a valid non-empty basal profile that is stored in the IIP's memory. When a new basal profile is activated then the old basal profile is automatically deactivated, since only one basal profile is allowed to be active at any time. The new profile activation is always confirmed by the user before it can take effect. These operations are introduced in this refinement along the lines seen in the First Refinement, above.

Third Refinement: Temporary Basal Profile Management. This refinement introduces temporary basal profile management (REQ6, REQ7) that allows for activating, deactivating and checking the validity of a selected temporary basal profile. The temporary basal profile management is similar to the basal profile management. As soon as the elapsed period is finished, the paused basal profile resumes after notifying the user. In this refinement, we formalize the possible operations related to the temporary basal management by introducing several new events just as in the previous refinement steps.

Fourth Refinement: Bolus Preset Management. This refinement introduces bolus preset management (REQ3, REQ9) that allows for creating a new bolus preset, removing an existing bolus preset, checking the validity of a created bolus preset, and activating the selected bolus preset. Using new events in a similar way to previous refinement steps, we can formalize the required behaviour. For example, scheduling a bolus has different states like 'no' bolus, 'normal' bolus, and 'extended' bolus. We define transitions between these states to inform the user by notification to confirm a proper bolus status.

Fifth Refinement: Bolus Delivery. This refinement introduces bolus delivery that includes events to start bolus delivery, to calculate the required dose for insulin delivery, and to check the validity of the calculated bolus and manually entered bolus. The bolus delivery formalization step describes how the system will calculate and behave when the user requests a bolus (REQ9, REQ11). When an IIP is *on*, the requested bolus is compared against an average bolus size. The bolus standard deviation must always satisfy the given range. The bolus notification process informs the user whether the bolus is within the regular bolus size, or whether it is larger/smaller than normal. At the time of bolus delivery, an IIP needs confirmation from the user. If the user does not confirm the bolus delivery confirmation, the bolus delivery is unchanged. This refinement formalizes the calculation of bolus delivery and other controlling operations to make sure that an IIP always delivers a correct amount of bolus at the scheduled time.

Sixth Refinement: Reminder Management. This refinement introduces reminder management (REQ10, REQ14) that allows for creating a new reminder, checking the validity of a newly entered reminder, and for removing an existing reminder. The reminder management is a complex task to control several user operations. It includes events to store and maintain reminders defined by the user. Invalid reminders will not be accepted. This refinement step introduces all the events necessary to model all the elements and operations for describing the reminder management, and to verify the requirements of reminder management.

Seventh Refinement: Insulin Output Calculator. This is the last refinement that models the insulin output calculator (REQ8, REQ11). It calculates the insulin required over the course of the day, the appropriate time segment, and the time steps for delivering the insulin. It also keeps track of the insulin delivered within the time segment. The infusion flow rate can be 0, if the system is *off*, and there is no active profile or the maximum amount of insulin has already been delivered.

In this refinement, we introduce 26 events to model the insulin calculator and 14 events to refine other previously abstractly defined events. Again, as an example to demonstrate the refinement process, a new event *InsulinOutputCalculator1* is defined to calculate the amount of insulin to output. It depends on both basal and bolus that are formalized in actions ($act1$-$act2$). The guards of this event are: power status is ready ($grd1$); temporary basal is active ($grd2$); there exists an active temporary basal in which the rate of temporary basal is less than or equal to the maximum allowable rate ($grd3$); bolus delivery is in progress ($grd4$); and there exists an active bolus in which the bolus amount to be delivered is greater than the remaining allowable maximum amount for the next time step ($grd5$). The generated proof obligations also guarantee that an IIP does not deliver excess insulin to a patient.

EVENT InsulinOutputCalculator1
WHEN
grd1 : $c\_pwrStatus = Ready\_pwrStatus$
grd2 : $TemporaryBasalIsActive = TRUE$
grd3 : $\exists x, y, z \cdot x \mapsto y \mapsto z = f\_activeTmpBasal \land$
$y \leq k\_maxOutputRate \land val = y \land val \in \mathbb{N}$
grd4 : $BolusDeliveryinProgress = TRUE$
grd5 : $\exists s, t \cdot s \mapsto t = f\_activeBolus \land$
$t > (k\_maxOutputRate - val)/k\_msPerHr * Delta\_T$
THEN
act1 : $f\_basalOut := (val/k\_msPerHr) * Delta\_T$
act2 : $f\_bolusOut := ((k\_maxOutputRate - val)/k\_msPerHr) * Delta\_T$
END

5.3 Safety Properties

Informally, a safety property stipulates that *"bad things"* do not happen during system execution. A formalized specification that satisfies a safety property involves an invariance argument. This section presents a list of safety properties using invariants ($spr1$ - $spr9$). These safety properties are introduced to make sure that the formalized IIP system is consistent and safe. The first safety property ($spr1$) ensures that when *EnteredBasProfValid* is *TRUE*, the entered basal delivery rate is within the safe range. Similarly, when *EnteredBasProfValid* is *TRUE*, $spr2$ ensures that the total amount

of insulin delivered over a day is within the state limit. $spr3$ and $spr4$ perform the same checks for the selected basal rate and amount when *SelectedBasalProfileIsValid* is *TRUE*. $spr5$ and $spr6$ perform the same checks for the temporary basal profile when *EnteredTemporaryBasalIsValid* is *TRUE*. $spr7$ states that when *SelectedPresetIsValid* is *TRUE*, the bolus rate of a selected bolus profile must be within the range of minimum bolus bound and maximum bolus bound. $spr8$ ensures that when *EnteredBolusIsValid* is *TRUE*, the bolus rate of the entered bolus profile must be within the range of minimum bolus bound and maximum bolus bound. The last safety property ($spr9$) states that the total amount of insulin to output over the next time unit is less than or equal to the maximum daily limit of insulin that can be delivered.

spr1 : $EnteredBasProfValid = TRUE \Rightarrow (\exists x, y \cdot x \mapsto y = M\_basProf \wedge$
$(\forall i \cdot i \in index\_range \wedge i \in dom(y) \Rightarrow y(i) \geq k\_minBasalBound$
$\wedge y(i) \leq k\_maxBasalBound))$

spr2 : $EnteredBasProfValid = TRUE \Rightarrow (\exists x, y, insulin\_amount \cdot x \mapsto y = M\_basProf \wedge$
$insulin\_amount \in y\_insulinValue \wedge (\forall i \cdot i \in index\_range \wedge i \in dom(y) \Rightarrow$
$insulin\_amount = insulin\_amount + y(i) * k\_segDayDur) \wedge$
$insulin\_amount \leq k\_maxDailyInsulin)$

spr3 : $SelectedBasalProfileIsValid = TRUE \Rightarrow (\exists x, y \cdot x \mapsto y = M\_basActSelected \wedge$
$(\forall i \cdot i \in index\_range \wedge i \in dom(y) \Rightarrow y(i) \geq k\_minBasalBound$
$\wedge y(i) \leq k\_maxBasalBound))$

spr4 : $SelectedBasalProfileIsValid = TRUE \Rightarrow$
$(\exists x, y, insulin\_amount \cdot x \mapsto y = M\_basProf \wedge$
$insulin\_amount \in y\_insulinValue \wedge$
$(\forall i \cdot i \in index\_range \wedge i \in dom(y) \Rightarrow$
$insulin\_amount = insulin\_amount + y(i) * k\_segDayDur) \wedge$
$insulin\_amount \leq k\_maxDailyInsulin)$

spr5 : $EnteredTemporaryBasalIsValid = TRUE \Rightarrow$
$(\exists x, y, z \cdot x \mapsto y \mapsto z = M\_tmpBas \wedge$
$y \geq k\_minBasalBound \wedge y \leq k\_maxBasalBound)$

spr6 : $EnteredTemporaryBasalIsValid = TRUE \Rightarrow$
$(\exists x, y, z \cdot x \mapsto y \mapsto z = M\_tmpBas \wedge y * z \leq k\_maxDailyInsulin)$

spr7 : $SelectedPresetIsValid = TRUE \Rightarrow (\exists x, y \cdot x \mapsto y = M\_bolSelected \wedge$
$y \geq k\_minBolusBound \wedge y \leq k\_maxBolusBound)$

spr8 : $EnteredBolusIsValid = TRUE \Rightarrow (\exists x, y \cdot x \mapsto y = M\_bolus \wedge$
$y \geq k\_minBolusBound \wedge y \leq k\_maxBolusBound)$

spr9 : $c\_insulinOut \leq k\_maxDailyInsulin$

5.4 Model Analysis

In this section, we present the proof statistics by presenting detailed information about generated proof obligations. Event-B supports *consistency checking* which shows that a list of events preserves the given invariants, and *refinement checking* which makes sure that a concrete machine is a valid refinement of an abstract machine.

Table 1. Proof Statistics

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	3	3(100%)	0(0%)
First Refinement	22	22(100%)	0(0%)
Second Refinement	98	82(83%)	16(17%)
Third Refinement	26	25(100%)	1(0%)
Fourth Refinement	52	45(87%)	7(13%)
Fifth Refinement	54	54(100%)	0(0%)
Sixth Refinement	66	60(91%)	6(9%)
Seventh Refinement	123	51(42%)	72(58%)
Total	444	342(77%)	102(23%)

This complete formal specification of an IIP contains 263 events, 16 complex data types, 15 enumerated types, and 25 constants for specifying the system requirements. The formal development of an IIP is presented through one abstract model and a series of seven refinement models. In fact, the refinement models are decomposed into several sub refinements. Therefore, we have a total of 43 refinement levels for describing the system behaviour. In this paper, we have omitted the detailed description of the 43

refinements by grouping them into the main components we used to present the formal specification of an IIP.

Table 1 shows the proof statistics of the development in the Rodin tool. To guarantee the correctness of the system behaviour, we established various invariants in the incremental refinements. This development resulted in 444 (100 %) proof obligations, of which 342 (77 %) were proved automatically, and the remaining 102 (23 %) were proved interactively using the Rodin prover (see Table 1). These interactive proof obligations are mainly related to the complex mathematical expressions, which are simplified through interaction, providing additional information for assisting the Rodin prover. Other proofs are quite simple and were achieved by simplifying the predicates.

6 Conclusion and Future Challenges

An insulin pump is a critical software-intensive medical device for delivering controllable continuous subcutaneous infusion of insulin to patients. An insulin delivery process can be programmed by monitoring the patient's condition. Every year recalls are reported by FDA related to insulin pump malfunctions, and many of these recalls are a result of software issues. These software issues include unexpected controller behaviour for delivering insulin, incorrect measurement of insulin dose, overdose of insulin delivery, incorrect inputs for configuration management, etc. To address these software issues, we have proposed a refinement based formal development of an insulin pump to capture the essential requirements and to verify the required safety properties.

To formalize the requirements of an IIP, we used the Event-B modelling language [11] that supports an incremental refinement approach to design a complete system using several layers from an abstract to a concrete specification. Each refined model was proven to guarantee the preservation of required (safety) properties in that model. The initial model captured the basic behaviour of IIP in an abstract way. Subsequent refinements were used to formalize the concrete behaviour for the resulting IIP that covers user operations, basal profile management, temporary basal management, bolus preset management, reminder management, and insulin calculation. In order to guarantee the 'correctness' of the system behaviour, we established various invariants in the incremental refinements. Our complete formal development of this IIP is available in the appendix of report [18], which is more than 1500 pages long. In summary, our main contributions are:

1. Formalizing the requirements of an IIP.
2. Verifying and validating the required behaviour of an IIP.
3. Defining a list of safety properties.
4. Demonstrating how we can help to meet FDA requirements for certifying IIPs using formal methods.
5. Showing how to use Event-B's refinement process to retain intellectual control of the modelling process, formalization, and analysis.

The prime objective of this IIP model is to verify the requirements of an IIP, to check that all the system operational and functional behaviours are consistent that may help to certify the IIPs. In the future, we will use this model to produce source code using

EB2ALL [8]. Moreover, we have plans to develop a closed-loop system using our glucose homeostasis model [3] to verify the correctness of system behaviour, to analyze system operations, and to provide the required safety assurances to meet certification standards.

References

1. Carayon, P., Wood, K.E.: Patient safety. Inf. Knowl. Syst. Manag. **8**(1–4), 23–46 (2009)
2. Chen, Y., Lawford, M., Wang, H., Wassyng, A.: Insulin pump software certification. In: Gibbons, J., MacCaull, W. (eds.) FHIES 2013. LNCS, vol. 8315, pp. 87–106. Springer, Heidelberg (2014)
3. Singh, N.K., Wang, H., Lawford, M., Maibaum, T., Wassyng, A.: Formalizing the glucose homeostasis mechanism. In: Duffy, V.G. (ed.) DHM 2014. LNCS, vol. 8529, pp. 460–471. Springer, Heidelberg (2014)
4. Keatley, K.L.: A review of the FDA draft guidance document for software validation: guidance for industry. Qual. Assur. **7**(1), 49–55 (1999)
5. A reseach and development needs report by NITRD: high-confidence medical devices: cyber-physical systems for 21st century health care. http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf
6. Lee, I., Pappas, G.J., Cleaveland, R., Hatcliff, J., Krogh, B.H., Lee, P., Rubin, H., Sha, L.: High-confidence medical device software and systems. Computer **39**(4), 33–38 (2006)
7. Bowen, J., Stavridou, V.: Safety-critical systems, formal methods and standards. Softw. Eng. J. **8**(4), 189–209 (1993)
8. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer GmbH, London (2013)
9. Méry, D., Singh, N.K.: Real-time animation for formal specification. In: Aiguier, M., Bretaudeau, F., Krob, D. (eds.) Complex Systems Design and Management, pp. 49–60. Springer, Berlin Heidelberg (2010)
10. Wassyng, A.: Though this be madness, yet there is method in it? In: Proceedings of FormaliSE, pp. 1–7. IEEE (2013)
11. Abrial, J.: Modeling in Event-B - System and Software Engineering. Cambridge University Press, Cambridge (2010)
12. Project RODIN: rigorous open development environment for complex systems (2004). http://rodin-b-sharp.sourceforge.net/
13. Masci, P., Ayoub, A., Curzon, P., Lee, I., Sokolsky, O., Thimbleby, H.: Model-based development of the generic PCA infusion pump user interface prototype in PVS. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP. LNCS, vol. 8153, pp. 228–240. Springer, Heidelberg (2013)
14. Kim, B.G., Ayoub, A., Sokolsky, O., Lee, I., Jones, P., Zhang, Y., Jetley, R.: Safety-assured development of the GPCA infusion pump software. In: 2011 Proceedings of the International Conference on Embedded Software (EMSOFT), pp. 155–164, October 2011
15. Wang, J., Liu, S., Qi, Y., Hou, D.: Developing an insulin pump system using the SOFL method. In: 4th Asia-Pacific Software Engineering Conference (APSEC), pp. 334–341 (2007)
16. Xu, H., Maibaum, T.: An Event-B approach to timing issues applied to the generic insulin infusion pump. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 160–176. Springer, Heidelberg (2012)
17. Sommerville, I.: Software Engineering, 7th edn. Pearson Addison Wesley, New Jersey (2004)
18. Singh, N.K., Wang, H., Lawford, M., Maibaum, T.S.E., Wassyng, A.: Report 18: formalizing insulin pump using Event-B. Technical report 18, McSCert, McMaster University, October 2014. https://www.mcscert.ca/index.php/documents/mcscert-reports