Refinement Based Formal Development of Human-Machine Interface

Romain Geniet[1] and Neeraj Kumar Singh[2(✉)]

[1] Université de Rennes 1, Rennes, France
romain.geniet@laposte.net
[2] INPT-ENSEEIHT/IRIT, University of Toulouse, Toulouse, France
nsingh@enseeiht.fr

Abstract. Human factors have been considered as the most common causes of accidents, particularly for interacting with complex critical systems related to avionics, railway, nuclear and medical domains. Mostly, a human-machine interface (HMI) is developed independently and the correctness of possible interactions is heavily dependent on testing, which cannot guarantee the absence of runtime errors. The use of formal methods in HMI development may assure such guarantee. This paper presents a methodology for developing an HMI using a *correct by construction* approach, which allows us to introduce the HMI components, functional behaviour and the required safety properties progressively. The proposed methodology, generic refinement strategy, supports a development of the *model-view-controller* (MVC) architecture. The whole approach is formalized using Event-B and relies on the Rodin tools to check the internal consistency with respect to the given safety properties, invariants and events. Finally, an industrial case study is used to illustrate the effectiveness of our proposed approach for developing an HMI.

Keywords: Human-machine interface (HMI) · Formal methods
Model-view-controller (MVC) · Refinement and proofs · Event-B
Verification · Validation

1 Introduction

The complexity of critical systems constantly increases and it is important to handle such complexity by addressing several aspects, such as system and user interface, of the system development to reduce the rate of system failure. Note that to design a safe interface that enables a user to interact with system unambiguously may help to reduce the rate of system failure. Developing a human-machine interface (HMI) is a difficult and time-consuming task due to complex system characteristics and user requirements, which allows anticipating human behaviour, system components and operational environment. An interactive system is composed of two main components: *functional core* and *interface*. An interface enables a user to communicate with a system.

M. Mazzara et al. (Eds.): STAF 2018 Workshops, LNCS 11176, pp. 240–256, 2018.
https://doi.org/10.1007/978-3-030-04771-9_19

Our work is focused on the development of HMI for checking the correctness of possible HMI behaviours. There are two main HMI concepts: *user-oriented concepts* and *designer-oriented concepts* [11]. Here, in our work, we use the designer-oriented concepts. In this work, our main objective is to investigate the formal development of HMI using a *correct-by-construction* approach in Event-B, particularly for the MVC architecture. As far as, we know that there is no MVC model in the field of HMI, which is formally developed using a *correct-by-construction* approach.

In this paper, we propose a generic development of HMI based on the MVC architecture in order to design and implement complex HMI progressively and then derive a set of patterns of design and proof that can be used in the HMI development. Here, we begin by formalizing the interaction behaviour and possible modes of an HMI. Then, we formalize the notion of controller and manipulation functions, and finally, we finish by adding the elements of the view component. All these modelling steps are applied progressively through satisfying the required safety properties. Moreover, this development also reflects modelling concepts for handling the problem of communication between HMI components. An incremental development of the MVC architecture for HMI preserves the required behaviour in an abstract model as well as in the refined models. The Event-B language is supported by the Rodin [5] platform, which provides a set of tools for developing, proving and managing the formal specifications. We use the ProB model checker tool [26] to analyse and validate the developed models of HMI.

The remainder of this paper is organized as follows. Section 2 presents the required background. In Sect. 3, we propose a methodology for developing a formal model of HMI based on the MVC architecture. Section 4 presents an overview on the selected case study. In Sect. 5, we present a formal development of the case study in Event-B. In Sect. 6, we discuss the results of our work and Sect. 7 presents the related work. Finally, Sect. 8 concludes the paper with future work.

2 Background

2.1 HMI Architecture

Seeheim and ARCH. Initially, this model was appeared in 1983 [25]. *Seeheim architecture* is a model with four components depicted in Fig. 2. These four components are: **(1) Presentation -** this component handles in/out data from a system; **(2) Dialog control -** this component creates a link between the presentation and interface, and a controller translates a set of interactions of a user in machine instructions; **(3) Interface -** this component allows operations on the data of an application; and **(4) Switch -** this component allows a feedback of that actions which are meaningless for an application.

In 1991, the Seeheim architecture was extended to develop the ARCH architecture [25]. For developing the ARCH architecture (see Fig. 1), two new components are introduced in the Seeheim architecture. These two new components make a link between the three existing components from Seeheim architecture, and the switch is removed in this model. The component presentation logic avoids contradictions between the presentation and dialog controller. The functional core is fully independent of the interface. It makes a link to the machine part of HMI (see Fig. 1). *The main advantage of the ARCH model as compared to Seeheim is the improvement of*

the decoupling between components thanks to adapter layers [25]. The adapters enable more flexibility during the evolution of a system, while the other expresses an existence of the functional core for an application.

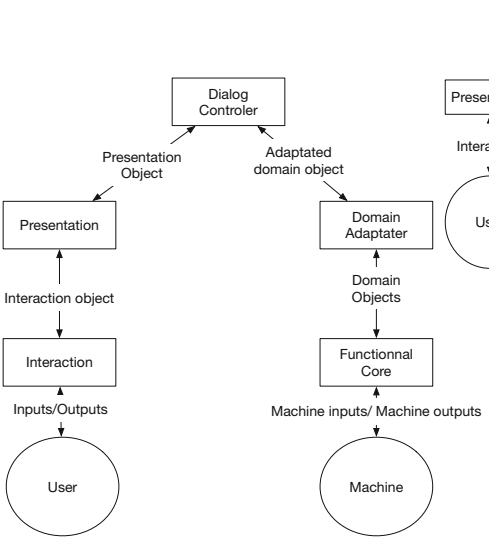

Fig. 1. ARCH architecture [1]

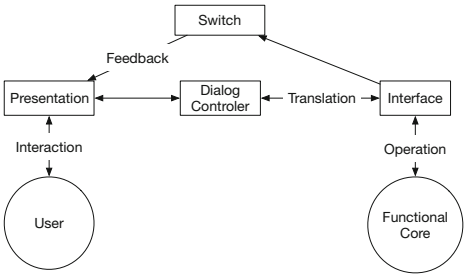

Fig. 2. Seeheim architecture [1]

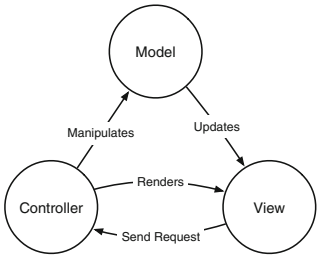

Fig. 3. MVC architecture

MVC. MVC architecture, proposed in 1979, is the most used architecture in the HMI development (see Fig. 3) [34]. There are three main components of this architecture that are described as follows: **(1) Model -** this is the central component of the MVC architecture that allows the management of data, logic and rules of an application; **(2) View -** this component allows the management of an interface display, and the used data is provided by the model; and **(3) Controller -** this component allows the management of data during an input activity from an interface (such as keyboard, mouse, voice...). In our work, we use this architecture to design an HMI.

2.2 HMI Properties

Usability Principles. According to Dix et al. [20], there are three main categories: **Learnability**-*the easy with which new can begin effective interaction and achieve maximal performance*; **Flexibility**-*the multiplicity of ways the user and system exchange information*; and **Robustness**-*the level of support provided to the user in determining successful achievement and assessment of goal-directed behaviour*. Learnability covers the properties of predictability, synthesizability, familiarity, generalizability, and consistency of possible interactions. Flexibility focuses on the dialog initiative, multi-threading, task migratability, substitutivity, customizability. Finally, Robustness addresses the properties of observability, recoverability, responsiveness and task conformance.

CARE. It is a simple framework for reasoning about the multimodal interaction of HMI from both the system and user perspectives [19,29]. A modality is a way of communication that is used by an interface. For example, an interactive map uses the display modality to communicate information with users. The main four properties are: **Complementarity -** a set of modalities must be used in a complementary way to realize a goal; **Assignation -** there is a unique modality to realize a goal; **Redundancy -** a set of modalities is used redundantly if all the modalities have same expressiveness to realize a goal; and **Equivalence -** a set of modalities is equivalent if anyone modality is sufficient to realize a goal.

2.3 Event-B

The Event-B modelling language is developed by Abrial [4,37], in which most of the constructs are borrowed from the B-method [3]. This modelling language is based on the first-order logic and set theory. The goal of this language is to design a complex system using a *correct-by-construction* approach. The correct-by-construction approach allows us to introduce different system behaviours and properties in successive refinements. The development begins with a very high level of abstraction. The refinement enables us to introduce more detailed behaviour and the required safety properties by transforming an abstract model to a concrete version. The final concrete model can be used to produce the source code in any programming language. Note that the refinement always preserves a relation between an abstract model and its corresponding concrete model. The newly generated proof obligations related to refinement ensures that the given abstract model is correctly refined by its concrete version.

There are two main components of Event-B: *context* and *machine*. A context is composed of several elements, such as *set*, *constant*, and *axiom*. The *set* and *constant* elements are defined to state the type definitions and constant definitions to describe the system behaviour. The *axioms* are some logical propositions that cannot be proved but these axioms are used as the base of mathematical reasoning. A context may be an extension of another context. Note that all the elements of the extended context exist in a new context without being declared. A *context* may also contain some theorems in form of logical properties that can be deducted from the existing axioms.

An Event-B model is characterized by a list of state variables that are modified by a list of events to model the changing behaviour of a system with respect to the given conditions. In general, an event can be described in the following form:

$$e \triangleq \textbf{any } var \textbf{ where } grd \textbf{ then } act \textbf{ end}$$

where var is a list of local variables, grd is a set of guards in form of the conjunction of predicates, and act is a set of parallel actions. Any event can be enabled if the given guards are $true$. If more than one event enables simultaneously then any event can be selected for execution non-deterministically, and if none of the events becomes enabled then the system becomes deadlocked. An event can be always enabled if the event is not guarded. In general, a set of actions of an event is a composition of assignments that execute simultaneously, in which a variable assignment can be either deterministic or non-deterministic. The deterministic assignment can be denoted as $x := \textbf{expr}(var)$,

where x is a state variable and **expr**(var) is an expression over the state variable var. The non-deterministic assignment can be denoted as $x :\in S$ or $x : |P(var, x')$, where S is a set of values and $P(var, x')$ is a predicate. In $x :\in S$, x can obtain any value from S and in $x : |P(var, x')$, x can obtain any value that can be satisfied by the predicate $P(var, x')$. Invariants of a machine define the type definition of variables and the required safety properties that must be satisfied during the system execution. In Event-B, there are three type of events: ordinary event, convergent event and anticipated event. The ordinary event has not any constraints. By default, all the events are ordinary events. The convergent event always associates with a variant that models the converging behaviour of a system. An anticipated event is a new event which is not convergent yet but should become convergent in the subsequent refinement.

The foundational semantic of the Event-B language is grounded on *before-after predicates* [4]. The before-after predicate shows a relation between the system states before and after execution of an event. To verify the correctness of an Event-B model, we need to show that the initialization and events preserve the defined invariants. It can be expressed as follows:

$$A(s,c), I(s,c,x), G_e(t,s,c,x), BA_e(t,s,c,x,x') \vdash I(s,c,x')$$

$$A(s,c), BA_{init}(s,c,x') \vdash I(s,c,x')$$

Event-B proof obligations (POs) also allow verifying the event feasibility to show that whenever an event is enabled then there is always a reachable state after the event activation. It can be defined as follows:

$$A(s,c), I(s,c,x), G_e(t,s,c,x) \vdash \exists x'.BA_e(t,s,c,x,x')$$

In the above formulas, $A(s,c)$ is a set of axioms, $I(s,c,x)$ is a set of invariants, $G_e(t,s,c,x)$ is a set of guards and $BA_e(t,s,c,x,x')$ is set of before-after predicates for an event and $BA_{init}(s,c,x')$ is a before-after predicate for the initial event using constants c, carrier sets s and variables x.

To verify the correctness of a refinement step, we need to discharge the generated proof obligations for a refined model. There are several POs, which are detailed in [4]. 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 the concrete model CM, respectively, where event f refines event e. $BA_e(t,s,c,x,x')$ and $BA_f(t,s,c,y,y')$ are before-after predicates of events e and f, respectively. The simulation PO (SIM) shows that the new modified action in the refined event is not contradictory to the abstract action and the concrete event simulates the corresponding abstract event. This SIM PO can be defined as follows:

$$A(s,c), I(s,c,x), J(s,c,x,y), G_f(s,c,x,y), BA_f(t,s,c,y,y') \vdash BA_e(t,s,c,x,x')$$

Similarly, in the refined events, we can strengthen the abstract guards to specify more concrete conditions. The generated POs ensure that if a concrete event is enabled then the corresponding abstract event will also be enabled. This PO is defined as follows:

$$A(s,c), I(s,c,x), J(s,c,x,y), G_f(s,c,x,y) \vdash G_e(s,c,x)$$

Rodin [5] is an open source tool based on the Eclipse framework for developing a formal model in the Event-B language. This is the collection of different tools that includes the project management, model development, refinement and proof assistance, model checking and code generation.

3 Methodology

For developing an HMI based on the MVC architecture using a *correct by construction* approach, we propose a generic development depicted in Fig. 4. On the upper part of the figure, we show the classical scheme of MVC with possible interaction protocol. On the bottom part of the figure, we sketch the possible refinement strategy. In this refinement strategy, each triangle corresponds to the formal development of the MVC components, such as *model*, *controller* and *view*. Note that these triangles are overlapped with each other due to some shared variables and functional behaviours.

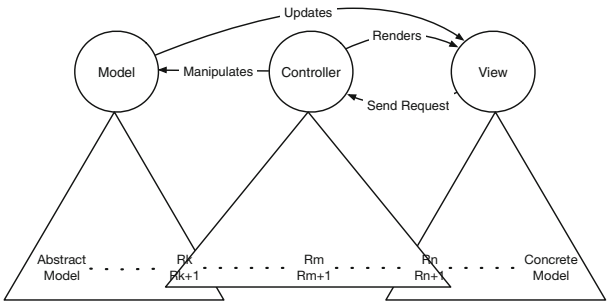

Fig. 4. MVC based refinement strategy

According to the proposed refinement strategy, first, we formalize the *model* components, which describe a very high level of abstraction of HMI in form of system modality. Note that this abstract model can be used in different refinement layers to introduce the complete modality of HMI, and we can also introduce the required safety properties in each refinement level to guarantee the correctness of the modes transitions of HMI. The next step of the development is to introduce the *controller* components and the required controller behaviour. In this phase of the development, we introduce the controller components and their static and dynamic properties. The static properties related to the controller can be defined by extending the context of the model, while the controller components and dynamic properties can be defined by introducing a set of new events and by refining the abstract events. For modelling the controller, we can also use different refinements to reduce the complexity of the controller modelling. All the required safety properties must be introduced in these refinements. The last component of the MVC architecture is *view*, which should be integrated in the previously developed models. In this last phase of the development, we introduce the components and the required properties of the view. The view can also be defined as similar to our previous development in several layers of refinements. By adding the view components, we can prove the correctness of the request functions and responses of the controller. In this step of the development, we implement the behaviours of different elements of HMI. When all the elements are designed and integrated, we introduce the interaction

properties for each component to check the correctness of the interaction behaviour of the developing HMI. Note that the formal development related to the view is complex, and we need to add several guards in different events to meet the desired properties of interaction behaviour for each view component of the HMI.

4 Case Study

In this section, we describe an industrial case study of HMI to understand the modelling and designing concepts, and interaction behaviour of different components. Figure 5 depicts a simple HMI that contains a set of graphical components in form of widgets. In this HMI, we have three modes *stop mode*, *limit mode* and *control mode*. These modes always appear on the top left corner of the HMI that shows an actual modality of the physical system. The stop mode indicates that the physical system connected with HMI is stopped, the limit mode represents that the speed of the physical system is limited, and the control mode indicates that the speed of the physical system is controlled. The HMI shows the selected speed, current speed and current mode. The speed of the vehicle is bounded (selected speed and current speed). The selected speed can be modified using widget components like slider and buttons ('+' and '−').

A set of informal requirements of HMI is defined as: $\mathbf{R_1}$**:** the selected speed is bounded; $\mathbf{R_2}$**:** the current speed is bounded; $\mathbf{R_3}$**:** only one button can be pressed at a time; $\mathbf{R_4}$**:** the slider can be moved only if no button is pressed; $\mathbf{R_5}$**:** the default mode of HMI is stopped; $\mathbf{R_6}$**:** the limit mode and control mode can be active.

Fig. 5. Graphical view of the case study

5 Formalization of HMI

To develop a formal model of the selected case study, we use the Event-B modelling language [4] that supports an incremental refinement to design a complete system in several layers (i.e. model, controller and view), from an abstract to a concrete specification. Firstly, the initial model captures the basic behaviour of the HMI in an abstract way. Then subsequent refinements are used to formalize the concrete behaviour for the resulting HMI that covers the different elements of the HMI. Note that, in this development, we follow the HMI development similar to our proposed methodology.

5.1 Abstract Model: Model

To model the HMI case study, we choose the MVC architecture. An abstract behaviour of the HMI is depicted in Fig. 6. This figure shows an automaton that models the changing states of the controller. When the system is in the stop mode then it can switch either in the limit mode or in the control mode. There are several possible interactions defined in this abstract automata to describe the model of HMI. In the context of the initial model, we define three enumerated sets: *MODES* - a set of different controller modes; *POWERED* - on and off power states; and *STATUS* - driving status and suspended status.

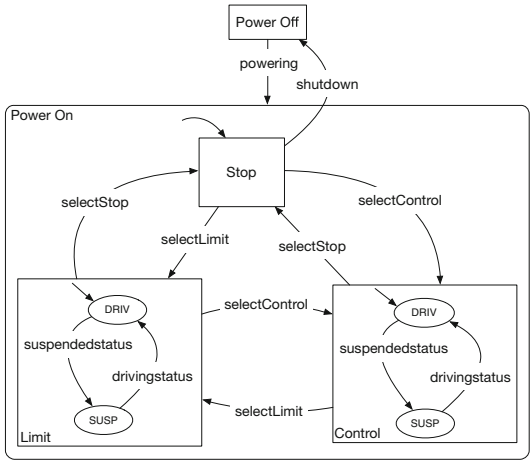

Fig. 6. Automata of an abstract model

$$
\begin{array}{l}
axm1 : partition(MODES, \{STOPPED\}, \{CONTROL\}, \{LIMIT\}) \\
axm2 : partition(POWERED, \{ON\}, \{OFF\}) \\
axm3 : partition(STATUS, \{DRIVING\}, \{SUSPENDED\})
\end{array}
$$

An abstract model is used to show the operating modes by observing the system interaction. The machine model formalizes the dynamic behaviour of the HMI. To define the dynamic properties, we introduce three variables *selectedmode*, *powered* and *status*. The variable *selectedmode* represents the current state of the HMI, the next variable *powered* represents the power status of the HMI and the last variable *status* indicates the current status of the system. Four interesting safety properties are defined using safety invariants ($saf1$-$saf4$). The first safety invariant ($saf1$) expresses that the currently selected mode is not in the stopped mode then the power is on. The next safety property ($saf2$) states that if the current mode is stopped then the status is suspended. The next safety property ($saf3$) states that when the system is in driving state then the selected mode is either in the control mode or in the limit mode. The last safety property ($saf4$) states that when the system is off then the system must be in the stopped mode.

$$
\begin{array}{l}
inv1 : selectedmode \in MODES \\
inv2 : powered \in POWERED \\
inv3 : status \in STATUS \\
saf1 : selectedmode \neq STOPPED \Rightarrow powered = ON \\
saf2 : selectedmode = STOPPED \Rightarrow \\
\qquad status = SUSPENDED \\
saf3 : status = DRIVING \Rightarrow selectedmode = \\
\qquad CONTROL \vee selectedmode = LIMIT \\
saf4 : powered = OFF \Rightarrow selectedmode = STOPPED
\end{array}
$$

EVENT powering
WHEN
grd1 : $powered = OFF$
THEN
act1 : $powered := ON$
END

In this abstract model, we introduce seven events: *powering* - to present the power status of the HMI; *shutdown* - to indicate the shutdown status of the HMI; *selectStop* - to select the stop mode; *selectControl* - to select the control mode; *selectLimit* - to

select the limit mode; *drivingstatus* - to show the driving status; and *suspendedstatus* - to show the suspended status. The event *powering* specifies the power *on* behaviour of the system. The guard of this event shows that the current power status is OFF and the action of this event sets the current power status as ON.

The next event *selectControl* is used to set the control mode and driving status, when system power is ON and the currently selected mode is not in the *CONTROL* mode. Similarly, the last event *drivingstatus* is also used to set the driving status, when the system power is ON, the currently selected mode is not stopped and the system is in suspended status. Rest of the events are modelled in a similar way, and all these events behave similar to the given abstract level automata (see Fig. 6).

EVENT selectControl
WHEN
grd1 : $powered = ON$
grd2 : $selectedmode \neq CONTROL$
THEN
act1 : $selectedmode := CONTROL$
act2 : $status := DRIVING$
END

EVENT drivingstatus
WHEN
grd1 : $powered = ON$
grd2 : $selectedmode \neq STOPPED$
grd2 : $status = SUSPENDED$
THEN
act1 : $status := DRIVING$
END

5.2 First and Second Refinements: Controller

There two different successive refinements for introducing the controller components. In both refinements, we introduce the controller behaviour according to the MVC architecture. In order to design the controller, we introduce the initial speed ($vinit$), maximum speed ($vmax$), and bounded speed ($SPEED$) using axioms ($axm1$ - $axm4$) in the first refinement. Note that the axiom $axm3$ states that the maximum speed must be greater than the initial speed.

In the first refinement, we only introduce a new variable $SelectedSpeed$, which is defined as $SelectedSpeed \in SPEED$. In this refinement, we introduce a new event *ChangeSpeed* for modifying the selected speed non-deterministically. This event will be refined in the next refinement to add more precise controller behaviour of the system.

$axm1 : vinit \in \mathbb{N}$
$axm2 : vmax \in \mathbb{N}$
$axm3 : vmax \geq vinit$
$axm4 : SPEED = 0..vmax$

EVENT ChangeSpeed
WHEN
grd1 : $powered = ON$
THEN
act1 : $SelectedSpeed :\in SPEED$
END

In the second refinement, we introduce a new constant $STEP$ defined as $STEP \in \mathbb{N}$. This constant is used in the refined model to change the selected speed through interacting several HMI components, such as buttons and sliders. In this second refinement, we introduce two new events, *IncreaseSpeed* - to increase a value of the selected speed; and *DecreaseSpeed* - to decrease a value of the selected speed, which are the refinements of the abstract event *ChangeSpeed*. The guards of the *IncreaseSpeed* state that the system power is ON, the choice of step value (x) is either 1 or default STEP, and the sum of the selected speed and the choice of step value (x) must be less than or equal to the maximum speed ($vmax$). The action of this event states that the selected speed increases by the step value (x).

EVENT IncreaseSpeed REFINES ChangeSpeed
ANY x
WHEN
grd1 : $powered = ON$
grd2 : $x = 1 \lor x = STEP$
grd3 : $SelectedSpeed + x \leq vmax$
THEN
act1 : $SelectedSpeed := SelectedSpeed + x$
END

EVENT DecreaseSpeed REFINES ChangeSpeed
ANY x
WHEN
grd1 : $powered = ON$
grd2 : $x = 1 \lor x = STEP$
grd3 : $SelectedSpeed - x \geq vmax$
THEN
act1 : $SelectedSpeed := SelectedSpeed - x$
END

The event *DecreaseSpeed* is also formalised similar to the event *IncreaseSpeed*. The guards of the *DecreaseSpeed* state that the system power is ON, the choice of step value (x) is either 1 or default STEP, and the subtraction of the choice of step value (x) from the selected speed must be greater than or equal to the maximum speed ($vmax$). The action of this event states that the selected speed decreases by the step value (x).

5.3 Third and Fourth Refinements: View

This is the last phase of our development according to our proposed methodology, which allows us introducing the view components of the MVC architecture using several refinements. In the third refinement, we introduce a set of HMI elements, such as buttons and slider, and possible interactions between HMI components, for example, *click* and *dblclick* operations of buttons, and *moving* and *sliding* operations of slider. In order to design the selected case study, we introduce a slider and two buttons ('+' and '−') to modify the selected speed. In this development, we also introduce a set of buttons to represent the *Toggle*, *Lim*, *Ctrl*, *Curr* and *Off* buttons. We introduce three enumerated sets *SLIDERMODE*, *SLIDERDIRECTION* and *PRESSED* in axioms ($axm1$ – $axm3$). A set of axioms ($axm4$ – $axm5$) is defined to represent the possible slider positions according to the changing speed of the system. An additional axiom ($axm6$) is defined to state that the maximum speed is equivalent to the maximum value of the slider position. It means that whenever the speed changes, the slider position also updates accordingly.

$axm1 : partition(SLIDERMODE, \{YES\}, \{NO\})$
$axm2 : partition(SLIDERDIRECTION, \{NONE\}, \{INCR\}, \{DECR\})$
$axm3 : partition(PRESSED, \{NOTPRESS\}, \{YESPRESS\})$
$axm4 : POSITION = 0 \mathinner{..} xmax$
$axm5 : speed \in POSITION \rightarrow SPEED \land speed = id$
$axm6 : xmax \in \mathbb{N} \land vmax = speed(xmax)$

In this refinement, we introduce ten new variables using invariants ($inv1$ – $inv5$). All these variables represent the different states of the HMI components in form of $PRESSED$, $SLIDERMODE$ or $SLIDERDIRECTION$.

$inv1 : pressedPlus \in PRESSED \land pressedMinus \in PRESSED$
$inv2 : pressedCur \in PRESSED \land pressedToggle \in PRESSED$
$inv3 : pressedOff \in PRESSED \land pressedLim \in PRESSED$
$inv4 : pressedCtrl \in PRESSED \land slidermode \in SLIDERMODE$
$inv5 : sliderdirection \in SLIDERDIRECTION \land sliderposition \in POSITION$

Several new safety properties ($saf1$–$saf10$) are introduced in this development. The first safety property ($saf1$) states that when the button ('+') is pressed then the rest

of the buttons are not pressed. Similar to the first safety property, next eight safety properties ($saf2$ – $saf9$) are introduced, which always guarantee that if the selected button is pressed then the other buttons are not pressed. The next safety property ($saf10$) states that when the slider direction is NONE then the sliding mode is active. The last safety property is a gluing invariant to establish a relation between abstract variable *selectedspeed* and concrete speed function *speed*.

$saf1 : pressedPlus = YESPRESS \Rightarrow (pressedLess = NOTPRESS \wedge$
$\quad pressedToggle = NOTPRESS \wedge pressedOff = NOTPRESS \wedge$
$\quad pressedLim = NOTPRESS \wedge pressedCtrl = NOTPRESS)$
$saf2 : ...$
$...$
$...$
$saf9 : ...$
$saf10 : sliderdirection \neq NONE \Rightarrow slidermode = YES$
$glu1 : \forall p \cdot p \in POSITION \wedge p = sliderposition) \Rightarrow selectedspeed = speed(p)$

In this development, we introduce 14 new events to describe the functional behaviour of the different HMI components. A new event *pressPlus* is defined to show the functional behaviour of the button ('+'). The guards of this event state that the power is ON, slider mode is not active, current button $pressedPlus$ is not pressed and the other remaining buttons are also not pressed. If all the given guards are true then the action states that the current button can be pressed. The next event *unpressPlus* is defined to model the button ('+') when this button is no more active to press. The guards of this event state that the power is on, the button is in the press state and the slider position is greater than the maximum speed. The action of this event states that the button will be switched in the $NOTPRESS$ mode. The other events are used in similar fashion to model the rest of the HMI components. Note that we have also introduced extra guards in other events to model the desired behaviour of the HMI.

EVENT pressPlus
WHEN
grd1 : $powered = ON \wedge slidermode = NO \wedge pressedPlus = NOTPRESS$
grd2 : $pressedMinus = NOTPRESS \wedge pressedCur = NOTPRESS \wedge$
$\quad pressedToggle = NOTPRESS \wedge pressedOff = NOTPRESS \wedge$
$\quad pressedLim = NOTPRESS \wedge pressedCtrl = NOTPRESS$
THEN
act1 : $pressedPlus := YESPRESS$
END

The fourth refinement is also the part of view component according to the MVC architecture. In this refinement, we introduce the current speed of the system which is produced by the physical system and its application. The development of the main physical system is beyond the scope of this work because we are mainly interested to design an HMI using a *correct by construction* technique. However, we introduce a new variable $currentspeed$ as $currentspeed \in SPEED$ and a new event to model an interface between the HMI and physical system. The current speed is defined as similar to the selected speed. The new event $updatecurrentspeed$ is defined to capture the current actual speed of the system.

EVENT unpressPlus
WHEN
grd1 : $powered = ON$
grd3 : $pressedPlus = YESPRESS$
grd4 : $sliderposition + STEP > vmax \vee$
$sliderposition + 1 > vmax$
THEN
act1 : $pressedPlus := NOTPRESS$
END

EVENT updatecurrentspeed
ANY v
WHEN
grd1 : $v \in SPEED$
grd2 : $v \neq currentspeed$
grd3 : $powered = ON$
THEN
act1 : $currentspeed := v$
END

A complete formal development of the HMI case study is available on our website[1].

5.4 Model Validation and Analysis

This section summarises the proof statistics of the generated proof obligations in each refinement. The Event-B supports mainly *consistency checking* and *model analysis*. The consistency checking shows that all the events always preserve the defined safety properties, and the refinement checking checks the correctness of the refinement process.

The model analysis is performed using ProB [26] model checker, which can be used to explore traces of Event-B models. The ProB tool supports *automated consistency checking*, *constraint-based checking* and it can also detect the possible deadlocks. Table 1 summarises the generated proof obligations for each refinement steps.

Table 1. Proof statistics

Model	Total number of POs	Automatic proof	Interactive proof
Abstract model	25	25(100%)	0(0%)
First refinement	5	5(100%)	0(0%)
Second refinement	3	3(100%)	0(0%)
Third refinement	233	219(94%)	14(6%)
Fourth refinement	31	25(81%)	6(19%)
Total	297	277(94%)	20(6%)

The stepwise development results in 297(100%) proof obligations, in which 277(94%) are proved automatically, and the remaining 20(6%) are proved interactively using the different Rodin provers, such as SMT solvers and standard B prover. Note that the third refinement has the highest number of proof obligations because, in this development, we introduce all the HMI components with required functional behaviour. To validate the developed HMI model, we use the ProB tool for animating the models. This validation approach refers to gaining confidence that the developed models are consistent with requirements. The ProB animation helps to identify the desired behaviour of the HMI model in different scenarios. In particular, this tool assists us in finding potential problems, and to improve the guard predicates of events. Moreover, we have also used the ProB tool as a model checker to prove the absence of errors (no counterexample exists) and deadlock-free. It should be noted that the ProB uses all the described safety properties during the model checking process to report any violation of safety properties against the formalized system behaviour.

In this development, the main derived properties from the usability principles, such as consistency, observability and task conformance, are considered. A set of invariants in form of safety properties is introduced equivalent to the subset of the HMI usability principles. Note that these properties are also validated using ProB model checker through animation. For example, in the abstract model, we check the behaviour of the

[1] http://singh.perso.enseeiht.fr/Conference/FMIS2018/HMI_Models.zip.

model components; in the second and third refinements, we check the behaviour of the *controller* components; and in the last third and fourth refinements, we check the behaviour of the *view* components.

6 Discussion

Stepwise refinement played an important role in our work for developing the HMI progressively. A stepwise refinement is a suggestive approach from a long time in order to design a complex system. As we have mentioned before that the refinement is a core concept in Event-B development. It is crucial, how to decide on what to introduce in a new refinement level. There may be no universally 'correct' pattern to follow. However, building on experience in HMI development we identified the order of: (1) Introduce the *model* components of MVC (possible modes of HMI); (2) Introduce the *controller* components of MVC; (3) Introduce the *view* components of MVC.

Note that the adopted notion of MVC allows us to build a complex HMI model systematically and this approach also allows us to do reasoning steps systematically considering usability principles. Due to the complex nature of HMI, we do not claim that the proposed modelling approach (see Fig. 4) can be a standard approach for handling any HMI. In fact, our results showed that the proposed modelling approach can be used to model most of the HMI models. To demonstrate the practicality of the identified modelling pattern based on the MVC (see Fig. 4), we have developed the selected HMI case study using a *correct by construction* approach. We described the system requirements using set-theoretical notations abstractly, that can be further refined incrementally to reach a concrete level similar to code. Event-B has a very good tool support that allows us to prove the given properties (mostly) automatically. Other formal modelling tools like VDM, Z, Alloy can be used in place of the Event-B modelling language.

As far as we know, there is no work related to the formal development of HMI based on the MVC architecture using progressive refinement. We used informal descriptions of the MVC architecture as a basis for this work. We also identified a list of safety properties in the refinement process to verify the correctness of overall formalized system behaviour, including newly introduced features. These safety properties guarantee that all possible executions of the system are safe if the generated proof obligations are successfully discharged – and if our list of safety properties is correct and complete. We have considered only the main safety properties related to modes and interaction of the view components. These properties are derived from the usability principles, such as learnability, flexibility and robustness. We can introduce the additional HMI properties in form of safety properties in different refinements to meet the goal of usability principles. Note that the presented case study does not cover the whole set of usability principles. In particular, the current work is focused on consistency, observability and task conformance. In addition, the use of the model checker allows us the validate the developed model with respect to the given safety properties. In summary, we can conclude that the some of the interesting critical properties of the HMI are proved and checked but other remaining properties can be checked during the testing process.

7 Related Work

There are several works related to the formal development of HMI, but most of them use different methods such as Petri net [31], process algebra [22] and model checking [2]. Bowen et al. [15] present a refinement approach for designing UI, and [14] describes models and techniques to incorporate the design artefacts into a formal development process of HMI to specify the system behaviour. [35] describes a refinement process to demonstrate that the given requirements of a device must be satisfied by the specification. Compos et al. [16] propose a framework for checking the HMI system for a given set of generic properties using model checkers. Combefis et al. [18] present a formal approach based on bisimulation to analyse the HMI mechanism. Navarre et al. [30] propose a framework for analysing the interactive systems, particularly for the combined behaviour of user task models and system models to check whether a user task is supported by the system model. [27] describes an approach for generating formal designs of HMI behaviour from task-analysis models and then the results are demonstrated through different case studies. [17] presents the use of formal techniques for the analysis of human-machine interactions. Michael et al. [23] present a formal approach and methodology for the analysis and generation of user interfaces. Palanque et al. [32,33] propose the development of HMI using Interactive Cooperative Objects (ICO) formalism, in which the object-oriented framework and possible functional behaviour are described with high-level Petri-nets. Bolton et al. [12,13] propose a framework to analyse human errors and system failures by integrating the task models and erroneous human behaviour with formal techniques to check the required safety properties.

Ameur et al. [8,9] propose an incremental development of an interactive system using B methods. The proposed approach targets the important problems of HMI related to reachability, observability and reliability. A global development approach for developing a software for human-computer interaction is proposed in [6,7] that can be used from the abstract model to the code generation. Silva et al. [36] propose an approach to generate user interface software, particularly in Java, from a declarative description in the Teallach MB-UIDE. CARE properties are defined using the first order logic in [10]. A new tool-supported approach from specification to the implementation is proposed in [24]. This approach is based on CAV architecture, which is a hybrid model of the MVC and PAC models. The Event-B language is also used for developing the multi-model interactive system using a correct by construction approach in [10]. The ARCH architecture [29] is used during the development of the multi-model interactive system. In addition, several safety properties are introduced to verify the required multi-model interactive behaviour.

In this paper, our approach is different from existing works. The proposed approach allows us to develop a formal model of an HMI based on the MVC architecture using a *correct by construction* approach by analysing the system requirements, modes and interaction mechanism. The use of refinement approach helps to introduce several properties in a progressive way and to verify the correctness of the HMI model under the given safety properties, which can be derived from the usability principles. Moreover, we can use progressive reasoning step in a complex model to cover the different HMI properties. In addition, the progressively developed model can be used for validating

the specified system requirements using the model checker and animation. Note that the final concrete model of the HMI can be used to generate source code in many programming languages using EB2ALL [21,28] in a prototype development or simulating a user interface.

8 Conclusion

This paper presents a generic methodology for developing a formal model of HMI using incremental refinement. In particular, the proposed methodology focused on the MVC architecture of HMI to analyze an interactive behaviour of a system under the given safety properties. We used the Event-B modelling language, together with its associated tools, to develop the proof-based formal model of HMI using a *correct by construction* approach. Our incremental development of HMI based on the MVC architecture reflects the complexity and modelling challenges in the area of HMI.

The proposed methodology is a generic solution of the HMI development that can help to certify the HMI software. Our goal is to integrate formal models in the development of HMI for verifying the desired behaviour under the relevant safety properties and be able to guarantee the correctness of the functional behaviour. The proposed generic methodology is used to develop the HMI case study for designing a safe interface progressively and checking the correctness of interactive behaviour.

Our future work intends on the proof of specification and their logical translation in order to create templates to conceive and prove the development of HMI in Event-B. Note that the current work does not cover the several HMI properties and CARE properties, so we plan to include these properties in the process of HMI development. Another important goal of this work is to validate the possible interaction using an interface. Thus, our new challenges in the future will be to develop a set of patterns like Dwyer's pattern in order to validate the model through animation and tests. Moreover, we plan to develop a set of libraries of HMI components in Event-B using ontology relations, so that it can be used later in the development of HMI.

References

1. https://tel.archives-ouvertes.fr/file/index/docid/48279/filename/2_2modelesinterface_referen.html
2. Abowd, G.D., Wang, H.M., Monk, A.F.: A formal technique forautomated dialogue development. In: Proceedings of the 1st Conference on Designing Interactive Systems: Processes, Practices, Methods, & Techniques, DIS 1995, pp. 219–226. ACM, New York (1995)
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
4. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
5. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in event-b. Int. J. Softw. Tools Technol. Transf. **12**(6), 447–466 (2010)
6. Ameur, Y.A.: Cooperation of formal methods in an engineering based software development process. In: 2000 Proceedings Second International Conference Integrated Formal Methods, IFM 2000, Dagstuhl Castle, Germany, 1–3 November, pp. 136–155 (2000)

7. Ameur, Y.A., Aït-Sadoune, I., Mota, J., Baron, M.: Validation et vérification formelles de systèmes interactifs multi-modaux fondées sur la preuve. In: Proceedings of the 18th International Conference of the Association Francophone d'Interaction Homme-Machine, Montreal, Quebec, Canada, 18–21 April 2006, pp. 123–130 (2006)
8. Ameur, Y.A., Girard, P., Jambon, F.: A uniform approach for specification and design of interactive systems: the B method. In: Design, Specification and Verification of Interactive Systems 1998, Supplementary Proceedings of the Fifth International Eurographics Workshop, 3–5 June 1998, Abingdon, United Kingdom, pp. 51–67 (1998)
9. Ameur, Y.A., Girard, P., Jambon, F.: Using the B formal approach for incremental specification design of interactive systems. In: Engineering for Human-Computer Interaction, IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction, Heraklion, Crete, Greece, 14–18 September, pp. 91–109 (1998)
10. Ameur, Y.A., Sadoune, I.A., Baron, M.: Etude et comparaison de scénarios de développements formels d'interfaces multi-modales fondés sur la preuve et le raffinement. RSTI- Ingénierie des Systèmes d'Informations **13**(2), 127–155 (2008)
11. Baron, M., Lucquiaud, V., Autard, D., Scapin, D.L.: K-made: Unenvironnement pour le noyau du modèle de description del'activité. In: Proceedings of the 18th Conference onL'Interaction Homme-Machine, IHM 2006, pp. 287–288. ACM, New York (2006)
12. Bolton, M.L., Siminiceanu, R.I., Bass, E.J.: A systematic approach to model checking human - automation interaction using task analytic models. IEEE Trans. Syst. Man Cybern. - Part A: Syst. Hum. **41**(5), 961–976 (2011)
13. Bolton, M.L., Bass, E.J.: Building a formal model of a human-interactive system: insights into the integration of formal methods and human factors engineering. In: First NASA Formal Methods Symposium - NFM, California, USA, 6–8 April, pp. 6–15 (2009)
14. Bowen, J., Reeves, S.: Formal models for user interface design artefacts. Innov. Syst. Softw. Eng. **4**(2), 125–141 (2008)
15. Bowen, J., Reeves, S.: Refinement for user interface designs. Electron. Notes Theor. Comput. Sci. **208**, 5–22 (2008)
16. Campos, J.C., Harrison, M.D.: Systematic analysis of control panel interfaces using formal tools. In: Graham, T.C.N., Palanque, P. (eds.) DSV-IS 2008. LNCS, vol. 5136, pp. 72–85. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70569-7_6
17. Combéfis, S., Giannakopoulou, D., Pecheur, C., Feary, M.: Learning system abstractions for human operators. In: Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering, MALETS 2011, pp. 3–10. ACM, New York City (2011)
18. Combéfis, S., Pecheur, C.: A bisimulation-based approach to the analysis of human-computer interaction. In: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2009, pp. 101–110. ACM, New York (2009)
19. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.M.: Four easy pieces for assessing the usability of multimodal interaction: the care properties. In: Nordby, K., Helmersen, P., Gilmore, D.J., Arnesen, S.A. (eds.) Human-Computer Interaction. IFIP Advances in Information and Communication Technology, pp. 115–120. Springer, Boston (1995). https://doi.org/10.1007/978-1-5041-2896-4_19
20. Dix, A., Finlay, J.E., Abowd, G.D., Beale, R.: Human-Computer Interaction, 3rd edn. Prentice-Hall Inc., Upper Saddle River (2003)
21. EB2ALL: An automatic code generation tool from Event-B (2011). http://eb2all.loria.fr/
22. Eijk, P.V., Diaz, M. (eds.): Formal Description Technique Lotos: Results of the Esprit Sedos Project. Elsevier Science Inc., New York (1989)
23. Heymann, M., Degani, A.: Formal analysis and automatic generation of user interfaces: approach, methodology, and an algorithm. Hum. Factors **49**(2), 311–330 (2007)

24. Jambon, F.: From formal specifications to secure implementations. In: Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces III, Valenciennes, France, 15–17 May 2002, pp. 51–62 (2002)
25. Lecrubier, V.: A formal language for designing, specifying and verifying critical embedded human machine interfaces. Theses, INSTITUT SUPERIEUR DE L'AERONAUTIQUE ET DE L'ESPACE (ISAE); UNIVERSITE DE TOULOUSE, June 2016. https://hal.archives-ouvertes.fr/tel-01455466
26. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
27. Li, M., Wei, J., Zheng, X., Bolton, M.L.: A formal machine-learning approach to generating human-machine interfaces from task models. IEEE Trans. Hum.-Mach. Syst. **47**(6), 822–833 (2017)
28. Méry, D., Singh, N.K.: Automatic code generation from event-b models. In: Proceedings of the Second Symposium on Information and Communication Technology, SoICT 2011, pp. 179–188. ACM, New York (2011)
29. Mohand Oussaïd, L.M.O.: Formal modelling and verification of multimodal human computer interfaces : output multimodality. Theses, ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers, December 2014. https://tel.archives-ouvertes.fr/tel-01127547
30. Navarre, D., Palanque, P., Paternò, F., Santoro, C., Bastide, R.: A tool suite for integrating task and system models through scenarios. In: Johnson, C. (ed.) DSV-IS 2001. LNCS, vol. 2220, pp. 88–113. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45522-1_6
31. Palanque, P., Bastide, R., Sengès, V.: Validating interactive system design through the verification of formal task and system models. In: Bass, L.J., Unger, C. (eds.) EHCI 1995. IFIP Advances in Information and Communication Technology, pp. 189–212. Springer, Boston (1996). https://doi.org/10.1007/978-0-387-34907-7_11
32. Palanque, P.A., Bastide, R.: Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In: Paternó, F. (ed.) Interactive Systems: Design, Specification, and Verification Focus on Computer Graphics (Tutorials and Perspectives in Computer Graphics), pp. 383–400. Springer, Berlin (1995). https://doi.org/10.1007/978-3-642-87115-3_23
33. Palanque, P., Bastide, R.: Verification of an interactive software by analysis of its formal specification. In: Nordby, K., Helmersen, P., Gilmore, D.J., Arnesen, S.A. (eds.) Human—Computer Interaction. IFIP Advances in Information and Communication Technology, pp. 191–196. Springer, Boston (1995). https://doi.org/10.1007/978-1-5041-2896-4_32
34. Reenskaug, T.M.H.: The original MVC reports (1979)
35. Ruksenas, R., Masci, P., Harrison, M.D., Curzon, P.: Developing and verifying user interface requirements for infusion pumps: a refinement approach. ECEASST, 69 (2013). ISSN: 1863-2122
36. Pinheiro da Silva, P., Griffiths, T., Paton, N.W.: Generating user interface code in a model based user interface development environment. In: Proceedings of the Working Conference on Advanced Visual Interfaces, AVI 2000, pp. 155–160. ACM, New York (2000)
37. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer, Heidelberg (2013)