

Formal Development of Multi-Purpose Interactive Application (MPIA) for ARINC 661

Neeraj Kumar Singh, Yamine Aït-Ameur, Dominique Méry, David Navarre, Philippe Palanque, Marc Pantel

▶ **To cite this version:**

Neeraj Kumar Singh, Yamine Aït-Ameur, Dominique Méry, David Navarre, Philippe Palanque, et al.. Formal Development of Multi-Purpose Interactive Application (MPIA) for ARINC 661. 7th International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2019), Nov 2019, Shenzhen, China. pp.21-39, 10.1007/978-3-030-46902-3_2 . hal-02942767

HAL Id: hal-02942767
https://hal.science/hal-02942767

Submitted on 18 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/26285

Official URL
https://doi.org/10.1007/978-3-030-46902-3_2

To cite this version: Singh, Neeraj and Ait Ameur, Yamine and Méry, Dominique and Navarre, David and Palanque, Philippe and Pantel, Marc *Formal Development of Multi-Purpose Interactive Application (MPIA) for ARINC 661.* (2020) In: 7th International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2019), 9 November 2019 - 9 November 2019 (Shenzhen, China).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Formal Development of Multi-Purpose Interactive Application (MPIA) for ARINC 661

N. K. Singh[1], Y. Aït-Ameur[1], D. Méry[2], D. Navarre[3], P. Palanque[3], and M. Pantel[1]

[1] INPT-ENSEEIHT / IRIT, University of Toulouse, France
[2] LORIA,Université de Lorraine & Telecom Nancy, Nancy, France
[3] IRIT, Université de Toulouse, Toulouse, France
neeraj.singh@toulouse-inp.fr, yamine.aitameur@toulouse-inp.fr, dominique.mery@loria.fr, navarre@irit.fr, palanque@irit.fr, marc.pantel@toulouse-inp.fr

Abstract. This paper reports our experience for developing Human-Machine Interface (HMI) complying with ARINC 661 specification standard for interactive cockpits applications using formal methods. This development relies on the FLUID modelling language, we have proposed and formally defined in the FORMEDICIS[1] project. FLUID contains essential features required for specifying HMI. To develop the Multi-Purpose Interactive Applications (MPIA) use case, we follow the following steps: an abstract model of MPIA is written using the FLUID language; this MPIA FLUID model is used to produce an Event-B model for checking the functional behaviour, user interactions, safety properties, and interaction related to domain properties; the Event-B model is also used to check temporal properties and possible scenario using the ProB model checker; and finally, the MPIA FLUID model is translated to Interactive Cooperative Objects (ICO) using the PetShop CASE tool to validate the dynamic behaviour, visual properties and task analysis. These steps rely on different tools to check internal consistency along with possible HMI properties. Finally, the formal development of the MPIA case study using FLUID and its embedding into other formal techniques, demonstrates reliability, scalability and feasibility of our approach defined in the FORMEDICIS project.

Keywords: Human-machine interface (HMI), formal method, refinement and proofs, Event-B, PetShop, verification, validation, animation.

1 Introduction

Developing a human-machine interface (HMI) is a difficult and time-consuming task [22] due to complex system characteristics and user requirements, which require anticipating human behaviour, system components and operational environment. Moreover, the design principles of HMI are different from traditional software development processes, including techniques and tools [29]. Considering every aspect of the HMI development process in a single framework, from requirement analysis to implementation, is

[1] Funded by ANR (Agence nationale de la recherche), https://anr.fr/Projet-ANR-16-CE25-0007

a challenging task. Since a long time, formal methods play an important role for analyzing system interaction [5, 10, 11], and their use has been widely adopted in the current development process of HMI. Yet, to our knowledge there is no standard approach that can be used to formally develop and design a safety-critical HMI from spec to code.

The ongoing project, ANR-FORMEDICIS [14] where our work takes place, aims to propose a suite that can be used for developing and designing safety-critical HMIs. In this project, we develop a pivot modelling language, FLUID (Formal Language of User Interface Design), for the formal specification of HMI based on state transitions systems allowing to express requirements, assumptions, expectations, nominal and non nominal properties, and scenarios. Then, formal models in common languages can the be derived from a FLUID model for verification, validation, simulation and animation. The derived formal models use theorem provers and model checkers for analyzing the different required functional properties, nominal and non nominal properties, and scenarios. In our work, we use the Event-B [1] modelling language for producing an abstract formal model and the PetShop CASE tool [27] for producing Interactive Cooperative Objects (ICO) model [23]. The produced models are analyzed with specific developed tools. Rodin [2] is used for Event-B models and PetShop for ICO models. The analyzed models provide feedback to the original FLUID model.

We propose to illustrate the FORMEDICIS approach applying it for the development of a complex case study issued from aircraft cockpit design: MPIA (Multi-Purpose Interactive Applications). First, we develop a FLUID model for MPIA and then we generate an Event-B model and an ICO model from the developed FLUID model. In this development, we begin by specifying different MPIA components, including functional behaviour, states, assumptions, expectations, interactions, properties and scenarios. The embedding of the formal FLUID development of MPIA in Event-B preserves the required behaviour in the developed model. In the generated model, we prove important properties, such as functional behaviour, user interactions, safety properties, and interaction related domain properties. We use the ProB model checker tool [21] to analyze and validate the developed models, and to check temporal properties and possible scenario for HMI. In the ICO model, we provide the dynamic behaviour of MPIA. The developed ICO specification fully describes the potential interactions that users may have with the application. It covers both input and output aspects related to users. In the ICO formalism, there are four components: a cooperative object which describes the behaviour of the object, a presentation part, activation function and rendering function to link between the cooperative object and the presentation part.

This paper is organized as follows. Section 2 presents the required background. Section 3 describes the FLUID language. Section 4 provides the selected MPIA case study. section 5 presents a formal development of the case study in FLUID. Section 6 and Section 7 illustrates the formal developments of the FLUID model in Event-B and PetShop, respectively. In Section 8, we provide an assessment of our work and Section 9 presents related work. Finally, Section 10 concludes the paper with future work.

2 Preliminaries

2.1 The Modelling Framework: Event-B

This section describes the modelling components of the Event-B language [1]. The Event-B language contains two main components, *context* for describing the static properties of a system using *carrier sets* s, *constants* c, *axioms* $A(s, c)$ and *theorems* $T_c(s, c)$, and *machine* for describing behavioural properties of a system using *variables* v, *invariants* $I(s, c, v)$, *theorems* $T_m(s, c, v)$, *variants* $V(s, c, v)$ and *events* evt. A context can be extended by another context, a machine can be refined by another machine and a machine can use *sees* relation to include other contexts.

An Event-B model is characterized by a list of *state variables* possibly modified by a list of *events*. A set of invariants $I(s, c, v)$ shows typing invariants and the required safety properties that must be preserved by the defined system. A set of events presents a state transition in which each event is composed of guard(s) $G(s, c, v, x)$ and action(s) $v :| BA(s, c, v, x, v')$. A *guard* is a predicate, built on state variables, for enabling the event's *action(s)*. An *action* is a generalized substitution that describes the ways one or several state variables are modified by the occurrence of an event.

The Event-B modelling language supports the *correct by construction* approach to design an abstract model and a series of refined models for developing any large and complex system. Refinements, introduced by the REFINES clause, transform an abstract model to a more concrete version by modifying the state description. A refinement allows modelling a system gradually by introducing safety properties at various refinement levels. New variables and new events may be introduced in a new refinement level. These refinements preserve the relation between the refining model and its corresponding refined concrete model, while introducing new events and variables to specify more concrete behavior of a system. The defined abstract and concrete state variables are linked by introducing the *gluing invariants*. The generated proof obligations ensure that each abstract event is correctly refined by its concrete version.

Rodin [2] is an integrated development environment (IDE) for the Event-B modelling language based on Eclipse. It includes project management, stepwise model development, proof assistance, model checking, animation and automatic code generation. Once an Event-B model is modelled and syntactically checked on the Rodin platform then a set of proof obligations (POs) is generated using the Rodin proof engine. Event-B supports different kinds of proof obligations, such as invariant preservation, non-deterministic action feasibility, guard strengthening in refinements, simulation, variant, well-definedness etc. More details related to the modelling language and proof obligations can be found in [1].

2.2 ICO Notation and PetShop CASE Tool

This section recalls the main features of the Interactive Cooperative Objects (ICOs) formal description technique used for modelling software of interactive systems. ICO is dedicated to the specification of interactive systems [23]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation,

inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets to describe their dynamic or behavioural aspects.

ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behavior of objects and communication protocol between objects are described by the Petri net dialect called Cooperative Objects (CO). In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behavior of the object, a presentation part (i.e. the graphical interface), and two functions (the activation function and the rendering function) which make the link between the cooperative object and the presentation part.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user). These aspects are expressed by means of the activation function (for input) and the rendering function (for output). ICOs description do not integrate graphical rendering of information and objects. This is usually delegated to Java code or to other description techniques such as UsiXML [9]. The ICO notation is fully supported by a CASE tool called PetShop [27]. All the models presented in the next sections have been edited and simulated using PetShop. Some formal analysis is also supported by the tool but limited to the underlying Petri net, removing the specificities brought by the high-level Petri net model.

3 FLUID Language

The FLUID language[2] developed in the FORMEDICIS project is organized in three main parts to describe *static, dynamic* and *requirements*. The static part defines type definition, constant, sets and the required features for interactions. The dynamic part defines a state-transition system for describing interactive system. The requirements part expresses the required behaviour, including user tasks and scenarios. A FLUID model is an INTERACTION module which is composed of six sections (see Fig. 1). The first three sections, DECLARATION, ASSUMPTIONS and EXPECTATIONS, describe the static part of a model. The following STATE and EVENT sections describe the dynamic part of a model, and the last REQUIREMENT section describes the requirement part of a model. The DECLARATION section allows to define new typing information that can be used to describe a HMI model.

The typing information may depend on generic and abstract types, such as $sets$, $constants$, *enumerated sets*, and *natural* and *integer numbers*. The STATE section declares a list of variables, which are classified as $Input$, $Output$, $SysInput$ and $SysOutput$. The interactions between system and user can be characterized by the $Input$ and $Output$ variables while the interactions between system components can be characterized by $SysInput$ and $SysOutput$ variables. Note that all these variables can be tagged using domain knowledge concepts borrowed from an external knowledge.

[2] Deliverable D1.1a: Language specification Preliminary version

Model using the $@tag$ (i.e. Enabled, Visible, Checked, Colors) to make explicit the HMI domain properties of HMI components. The EVENT section describes a set of events to present a state transition in which each event is composed of guard(s) and action(s). All these events are also categorized as $acquisition$, $presentation$ and $internal$ events. Acquisition events model acquisition operations of HMI component by modifying the acquisition state variables. Similarly, the presentation events model presentation operation by modifying the presentation state variables. The internal events model internal operations by modifying the internal state variables. These classification of events allow to check reactive properties, such as one stating that every acquisition is immediately followed by a presentation event or an internal event. This section also contains an INITIALISATION event to set initial values.

INTERACTION *Component_Name*
- **DECLARATION**
  - *SETS s*
  - *CONSTANT c*
- **STATE**
  - *Input State Variables*
  - *Output State Variables*
  - *SysInput State Variables*
  - *SysOutput State Variables*
    - v //A variable without $@tag$
    - $v@tag$ //A variables with domain specific $@tag$
- **EVENTS**
  - **INIT**
  - *Acquisition Events*
  - *Presentation Events*
  - *Internal Events*
    - **Event** $evt@tag[x]$
      - **where**
        - $G(s, c, v, x, v@tag, x@tag)$
      - **then**
        - $v :\mid BA(s, c, v, x, v', v@tag, x@tag, v'@tag)$
      - **end**
- **ASSUMPTIONS**
  - $A(s, c)$
- **EXPECTATIONS**
  - $Exp(s, c)$
- **REQUIREMENTS**
  - **PROPERTIES**
    - $Prop(s, c, v, v@tag)$
  - **SCENARIOS**
    - **NOMINAL**
      - $SC(s, c, v, v@tag)$
    - **NON NOMINAL**
      - $NSC(s, c, v, v@tag)$

END *Component_Name*

Fig. 1: FLUID Model structure

The ASSUMPTIONS section introduces the required assumptions related to environment that includes the user and machine agents. These assumptions can be expressed as logical properties to express HMI properties. The EXPECTATIONS section describes *prescriptive* statements that are expected to be fulfilled by parts of the environment of an interactive system. Note that the assumptions and expectations can be expressed in the same way, but both are different. The REQUIREMENTS section is divided into two subsections, known as PROPERTIES and SCENARIOS. The PROPERTIES section describes in logic all the required properties of an interactive system that must be preserved by a defined system. The SCENARIOS section describes both nominal and non-nominal scenarios using algebraic expressions, close to CTT [28], for analyzing possible acceptable and non-acceptable interactions.

4 MPIA Case Study

ARINC 661 is a standard, designed by the Airlines Electronic Engineering Committee (AEEC), for normalizing the definition of a Cockpit Display System (CDS) [6] and it provides guidelines for developing the CDS independently from the aircraft systems. The CDS provides graphical and interactive services to use applications within the flight deck environment. It controls user-system interaction by integrating input devices, such as keyboard and mouse.

We present the Multi-Purpose Interactive Application (MPIA) that complies with ARINC 661 standard to demonstrate our formal modelling and verification approach considering several software engineering concepts related to HMI. Fig. 2 depicts MPIA which is a real User Application (UA) for handling several flight parameters. This application contains a tabbed panel with three tabs, WXR for managing weather radar information, GCAS for Ground Collision Avoidance System parameters and AIRCOND for dealing with air conditioning settings. A crew member is allowed to switch to any mode (see Fig. 2) using tabs. These tabs have three different applications which can be controlled by the pilot and the co-pilot using any input devices.

The MPIA window of any tab is composed of three main parts: *information area*, *workspace area* and *menu bar*. The information area is the top bar of any tab that splits in two parts for displaying the current state of the application on the left part and the error messages, actions in progress or bad manipulation when necessary on the right part. The workspace area shows changes according to the selected interactive control panel. For example, WXR workspace displays all the modifiable parameters of the weather radar sensor, GCAS workspace shows some of the working modes of GCAS, and AIRCOND workspace displays the selected temperature inside an aircraft. The menu bar area contains three tabs for accessing the interactive control panels related to WXR, GCAS and AIRCOND.

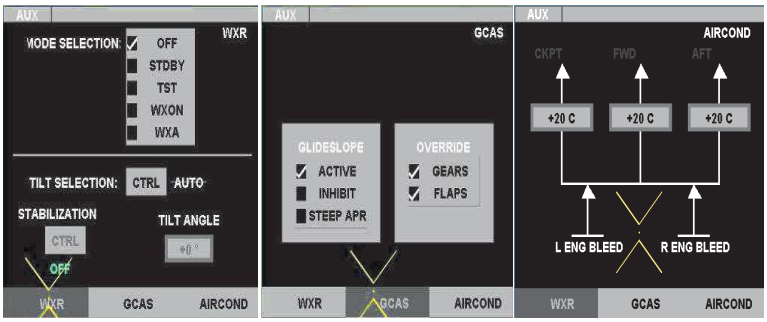


Fig. 2: Snapshots of the MPIA (from left to right: WXR, GCAS and AIRCOND)

5 Formal Development of MPIA in FLUID

We present a formal description of MPIA in FLUID. Due to space limitation, we show only the FLUID model of weather radar information (WXR). The other HMI widgets, such as GCAS and AIRCOND, of MPIA are developed in a similar way.

5.1 Declaration

For modelling the HMI of WXR in FLUID, we define a set of enumerated datatypes and a constant to represent system properties in the DECLARATION clause. Three enumeration sets are: WXR_MODE_SELC_SET for modes, WXR_TILT_STAB-_MSG for messages, and WXR_ACTIONS for actions. A constant WXR_ANGL_RANG is defined a range of tilt angle.

5.2 State

In WXR model, we define several state variables in STATE clause for representing *Input, Output, SysInput* and *SysOutput* states. There are four variables to represent input or acquisition states and six variables to represent output or presentation states. All these variables associated with tag information (*Input, Enabled, Visible, Checked, etc.*) are defined with the given datatypes. Note that the associated $tags$ are defined in a HMI metadata library, including types.

5.3 Events

To model the functional interactive behaviour of WXR, we define a set of events, including an INIT event in the EVENT clause. The *INIT* event only sets initial value for each state variable while the other events are used to model possible HMI behaviour (state changes). In the INIT event, we show initial state of an acquisition variable ($A_ModeSelection$) and a presentation variable ($P_checkMode$), including tag details. Other state variables and their associated $tags$ are initialized in a similar way.

```
DECLARATION
 // WXR Mode enumeration set
TYPE WXR_MODE_SELC_SET = enumeration (M_OFF, STDBY, TST, WXON, WXA)
 // WXR Tilt and Stabilisation message enumeration set
TYPE WXR_TILT_STAB_MSG = enumeration (ON, OFF, AUTO, MANUAL)
 // WXR Tilt angle range
CONSTANT WXR_ANGL_RANG = [ -15 .. 15 ]
 // WRX actions
TYPE WXR_ACTIONS = enumeration (TILT_CTRL, STAB_CTRL)
```

```
STATE Section
 // Acquisition states
A_ModeSelection@{Input, Checked} : WXR_MODE_SELC_SET // Mode state
A_TiltSelection@{Input, Enabled} : WXR_TILT_SELC_SET // Tilt state
A_Stabilization@{Input, Enabled} : WXR_STAB_SELC_SET // Stabilization state
A_TiltAngle@{Input,Enabled} : WXR_ANGL_RANG  // Tile angle state
...
 // Presentation states
 // Radio buttons presentation states
P_checkMode@{Output, Checked} : WXR_MODE_SELC_SET → BOOL
 // CTRL tilt button presentation state
P_ctrlModeTilt_Button@{Output, Enabled} : WXR_ACTIONS
 // CTRL tilt label presentation state
P_ctrlModeTilt_Label@{Output, Visible} : WXR_TILT_STAB_MSG
 // CTRL stablization button presentation state
P_ctrlModeStab_Button@{Output, Enabled} : WXR_ACTIONS
 // CTRL stablization label presentation state
P_ctrlModeStab_Label@{Output, Visible} : WXR_TILT_STAB_MSG
 // Tilt angle value in the presentation state
P_TiltAngle@{Output, Enabled} : WXR_ANGL_RANG
```

The FLUID model contains 6 acquisition events in the acquisition clause, and 7 presentation events in the presentation clause. Here, we only show two acquisition events ($modeSelection$ and $tiltCtrl$) and one presentation event ($checkMode$) to demonstrate the modelling concepts related to HMI. Note that the name of acquisition event is followed by $@Acquisition$, and the name of presentation event is followed by $@Presentation$. The semantics of FLUID language guarantee that an acquisition event is always followed by the corresponding presentation event or internal event to express an interaction behaviour composed of several atomic events related to input, output etc.

The event $modeSelection$ is allowed to select any mode to the input or acquisition state ($A_ModeSelection$) from the workspace area of WXR (see Fig. 2). Note that only input variable and associated tag value are updated through event's actions. Similarly, the event $tiltCtrl$ is used to select a possible action to the input or acquisition state ($A_TiltSelection$). In this event, the actions are also used to update input variable, including tag. The event $checkMode$ presents the state changing behaviour of a widget (radio) defined in the workspace area (see Fig. 2).

The guard of this event state that the selected widget option, acquired by the acquisition state ($A_ModeSelection$) should not be $Checked$. The action of this event shows the selected option as $TRUE$ and the other options as $FALSE$, and the associated tag is updated as $TRUE$. Other events related to acquisition and presentation are modelled in a similar way.

5.4 Requirements

The REQUIREMENTS clause of FLUID model contains a set of required properties, and nominal and non nominal scenarios expressing expected, respectively unexpected, behaviors. In our model, we define 8 safety properties to check the correctness of HMI model. The first safety property ($Prop_1$) states that always a single option is selected from the workspace area (see Fig. 2). The second property ($Prop_2$) states that the acquisition event $modeSelection$ is always followed by the presentation event $checkMode$. Other properties are defined to check the interaction behaviour of HMI components. We define a nominal scenario SC_1 and a non nominal NSC_1 which are started by the INIT event that is followed by the mode selection, tilt selection, stabilization and tilt angle activities using interleaving operator ($||$). Note that each activity is composed of acquisition and presentation events in a sequential order (;). In addition, if there are more than one possible events of acquisition, or presentation then we use optional operator [ ] to compose them. To simulate these scenarios iteratively, we use $*$ operator. Note that the nominal scenario shows possible expected HMI interactions that may occur, while the non nominal scenario shows unexpected HMI interaction that must not occur.

EVENTS Section

// Initialisation Event

INIT =

```
A_ModeSelection := OFF
A_ModeSelection@Checked := TRUE
...
// Only OFF mode is selected at initialisation
P_checkMode := {i ↦ j | i ∈ WXR_MODE_SELC_SET ∧
j = FALSE } ∪ { M_OFF ↦ TRUE } )\{M_OFF ↦ FALSE}
P_checkMode@Checked := TRUE
...
```

// ACQUISITION Events

// Any mode is allowed to select from WXR to acquisition state

Event modeSelection@Acquisition =

```
ANY
  mode
WHERE
  mode : WXR_MODE_SELC_SET
THEN
  A_ModeSelection := mode
  A_ModeSelection@Checked := TRUE
END
```

// The tilt selection model : AUTO or MANUAL (to acquisition state).

// The CTRL push-button allows to swap between the two modes

Event tiltCtrl@Acquisition =

```
ANY
  n_tilt
WHERE
  n_tilt : WXR_ACTION ∧ n_stab = TILT_CTRL ∧
  n_stab@Enabled = TRUE
THEN
  A_TiltSelection := n_tilt
  A_TiltSelection@Enabled := TRUE
END
```

Event stabCtrl@Acquisition = . . .
Event tiltAngle@Acquisition = . . .
Event tiltAngle_Greater_15@Acquisition = . . .
Event tiltAngle_Less_15@Acquisition = . . .

// PRESENTATION Events

// Presentation of radio button: Only selected mode will be checked as TRUE

Event checkMode@Presentation =

```
WHEN
  A_ModeSelection@Checked = TRUE
THEN
  P_checkMode:=( {i ↦ j | i ∈ WXR_MODE_SELC_SET
  ∧ j = FALSE }∪{ A_ModeSelection ↦ TRUE } )\
  {A_ModeSelection ↦ FALSE}
  P_checkMode@checked := TRUE
END
```

Event ctrlModeTilt_Auto@Presentation = . . .
Event ctrlModeTilt_Manual@Presentation = . . .
Event ctrlModeStab_On@Presentation = . . .
Event ctrlModeStab_Off@Presentation = . . .
Event tiltAngle_True@Presentation = . . .
Event tiltAngle_False@Presentation = . . .

REQUIREMENTS Section

PROPERTIES

```
Prop1 :∀ m1,m2· m1∈ WXR_MODE_SELC_SET ∧ m2∈ WXR_MODE_SELC_SET ∧ m1↦ TRUE ∈ prj1(prj1(P_checkMode)) ∧
       m2↦ TRUE ∈ prj1(prj1(P_checkMode)) ⇒ m1=m2
Prop2 :G(e(modeSelection@Acquisition) ⇒ X (e(checkMode@Presentation) )))
Prop3 :(e(tiltAngle@Acquisition) ⇒ (e(tiltAngle_True) or e(tiltAngle_False@Presentation)))
Prop4 :{P_ctrlModeTilt_Label = (AUTO↦Output)↦TRUE ⇒ P_ctrlModeStab_Label = (OFF↦Output)↦TRUE}
Prop5 :{P_ctrlModeTilt_Label = (MANUAL↦Output)↦TRUE ⇒ P_ctrlModeStab_Label = (ON↦Output)↦TRUE}
Prop6 :{P_ctrlModeTilt_Label = (AUTO↦Output)↦TRUE ⇒ P_ctrlModeStab_Button = (STAB_CTRL↦Output)↦FALSE}
Prop7 :{P_ctrlModeTilt_Label = (MANUAL↦Output)↦TRUE ⇒ P_ctrlModeStab_Button = (STAB_CTRL↦Output)↦TRUE}
Prop8 :{P_ctrlModeTilt_Label = (MANUAL↦Output)↦TRUE ⇒ P_TiltAngle = (10↦Output)↦TRUE}
```

SCENARIOS

NOMINAL

```
SC_1 = INIT; ((modeSelection@Acquisition; checkMode@Presentation)
|| (tiltCtrl@Acquisition; (ctrlModeTilt_Auto@Presentation [] ctrlModeTilt_Manual@Presentation))
|| (stabCtrl@Acquisition; (ctrlModeStab_On@Presentation [] ctrlModeStab_Off@Presentation))
|| (tiltAngle@Acquisition [] tiltAngle_Greater_15@Acquisition [] Evt_tiltAngle_Less_15@Acquisition);
(tiltAngle_True@Presentation [] Evt_tiltAngle_False@Presentation))*
```

NON NOMINAL

```
SC_1 = INIT; ((modeSelection@Acquisition; checkMode@Presentation)
|| (tiltCtrl@Acquisition; ctrlModeTilt_Auto@Presentation ; (stabCtrl@Acquisition[]tiltAngle@Acquisition)))*
```

In this model, the SC_1 shows possible interactions of WXR HMI while the NSC_1 shows some of the impossible WXR HMI interactions, for example, if an acquisition of tilt selection is followed by the auto mode presentation then the acquisition of stabilization or tilt angle is not possible.

6 Exploring the MPIA FLUID Model in Event-B

A FLUID model is translated into Event-B as follows: 1) An `INTERACTION` FLUID component is interpreted as a machine and a context in Event-B; 2) All the constants and sets defined in a FLUID model correspond to an Event-B context; 3) FLUID states are translated into a set of variables in an Event-B model, and the variable typing is also defined as typing invariants of Event-B; 4) FLUID initialisation event and the other events are transformed into an Event-B initialisation event and to a set of events; and 5) The properties of FLUID model are translated into Event-B invariants. Note that some properties are translated into temporal properties using LTL or CTL formula in ProB to check system properties and to animate our models. Finally, the produced Event-B model is checked within the Rodin environment and all the defined safety properties proved successfully.

6.1 Model

Context. In the translated model, two different contexts are defined, the first one contains domain specific information related to HMI while the other one is used to define static properties of HMI. In the domain specific context, we define possible tag information for different widgets, for example, we define an enumerated set HMI_TAG to state the tag properties of HMI states in $daxm1$. In addition, we also define three constants, CHECKED, VISIBLE and ENABLED, as boolean to define tag information for HMI widgets ($daxm2$). In the second context, we declare three enumerated sets, WXR_MODE_SELC_SET for modes, WXR_MODE_SELC_SET for a set of messages, and WXR_ACTIONS for a set of actions to specify the MPIA components using axioms ($axm1$-$axm3$). Enumerated sets are defined using the partition statement. We also declare a constant, WXR_ANGL_RANG, to specify a range (-15 .. +15) of the tilt angle in $axm4$.

$$
\begin{array}{l}
daxm1 : partition(HMI_TAG, \{Input\}, \{Output\}, \{SysInput\}, \{SysOutput\}) \\
daxm2 : CHECKED = BOOL \land VISIBLE = BOOL \land ENABLED = BOOL \\
\\
axm1 : partition(WXR_MODE_SELC_SET, \{M_OFF\}, \{STDBY\}, \{TST\}, \{WXON\}, \{WXA\}) \\
axm2 : partition(WXR_TILT_STAB_MSG, \{AUTO\}, \{MANUAL\}, \{ON\}, \{OFF\}) \\
axm3 : partition(WXR_ACTIONS, \{TILT_CTRL\}, \{STAB_CTRL\}) \\
axm4 : WXR_ANGL_RANG = -15 \mathinner{..} 15
\end{array}
$$

Machine. An Event-B machine is also derived from the FLUID model that is translated straightforward. The generated Event-B model shows the HMI behaviour and possible interactions with MPIA widgets. In this model, we introduce 11 state variables ($inv1$ - $inv11$) to model the dynamic behaviour of the system. All these variables are similar to the FLUID model and are declared as $tuple$ using cartesian product ($\times$). Note that each variable contains state information and tag information related to HMI. In the

current model, we introduce a safety property $saf1$ (see property $Prop1$) to state that there is only one mode selected from the MODE SELECTION of WXR. Note that other properties ($Prop2$ - $Prop8$) of the FLUID model are defined later in the ProB model checker.

$inv1 : A_ModeSelection \in WXR_MODE_SELC_SET \times HMI_TAG \times CHECKED$
$inv2 : A_TiltSelection \in WXR_ACTIONS \times HMI_TAG \times ENABLED$
$inv3 : A_Stabilization \in WXR_ACTIONS \times HMI_TAG \times ENABLED$
$inv4 : A_TiltAngle \in WXR_ANGL_RANG \times HMI_TAG \times ENABLED$
$inv5 : P_checkMode \in (WXR_MODE_SELC_SET \rightarrow BOOL) \times HMI_TAG \times CHECKED$
$inv6 : P_ctrlModeTilt_Button \in WXR_ACTIONS \times HMI_TAG \times ENABLED$
$inv7 : P_ctrlModeTilt_Label \in WXR_TILT_STAB_MSG \times HMI_TAG \times VISIBLE$
$inv8 : P_ctrlModeStab_Button \in WXR_ACTIONS \times HMI_TAG \times ENABLED$
$inv9 : P_ctrlModeStab_Button \in WXR_ACTIONS \times HMI_TAG \times ENABLED$
$inv10 : P_ctrlModeStab_Label \in WXR_TILT_STAB_MSG \times HMI_TAG \times VISIBLE$
$inv11 : P_TiltAngle \in WXR_ANGL_RANG \times HMI_TAG \times ENABLED$
$saf1 : \forall m1, m2 \cdot m1 \in WXR_MODE_SELC_SET \wedge m2 \in WXR_MODE_SELC_SET \wedge$
$m1 \mapsto TRUE \in prj1(prj1(P_checkMode)) \wedge m2 \mapsto TRUE \in prj1(prj1(P_checkMode)) \Rightarrow m1 = m2$

Events. In this translated model, we introduce 14 events, including the INITIALISATION event. The INITIALISATION event is used to set the initial value for each declared state. All these state variables are assigned as tuples to show initial states of MPIA.

For example, $P_checkMode$ is set as M_OFF mode and other modes are not selected from the option widget of MPIA (see $act6$).

EVENT INITIALISATION
BEGIN
act1 : $A_ModeSelection := M_OFF \mapsto Input \mapsto TRUE$
act2 : $A_TiltSelection := TILT_CTRL \mapsto Input \mapsto TRUE$
. . .
. . .
act6 : $P_checkMode := (((\{i \mapsto j | i \in WXR_MODE_SELC_SET \wedge j = FALSE\} \cup$
$\{M_OFF \mapsto TRUE\}) \setminus \{M_OFF \mapsto FALSE\}) \mapsto Output \mapsto TRUE$
act7 : $P_ctrlModeTilt_Button := TILT_CTRL \mapsto Output \mapsto TRUE$
. . .
. . .
END

The event *modeSelection@Acquisition* selects the WXR mode in acquisition mode. The guard of this event allows to choose any mode by selecting the option widget. The action of this event states that the acquisition state $A_ModeSelection$ of WXR mode sets the selected mode with tag information, such as this variable is in acquisition state and $checked$. The event *tiltCtrl@Acquisition* is also specified in similar style to model the acquisition behaviour of the tilt angle.

EVENT modeSelection@Acquisition
ANY $mode$
WHERE
grd1 : $mode \in WXR_MODE_SELC_SET$
THEN
act1 : $A_ModeSelection := mode \mapsto Input \mapsto TRUE$
END

EVENT tiltCtrl@Acquisition
ANY n_tilt
WHERE
grd1 : $n_tilt \in WXR_ACTIONS \times HMI_TAG \times ENABLED \wedge$
$prj1(prj1(n_tilt)) = TILT_CTRL \wedge prj2(n_tilt) = TRUE$
THEN
act1 : $A_TiltSelection := n_tilt$
END

The event *checkMode@Presentation* is related to presentation to model the WXR mode. The guard of this event state that acquisition state, *A_ModeSelection*, of WXR mode is checked (TRUE) and the action of this event updates the presentation state variable, *P_checkMode*. The $P_checkMode$ is set as only the selected acquisition mode and other modes are not selected from the option widget of MPIA (see $act1$). Other remaining acquisition and presentation events are modelled in a similar way. A complete formal development of the MPIA case study is available at[3].

[3] http://singh.perso.enseeiht.fr/Conference/FTSCS2019/MPIA_Models.zip

EVENT checkMode@Presentation
ANY n_tilt
WHERE
grd1 : $prj2(A_ModeSelection) = TRUE$
THEN
act1 : $P_checkMode := ((\{i \mapsto j | i \in WXR_MODE_SELC_SET \wedge j = FALSE\} \cup \{prj1(prj1(A_ModeSelection)) \mapsto TRUE\}) \setminus \{prj1(prj1(A_ModeSelection)) \mapsto FALSE\}) \mapsto Output \mapsto TRUE$
END

6.2 Model Validation and Analysis

This section summarises the generated proof obligations using Rodin prover. This development results in 44 proof obligations, in which 41 (93%) are proved automatically, and the remaining 3 (7%) are proved interactively by simplifying them.

The model analysis is performed using ProB [21] 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 possible deadlocks. Note that the generated Event-B model is used directly in ProB. In this work, we use the ProB tool as a model checker to prove the absence of errors (no counterexample exists) and deadlock-free. We also define LTL properties ($Prop1$-$Prop7$) in ProB of the FLUID model to check the correctness of the generated MPIA model. Note 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. To validate the developed MPIA model, we also 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.

$Prop1 : (G(e(AE_modeSelection) => X(e(PE_checkMode))))$
$Prop2 : (e(AE_tiltAngle) => (e(PE_tiltAngle_True) ore(PE_tiltAngle_False)))$
$Prop3 : \{P_ctrlModeTilt_Label = (AUTO|- > Output)|- > TRUE => P_ctrlModeStab_Label = (OFF|- > Output)|- > TRUE\}$
$Prop4 : \{P_ctrlModeTilt_Label = (MANUAL|- > Output)|- > TRUE => P_ctrlModeStab_Label = (ON|- > Output)|- > TRUE\}$
$Prop5 : \{P_ctrlModeTilt_Label = (AUTO|- > Output)|- > TRUE => P_ctrlModeStab_Button = (STAB_CTRL|- > Output)|- > FALSE\}$
$Prop6 : \{P_ctrlModeTilt_Label = (MANUAL|- > Output)|- > TRUE => P_ctrlModeStab_Button = (STAB_CTRL|- > Output)|- > TRUE\}$
$Prop7 : \{P_ctrlModeTilt_Label = (MANUAL|- > Output)|- > TRUE => P_TiltAngle = (10|- > Output)|- > TRUE\}$

7 Exploring the MPIA FLUID Model in PetShop

This section describes the embedding of the FLUID model in PetShop for verifying MPIA interaction behaviour using Petri nets. The ICO specification of MPIA is executable. That allows us to get a quick prototype before its implementation. The MPIA model is also produced in the ICO specification language from the FLUID model. Note that the ICO model only consider input and output aspects extracted from the MPIA FLUID model. These input and output aspects are defined by adding more precise details for execution purpose by analysing and refining the MPIA FLUID model. In the following section, we describe only the development of MPIA in PetShop.

Structuring of the Modelling. ICOs are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and

its "output" aspects (i.e. when and how the application displays information relevant to the user). In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behaviour of the object, a presentation part, and two functions (the activation function and the rendering function) which make the link between the cooperative object and the presentation part. As stated above we present how ICOs are used for describing an interactive application using the WXR application presented in the introduction part of the section 4. We thus successively presents the four ICO parts for that application.

Presentation Part. The Presentation of an object states its external appearance. In the case of a WIMP interface, this Presentation is a structured set of widgets organized in a set of windows. Each widget is for the user to interact with the interactive system (provide input) and/or for the system to present information to the user (present output). The way used to render information (either in the ICOs description and/or code) is hidden behind a set of rendering methods (in order to render state changes and availability of event handlers) and a set of user events, embedded in a software interface, in the same language as the one used for the COs interface description.

```
Public interface WXR_PAGE extends ICOWidget {
// List of user events.
public enum WXR_PAGE_events {asked_off, asked_stdby, asked_wxa,
asked_wxon, asked_tst, asked_auto asked_stabilization,
asked_changeAngle}
// List of activation rendering methods.
void setWXRModeSelectEnabled(WXR_PAGE_events, List<ISubstitution>);
void setWXRTiltSelectionEnabled (WXR_PAGE_events, List<ISubstitution>);
// List of rendering methods.
void showModeSelection (IMarkingEvent anEvent);
void showTiltAngle (IMarkingEvent anEvent);
void showAuto (IMarkingEvent anEvent);
void showStab (IMarkingEvent anEvent);
}
```

Fig. 3: Software interface of the page WXR from the user application MPIA

Cooperative Objects. Using the Cooperative Object (CO) description technique, ICO adds the following features: (1) Links between user events from the presentation part and event handlers from the Cooperative Object description; (2) Links between user events availability and event-handlers availability; and (3) Links between state in the Cooperative Object changes and rendering. As stated above, a CO description is made up of a software interface and its behaviour is expressed using high-level Petri nets. The WXR page does not offer public methods (except the default ones for allowing the event mechanism), and this is why there is no software interface here.

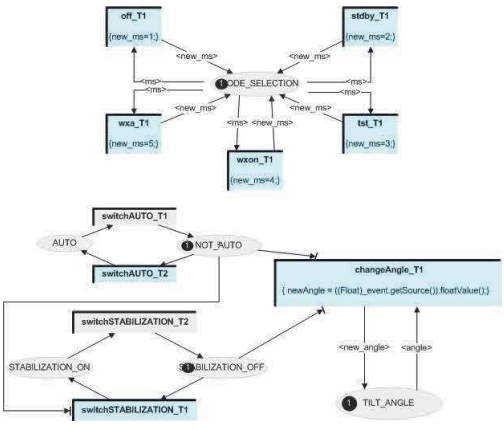


Fig. 4: High-level Petri net model describing the behaviour of the page WXR

Figure 4 shows the entire behaviour of page WXR which is made of two non connected parts: (1) The Petri net in the upper part handles events received from the 5 CheckButtons (see left-hand side of Fig. 2 for the presentation part). Even though they

are CheckButtons the actual behaviour of that application makes it only possible to select one of them at a time. The current selection (an integer value from 1 to 5) is carried by the token stored in MODE_SELECTION place and corresponds to one the possible CheckButtons (OFF, STDBY, TST, WXON, WXA). The token is modified by the transitions (new_ms = 3 for instance) using variables on the incoming and outgoing arcs as formal parameters of the transitions. (2) The Petri net in the lower part handles events from the 2 PicturePushButton and the EditBoxNumeric. Interacting with these buttons will change the state of the application. In the current state, this part of the application is in the manual state and the tokens are placed in the NOT_AUTO and STABILIZATION_OFF. This configuration of tokens is required to make available of the edit box to the user (visible on the model as transition changeAngle_T1 is in a darker colour).

Activation Function. For WIMP interfaces user towards system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the CO event handlers. The relationship between user services and widgets is fully stated by the activation function that associates each event from the presentation part to the event handler to be triggered and to the corresponding rendering method for representing the activation or the deactivation: When a user event is triggered, the Activation function is notified (via an event mechanism) and requires the CO to fire the corresponding event handler providing the value from the user event. When the state of an event handler changes (i.e. becomes available or unavailable), the Activation function is notified (via the observer and event mechanism presented above) and calls the corresponding activation rendering method from the presentation part with values coming from the event handler.

The activation function is fully expressed through a mapping to a CO behaviour element. Figure 5 shows the activation function for page WXR. Each line in this table describes the three objects taking part in the activation process.

User Events	Event handler	Activation Rendering
asked_off	Off	setWXRModeSelectEnabled
asked_stdby	Stdby	setWXRModeSelectEnabled
asked_tst	Tst	setWXRModeSelectEnabled
asked_wxon	Wxon	setWXRModeSelectEnabled
asked_wxa	Wxa	setWXRModeSelectEnabled
asked_auto	switchAUTO	setWXRTiltSelectionEnabled
asked_stabilization	switchSTABILIZATION	setWXRTiltSelectionEnabled
asked_changeAngle	changeAngle	setWXRTiltSelectionEnabled

Fig. 5: Activation Function of the page WXR

The first line, for instance, describes the relationship between the user event ask_-off (produced by clicking on the CheckButton OFF), the event handler off (from the behaviour) and the activation rendering method setWXRModeSelectEnabled from the presentation part. More precisely: (i) When the event handler off becomes enabled, the activation function calls the activation rendering method setWXRModeSelectEnabled providing it with data about the enabling of the event handler. On the physical interaction side, this method call leads to the activation of the corresponding widget (i.e. presenting the checkButton OFF as available). (ii) When the button OFF of the presentation part is pressed, the presentation part raises the event called asked_off. This event is received by the activation function which requires the behaviour part to fire the event handler off (i.e. the transition off_T1 in the Petri net of Figure 4).

Rendering function. For WIMP interfaces system towards user interaction (outputs) present to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external ap-

pearance by reflecting system states changes on the user interface. Indeed, when the state of the Cooperative Object changes (e.g. marking changes for a given place), the Rendering function is notified (via the observer and event mechanism) and calls the corresponding rendering method from the presentation part with tokens or firing values as parameters. In a similar way as for the Activation function, the Rendering function is fully expressed as a CO class.

The rendering function of the WXR application is presented in Fig. 6. In this table one line describes the three objects taking part in the rendering process. The first line for instance describes the relationship between the place MODE_SELECTION, the event linked to this place (and in which we are interested in token_enter) and the rendering method showModeSelection from the presentation part component.

The signification of this line is: When a token enters the place MODE_SELECTION, the rendering function is notified and calls the rendering method showModeSelection providing it with data concerning the new marking of the place that is used as parameters of the rendering method.

ObCS Node name	ObCS event	Rendering method
MODE_SELECTION	token_enter	showModeSelection
TILT_ANGLE	token_enter	showTiltAngle
AUTO	marking_reset	showAuto
AUTO	token_enter	showAuto
AUTO	token_remove	showAuto
STABILIZATION_ON	marking_reset	showStab
STABILIZATION_ON	token_enter	showStab
STABILIZATION_ON	token_remove	showStab

Fig. 6: Rendering Function of the page WXR

8 Assessment

To the best of our knowledge, there is currently no full fledge development framework for covering every aspect of modelling and designing related to interactive systems. Our work project targets such a framework for interactive systems complying with ARINC 661 standard. This is the first integrated formalised framework for formal development of HMI. To support the proposed framework, we have developed a pivot modelling language, FLUID, to specify HMI requirements. Since a long time, stepwise refinement plays an important role for modelling complex systems. We also target a correct by construction design of interactive systems abstractly and then progressively develop a concrete model closed to an implementation. This progressive development allows us to introduce functional behaviour and safety properties related to system and user interactions.

The proposed language is expressive enough to cover possible functional behaviour, system input and output states, presentation, and nominal and non-nominal scenarios. The FLUID language allows us to build a complex HMI systematically, including reasoning for each step systematically considering functions, properties and domain knowledge related to HMI. To demonstrate the practicality of the proposed language, we have developed industrial examples. We have already developed the HMIs for Automatic Cruise Control (ACC), Traffic alert and Collision Avoidance System (TCAS) and MPIA. We can provide a list of safety properties, and nominal and non-nominal scenarios to check the correctness of a formalized system including interaction behaviour. The properties and scenarios derive from the usability principles, such as usability, flexibility and robustness. The presented case study covers only some of the usability principles. such as consistency, observability, tagging and task conformance. In addition, the

ICO specification fully describe the potential interactions that users may have with the application to validate the dynamic behaviour, visual properties and task analysis.

Modelling an interactive system using the FLUID language provides a common understanding for the various stakeholders. In summary, the FLUID model is an abstract pivot core model of HMI for expressing interaction behaviour using state transition systems, assumptions, properties and scenarios. If there will be any error detected then the FLUID model can be modified accordingly. Many techniques, like Event-B, ProB, ICO, task analysis with CTT have been applied on FLUID model. This modelling and analysing steps can be applied iteratively to obtain a correct FLUID model. Similar to this framework, in our MPIA case study, we use on the Event-B modelling language for specifying system and defining safety properties while we use ICO for analysing possible interactions by refining the FLUID model. Note that the use of different tools provides us more confidence on the defined FLUID model. On the other hand we need to check the combination of the approach for an interactive system and the freedom of the integration of different techniques and tools.

9 Related Work

Several approaches are developed in the past years for modelling, designing, verifying and implementing interactive systems. Due to increasing complexity, formal methods is considered as a first-class citizen for modelling and designing the interaction behaviour of HMI for critical systems. There are several approaches, such as Petri net, process algebra and model checking, have been used successfully for checking the intended behaviour of HMI. Palanque et al. [25, 26] 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.

Compos et al. [11] propose a framework for checking the HMI system for a given set of generic properties using model checkers. Navarre et al. [24] propose a framework for analyzing 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. Bolton et al. [10] propose a framework to analyze human errors and system failures by integrating the task models and erroneous human behaviour.

In [5], the authors propose an incremental development of an interactive system using B methods to model the important properties of HMI, such as reachability, observability and reliability. A development lifecycle for generating source code for HMI from an abstract model is presented in [3]. The Event-B language is used for developing the multi-model interactive system supporting with CARE properties using correct by construction approach in [4]. In [19], the authors propose an approach with supported tools based on CAV architecture, hybrid model of MVC and PAC, for developing HMI from specification to implementation. In [16], the authors present a developed methodology, based on MVC architecture, for developing an HMI using a correct by construction approach for introducing functional behaviour, safety properties and HMI components.

A formal interaction mechanism is described using the synchronous data flow language Lustre [17] at ONERA. In [7], the authors present derivation of possible interactions from an informal description of the interactive system. These derived interactions

are used to model a formal model of the interactive system for checking and validating the required HMI behaviour of interactive system, and for generating the test cases [8]. A modelling language, LIDL (LIDL Interaction Description Language), is proposed in [20] to describe a formal description of possible interaction of HMI. In this language, the static nature of HMI is specified using interfaces and the dynamic nature of HMI is specified as interactions. The semantics of this language is based on synchronous data flows similar to Lustre that makes the process easy for formal verification and code generation. In [15], the authors propose a formal development process for designing HMI for safety-critical systems using LIDL and S3 solver.

The project *CHI+MED* [13] proposes modelling in Modal Action Logic (MAL) and proofs in PVS for developing HMI of medical systems. In [18], the authors present a methodology to design a user interface compliant with use-related safety requirements using formal methods. In [12], the authors propose an approach for checking the required properties of executable models of interactive software in *djnn* framework. The *djnn* framework describes interactive components in hierarchical manner, including the low level details such as graphics, behaviours, computations and data manipulations.

All the above approaches are all confronted with different issues like the lack of abstraction or of formal design patterns for handling different aspects of interactive systems. Nevertheless, the main contribution of these researches and studies is to demonstrate only parts of the interactive systems such as interaction, task analysis etc. To our knowledge there is no work related to modelling, refinement, domain knowledge integration and management, scenarios, task analysis together for developing interactive systems. Our work is the first integrated framework for modelling and designing interactive systems by defining different components of interactive systems. Note that our defined language FLUID is able to model interaction behaviour, domain properties, scenarios and tasks properties for interactive systems using a correct by construction. To specify everything in one language provides a common understanding to the various stockholders.

10 Conclusion

This paper presents a formal approach for developing Human Machine Interface complying with ARINC 661. This development approach is centered around the pivot modelling language, FLUID, which is proposed in our FORMEDICIS project for specifying HMI requirements. A FLUID model consists of states, assumptions, expectations, nominal and non nominal properties, and scenarios. A formal model can be derived from a FLUID model for reasoning and analyzing an interactive behaviour of a system under the given safety properties. In our work, we have used the Event-B modelling language for producing a formal model and PetShop CASE tool for producing ICO model. We have used MPIA case study for developing a FLUID model. Further, the FLUID model is used for producing Event-B model and ICO model. The Event-B model is used to check interaction behaviour considering domain properties, including safety properties, and the ICO model is used for validating visual properties and in task analysis. Moreover, we have also used the ProB model checker tool to analyze and to validate the developed MPIA model. The formalization and the associated proofs presented in this

work can be easily extended to other formal methods and model checkers that can be used for modelling interactive systems.

As future work, our objective is to define a refinement relationship for FLUID models to get closer to an implementation. Such refinement allows us to perform formal verification at the code level and we do not need to add any other verification approach. Another future work is to automate the model generation process from a FLUID model, so that a formal model can be produced and verified in any target modelling language.

Acknowledgment. This study was undertaken as part of the FORMEDICIS (FORmal MEthods for the Development and the engineering of Critical Interactive Systems) ANR-16-CE25-0007.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York, NY, USA, 1st edn. (2010)
2. 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 (Nov 2010)
3. Aït-Ameur, Y.: Cooperation of formal methods in an engineering based software development process. In: Integrated Formal Methods, Second International Conference, IFM 2000, Dagstuhl Castle, Germany, November 1-3, 2000, Proceedings. pp. 136–155 (2000)
4. Ait-Ameur, Y., Ait-Sadoune, I., Baron, M.: Etude et comparaison de scénarios de développements formels d'interfaces multi-modales fondés sur la preuve et le raffinement. In: RSTI-Ingénierie des Systèmes d'Informations 13(2). pp. 127–155 (2008)
5. Aït-Ameur, Y., Girard, P., Jambon, F.: Using the B formal approach for incremental specification design of interactiv systems. In: Engineering for Human-Computer Interaction, IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction, September 14-18,, Heraklion, Crete, Greece. pp. 91–109 (1998)
6. ARINC 661-2: Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. Arinc Specification 661-2 (2005)
7. Ausbourg (d'), B., Durrieu, G., Roché, P.: Deriving a formal model of an interactive system from its UIL description in order to verify and to test its behaviour. In: Proceedings of the Eurographics Workshop DSV-IS'96. Namur, Belgium (June 1996)
8. Ausbourg(d'), B.: Using Model Checking for the Automatic Validation of User Interfaces Systems. In: Markopoulos, P., Johnson, P. (eds.) Design, Specification and Verification of Interactive Systems '98. Eurographics, Springer (June 1998)
9. Barboni, E., Martinie, C., Navarre, D., Palanque, P.A., Winckler, M.: Bridging the gap between a behavioural formal description technique and a user interface description language: Enhancing ICO with a graphical user interface markup language. SCP 86, 3–29 (2014)
10. Bolton, M.L., Siminiceanu, R.I., Bass, E.J.: A systematic approach to model checking human - automation interaction using task analytic models. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 41(5), 961–976 (2011)
11. Campos, J.C., Harrison, M.D.: Systematic Analysis of Control Panel Interfaces Using Formal Tools, pp. 72–85. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
12. Chatty, S., Magnaudet, M., Prun, D.: Verification of properties of interactive components from their executable code. In: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 276–285. EICS'15, ACM, NY, USA (2015)

13. Curzon, P., Masci, P., Oladimeji, P., Rukšėnas, R., Thimbleby, H., D'Urso, E.: Human-Computer Interaction and the Formal Certification and Assurance of Medical Devices: The CHI+MED Project. In: 2nd Workshop on Verification and Assurance (Verisure2014), in association with Computer-Aided Verification (CAV), Vienna Summer of Logic (2014)
14. FORMEDICIS Project. https://anr.fr/Projet-ANR-16-CE25-0007
15. Ge, N., Dieumegard, A., Jenn, E., d'Ausbourg, B., Aït-Ameur, Y.: Formal development process of safety-critical embedded human machine interface systems. In: 11th International Symposium on Theoretical Aspects of Software Engineering, TASE'17. pp. 1–8 (2017)
16. Geniet, R., Singh, N.K.: Refinement based formal development of human-machine interface. In: Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers. pp. 240–256 (2018)
17. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous dataflow programming language Lustre. In: Proceedings of IEEE. pp. 1305–1320. No. 9 in 79 (September 1991)
18. Harrison, M.D., Masci, P., Campos, J.C., Curzon, P.: Verification of user interface software: The example of use-related safety requirements and programmable medical devices. IEEE Trans. Human-Machine Systems 47(6), 834–846 (2017)
19. Jambon, F.: From formal specifications to secure implementations. In: Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France. pp. 51–62 (2002)
20. 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 (Jun 2016), https://hal.archives-ouvertes.fr/tel-01455466
21. Leuschel, M., Butler, M.: ProB: A Model Checker for B, pp. 855–874. LNCS, Springer (2003)
22. Myers, B.A.: Why are human-computer interfaces difficult to design and implement? Tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA (1993)
23. Navarre, D., Bastide, R., Palanque, P.: A tool-supported design framework for safety critical interactive systems. Interacting with Computers 15(3), 309–328 (2003)
24. Navarre, D., Palanque, P.A., Paternò, F., Santoro, C., Bastide, R.: A tool suite for integrating task and system models through scenarios. In: 8th International Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS). pp. 88–113 (2001)
25. Palanque, P., Bastide, R., Sengès, V.: Validating interactive system design through the verification of formal task and system models, pp. 189–212. Springer US, Boston, MA (1996)
26. Palanque, P.A., Bastide, R.: Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In: Design, Specification and Verification of Interactive Systems, Proc. of the First International Eurographics Workshop, Italy. pp. 383–400 (1994)
27. Palanque, P.A., Ladry, J., Navarre, D., Barboni, E.: High-fidelity prototyping of interactive systems can be formal too. In: Human-Computer Interaction. New Trends, 13th International Conference, HCI International 2009, San Diego, CA, USA, Part I. pp. 667–676 (2009)
28. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, pp. 362–369. Springer US, Boston, MA (1997)
29. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N.: Designing the User Interface - Strategies for Effective Human-Computer Interaction, 6th Edition. Pearson (2016)