Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Designing for resilience to hardware failures in interactive systems: A model and simulation-based approach

David Navarre*, Philippe Palanque, Eric Barboni, Jean-François Ladry, Célia Martinie

IRIT, University Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France

ARTICLE INFO

Keywords:
Model-based approaches
ARINC 661 specification
Formal description techniques
Interactive software engineering
Interactive cockpits

ABSTRACT

The paper proposes a formal description technique and a supporting tool that provide a means to handle both static and dynamic aspects of input and output device configurations and reconfigurations. More precisely, in addition to the notation, the paper proposes an architecture for the management of failure on input and output devices by means of reconfiguration of in/output device configuration and interaction techniques. Such reconfiguration aims at allowing operators to continue interacting with the interactive system even though part of the hardware side of the user interface is failing. These types of problems arise in domains such as command and control systems where the operator is confronted with several display units. The contribution presented in the paper thus addresses usability issues (improving the ways in which operators can reach their goals while interacting with the system) by increasing the reliability of the system using diverse configuration both for input and output devices.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

Command and control systems have to handle large amounts of increasingly complex information. Current research work in the field of human–computer interaction promotes the development of new interaction and visualization techniques in order to increase the bandwidth between the users and the systems. Such an increase in bandwidth can have a significant impact on efficiency (for instance the number of commands triggered by the users within a given amount of time) and on error-rate [28] (the number of slips or mistakes made by the users).

Post-WIMP user interfaces [31] provide users with several interaction techniques that they can choose from and provide the possibility to exploit different output devices according to different criteria such as, work load, cognitive load, or availability (of the system devices). This includes, for instance, keyboard and mouse as hardware input devices and doubleclick, drag and drop, CTRL+click, etc. as interaction techniques. Exploiting such possibilities calls for methods, techniques, and tools to support various configurations at the specification level (specify in a complete and unambiguous way the configurations, i.e. the set of desired interaction techniques and output configurations), at the validation level (ensure that the configurations meet the requirements in terms of usability, reliability, human-error-tolerance, fault-tolerance, and possibly security), at the implementation level (support the process of going from the specification to the implementation of the configurations in a given system), and for testing (how to test the efficiency of the configurations and of the re-configured system).

A recent trend in human–computer interaction addresses the issue of dynamic reconfiguration of interfaces under the concept of plasticity coined by J. Coutaz [15]. However, research work on plasticity mainly addresses reconfiguration at the output level, i.e. adapting the presentation part of the user interface to the display context (shrinking or expanding presentation objects according to the space available on the display). In addition, reliability issues and specification aspects of plastic interfaces are not considered. Work recently done with web site personalization/configuration [16] and [29] struggle with the same concepts and constraints even though, here again, personalization remains at a cosmetic level and does not deal with how the users interact with the web application. Our work differs significantly as users are pilots following long and intensive training programme (including on-the-fly training) and thus being trained to authorised reconfigurations while web users passively undergo the (most of the time unexpected) reconfigurations.

These issues go beyond current state of the art in the field of interactive systems engineering where usually each interactive system is designed with a predefined set of input and output devices that are to be used according to a static set of interaction techniques and are identified at design time. This set can sometimes gather many different interaction techniques and input/output devices as, for instance, in military cockpits [10].

* Corresponding author.
E-mail addresses: navarre@irit.fr (D. Navarre), palanque@irit.fr (P. Palanque), ladry@irit.fr (J.-F. Ladry).

0951-8320/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2010.06.028

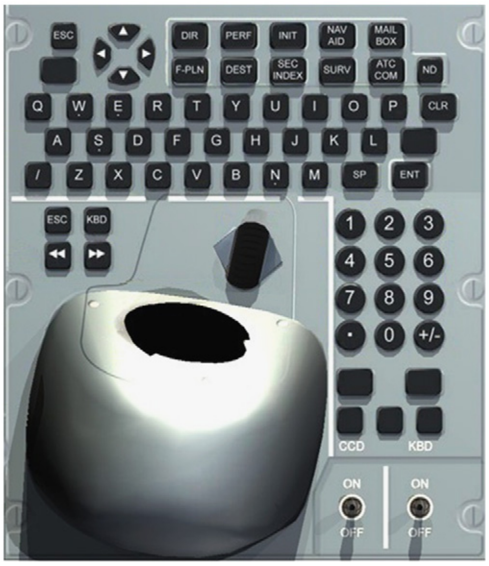


Fig. 1. KCCU (Keyboard Cursor Control Unit) for interactive cockpits taken from Aviation Week & Space Technology (September 27, 2004).

Current safety critical systems, for example, the cockpit of the Airbus A380, presents 8 display units of which 4 of them offer interaction via a mouse and a keyboard by means of an integrated input device called KCCU (Keyboard Cursor Control Unit, see Fig. 1). Applications are allocated to the various display units. If one of the display units fails, then, according to predefined criteria (like the importance of the application according to the flight phase) the applications (displayed on that faulty unit) are migrated to other available display units. This paper proposes a formal description technique and a supporting tool that provide a means to handle both static and dynamic aspects of input and output devices configuration and reconfiguration. The justification of using formal description techniques is three fold:

- The possibility to define in a complete and unambiguous way the behaviour of the input and output devices, the interaction techniques, the authorised configurations, and the reconfiguration mechanism.
- The possibility to reason about that models in order to be able to assess the behaviour of the configurations (e.g. for all the possible configuration a given application is always presented to the operator while other ones might be removed from the set of accessible applications).
- The possibility via the tool PetShop [9] supporting the formal notation to interactively prototype the behaviours and to modify and adjust them according to operator's requirements and global performance.

It is important to note that the purpose of the paper is neither to provide information about the efficiency of a configuration (with respect to other ones) nor it aims at providing ways of assessing the validity of a configuration. The scope is more limited as it only targets at providing a mean to model the interactive system, to model a set of configurations, and to model how configurations evolve according to detected failures.

According to David Wood's definition of resilience in [19] (page 21 second paragraph), resilience applies to systems adapting to unanticipated variability or perturbations. The paper deals explicitly with processes for monitoring and managing resilience as defined later in that paper. Indeed, it targets at supporting the modelling and the architecture of solutions addressing "buffering capacity"[1] even though other aspects such as management, "flexibility"[2], … related ones are clearly beyond the scope of this paper.

The paper is structured as follows: Section 2 briefly presents the formal description technique called ICO. Section 3 introduces the ARINC 661 specification and the generic software architecture for reconfiguration before focusing on an architecture (compliant with the ARINC 661 specification), which is able to handle reconfigurations of both input and output devices. Section 4 is dedicated to the configuration management. It proposes a set of configuration-manager-models applied on a case study (in the field of interactive cockpits). More precisely it illustrates how the reconfigurations (that occur after a hardware failure) are rendered on the display units. Section 5 concludes the paper and presents an agenda for future work.

2. ICOs a formal description technique for interactive systems

The aim of this section is to present the main features of the interactive cooperative objects (ICO) formalism that we have proposed for the formal description of interactive systems. We encourage the interested reader to look at [23,24] for a complete presentation of this formal description technique.

2.1. Informal overview of ICOs

The ICO formalism is a formal description technique 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 [18] to describe their dynamic or behavioural aspects [6].

The ICO notation has evolved to address news challenges raised by the various application domains it has been applied to. This paper briefly presents the current version with the last extensions.

ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by the Petri net dialect called cooperative objects (CO).

Petri nets [27] are a formalism that features a complete equivalence between a graphical and an algebraic representation. As classic Petri nets do not easily allow describing data, the introduction of object Petri nets (OPN) [20] provides a means to handle more complex data structure using the object paradigm. Based on OPN, a cooperative object states how the object reacts to external stimuli according to its inner state. Its behaviour, called the object control structure (ObCS) is described by means of OPN as shown in the case study presented in the next section.

2.2. Interactive cooperative objects formalism (ICO)

In the ICO formalism, an object is an entity featuring four components: a *cooperative object* that describes the behaviour of the object, a *presentation part* (i.e. the graphical interface), and

[1] Buffering capacity is defined as follows: "the size or kinds of disruptions the system can absorb or adapt to without fundamental breakdown …

[2] Flexibility is defined as follows: "the system's ability to restructure itself in response to external changes or pressures".

two functions (the *activation function* and the *rendering function*) which make the link between the cooperative object and the presentation part.

2.2.1. *Cooperative object*

Using the cooperative object formalism, ICO provides the following features:

- Links between user events from the presentation part and event handlers from the CO.
- Links between user event availability and event handler availability.
- Links between state in the CO changes and rendering.

2.2.2. *Presentation part*

The presentation of an object states its external appearance. This presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user→system interaction) and/or a way to display information from this interactive system (system→user interaction). Even if the method used to render (description and/or code) is out of the scope of an ICO specification, it is possible for it to be handled by an ICO, viewing the presentation part as a set of rendering methods (in order to render state changes and availability of event handlers) and a set of user events.

2.2.3. *Activation function*

The user→system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the CO event handlers. The relation between user services and widgets is fully stated by the activation function that associates each event from the presentation part with the event handler to be triggered and the associated rendering method for representing the activation or the deactivation:

- When a user event is triggered, the activation function is notified (via the event mechanism) and asks the CO to fire the corresponding event handler providing it values that only come from the user event.
- When the state of an event handler changes (e.g. becomes available or not), the activation function is notified and calls the corresponding activation rendering method from the presentation part with values for its parameters that only come from the event handler.

2.2.4. *Rendering function*

The system→user interaction (outputs) aims at presenting the state changes that occurs in the system to the user. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes:

- When the state of the cooperative object changes (e.g. marking changes for a place), the rendering function is notified and call the corresponding rendering method from the presentation part with values for its parameters that only come from the event handler.

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 might 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).

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [23]. The specification can also be validated using analysis and proof tools developed within the Petri net community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique. This formal specification technique has already been applied in the field of air traffic control interactive applications [23], space command and control ground systems [26], interactive military [10], or civil cockpits [25]. The example of civil aircraft is used in the next section to illustrate the specification of embedded systems.

To summarize, we provide here the symbols used for the ICO formalism and a screenshot of the tool:

- States are represented by the distribution of tokens into places.
- Actions triggered in an autonomous way by the system are represented by rectangle boxes and called transitions.
- Actions triggered by users are represented by a transition with a thicker boarder (such as the upper left transition in Fig. 2).

Fig. 2 presents a screenshot of the Petshop environment where (1) is the design space, for models edition, (2) is the list of ICO models for the current project, (3) is a mini-map of the ICO model under edition, and (4) is the user interface of the project currently executed by Petshop (this application is presented in detail in Section 4). Modifications made to the models immediately impact the current execution of the project. This makes it easier for the "users" to assess the impact of these modifications on the behaviour of the application.

3. An architecture for reliable and reconfigurable user interfaces

One of the aims of the work presented in this paper is to define an architecture that supports usability aspects of safety-critical systems by taking into account potential malfunctions in the input (output) devices that allow the operators to provide (perceive) information or trigger commands (perceive command results) to the system. Indeed, any malfunction related to such input devices might prevent operators to intervene in the systems functioning, thus jeopardize the mission, and potentially put human life at stake. In systems offering standard input device combination such as mouse+keyboard, it is possible to handle one input device failure by providing redundancy in the use of the device. For instance, a soft keyboard such as the ones defined in [21] can provide an efficient palliative for a keyboard failure[3].

This section deals with architectural concerns for handling reconfigurations in the context of interactive cockpits complaints with the standard ARINC 661 specification. Thus, we first very briefly introduce this standard which defines software interfaces of cockpit display systems. We then successively present a generic architecture for handling reconfigurations and its refinement to address ARINC 661 specification specificities.

[3] This kind of management of input device failure could and should prevent the typical error message on PCs when booting with a missing keyboard "Keyboard Failure strike F1 key to continue".

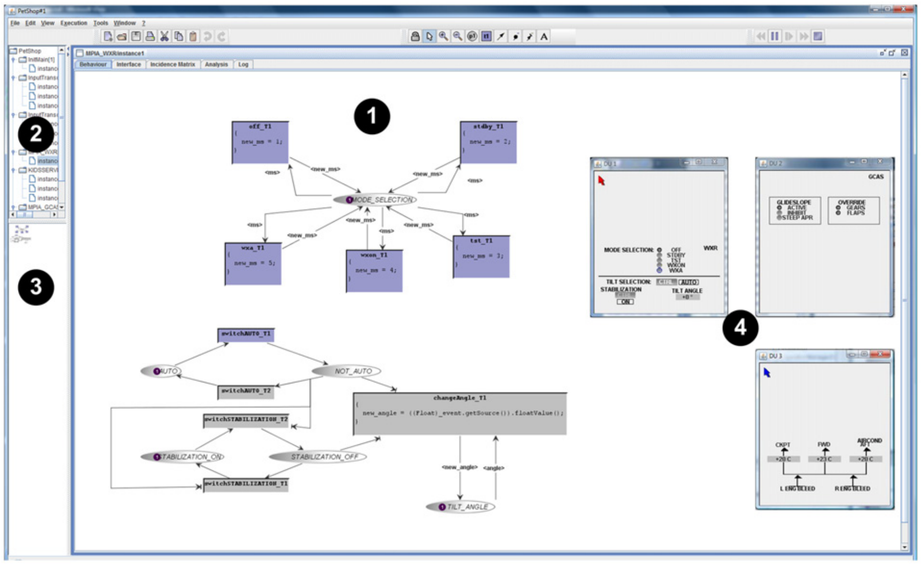


Fig. 2. A screenshot of Petshop environment.

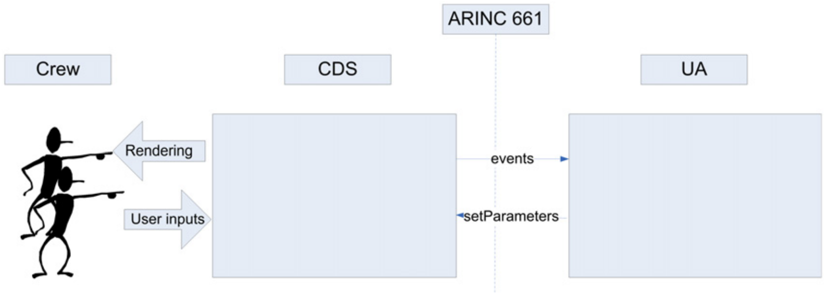


Fig. 3. Abstract architecture and communication protocol between cockpit display system and a user application.

3.1. ARINC 661 specification

The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the cockpit display system (CDS) used in all types of aircraft installations. The standard is called ARINC 661—cockpit display system interfaces to user systems [2,3].

In ARINC 661, a user application is defined as a system that has two-way communication with the CDS (cockpit display system):

- Transmission of data to the CDS, which can be displayed to the flight deck crew.
- Reception of input from interactive items managed by the CDS.

According to the classical decomposition of interactive systems into three parts (presentation, dialogue, and functional core) defined in [14], the CDS part (in Fig. 3) may be seen as the presentation part of the whole system, provided to the crew members, and the set of UAs may be seen as the merge of both the dialogue and the functional core of this system. ARINC 661 then puts on one side input and output devices (provided by avionics equipment manufacturers) and on the other side the user applications (designed by aircraft manufacturers). Indeed, the consistency between these two parts is maintained through the communication protocol defined by ARINC 661.

3.2. A generic architecture for user interaction reconfiguration

In comparison with the ARCH software architecture [5], we propose the following generic extension to support configuration definitions and reconfiguration management in the field of safety critical application. This architecture aims at describing the various components as well as their interrelations. As stated in the introduction, the architecture targets resilient systems [19] offering a *continuity of interaction service* despite partial failure of an input or output device.

In order reach this goal, we propose to decompose the interactive system into two parts: the server side[4] (including the window manager, the interaction techniques, and the (re)configuration manager) and the application side (including all the graphical components such as widgets … up to the functional core).

Fig. 4 presents the architecture of the two components of the reconfigurable interactive system. The left-hand side shows the architecture of the interaction server (software part of the CDS) while the application architecture is represented on the right-hand side. It is noticeable that both components are compliant with the ARCH model and that their interconnection point is the physical interaction part of the application with the functional core part of

[4] This terminology comes from the ARINC 661 specification standard. While such wording could be questionable in the field of interactive systems engineering we prefer to conform to the standard this work is applied to.

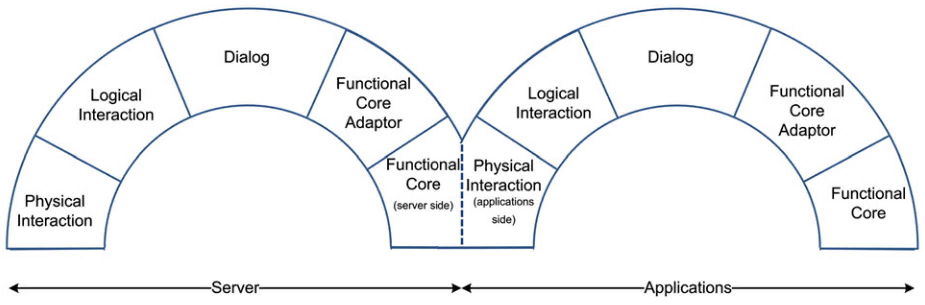


Fig. 4. The server-application dichotomy and their connection point according to the ARCH architecture.

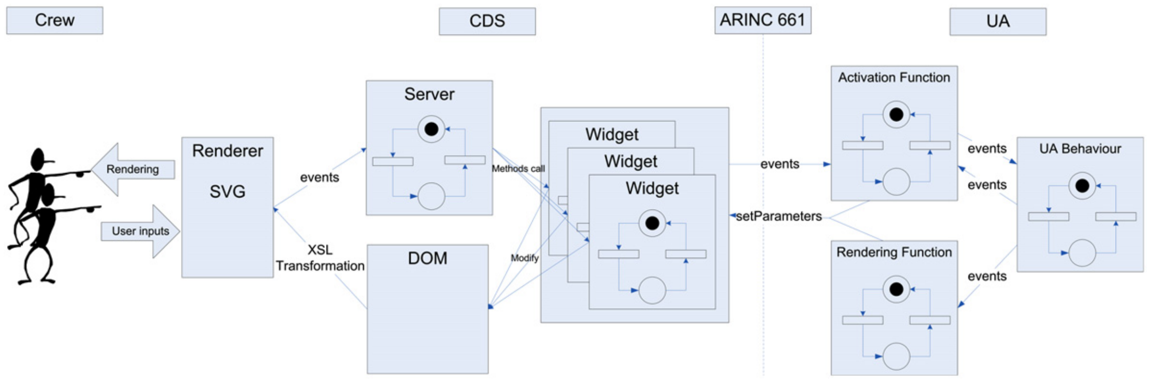


Fig. 5. Detailed architecture compliant with ARINC 661 specification not supporting interaction failures.

the server. More precisely, this "shared" component holds the set of widgets available in the various windows of the application. On the application side, they represent the physical interaction (where the crew member can interact with). On the server side, these widgets correspond to the data managed by the server.

The detail of this architecture (including the structure and behaviour of each component) is presented in the next section on a case study in the field of interactive cockpits.

3.3. *An ARINC 661 compliant architecture to support interaction failure*

The architecture presented in Fig. 5 proposes a structured view on the findings of a project dealing with formal description techniques for interactive applications compliant with the ARINC 661 specification [13,7]. Applications are executed in a cockpit display system (CDS) that aim to provide flight crew with all the necessary information to try to ensure a safe flight.

We are dealing with applications that exclude primary cockpit applications such as PFD (primary flight display) and ND (navigation displays) and only deal with secondary applications such as the ones allocated to the MCDU (multiple control display unit). For previous CDSs (such as the glass cockpit of the A320) these applications were not interactive (they only displayed information to the crew) and inputs were made available through independent physical buttons located next to the display unit. The location in the cockpit in-between the pilot and the first officer make it possible for both of them to use such application.

Within this project, we proposed a unique notation (ICOs) to model the behaviour of all the components of an interactive application compliant with ARINC 661 specification. This includes each interactive component (called widgets) the user application (UA) per se and the entire window manager (responsible for the handling of input and output devices, and the dispatching of events (both those triggered by the UAs and by the pilots) to the recipients (the widgets or the UAs).

The two main advantages of the architecture presented in Fig. 5 are as follows:

- Every component that has an inner behaviour (server, widgets, UA, and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modelled using the ICO formal description technique thus making it possible to analyse and verify the correct functioning of the entire system.
- The rendering part is delegated to a dedicated language and tool (such as SVG (scalable vector graphics), and thus making the external look of the user interface independent from the rest of the application, providing a framework for easy adaptation of the graphical aspect of cockpit applications.

However, this architecture does not support reconfiguration of input or output devices in the cockpit, neither in case of redesign nor in case of failure while in operation. However, requirements specification for a display unit (DU) like the one of the Airbus A380 explicitly requires the possibility for the co-pilot to read information on the DU of the pilot (in case of failure on his/her side for instance).

The new architecture we propose has been extended to explicitly manage the reconfiguration of applications on the display units. It presents a refinement of the architecture proposed in Fig. 4. In that architecture (presented in Fig. 6), all the elements of which the behaviour is formally defined using the ICO formalism appear in a box featuring a small Petri net inside. Indeed, the input and output devices are formally described using the ICO notation in order to be handled by a configuration manager, which is also responsible for reconfiguring devices and

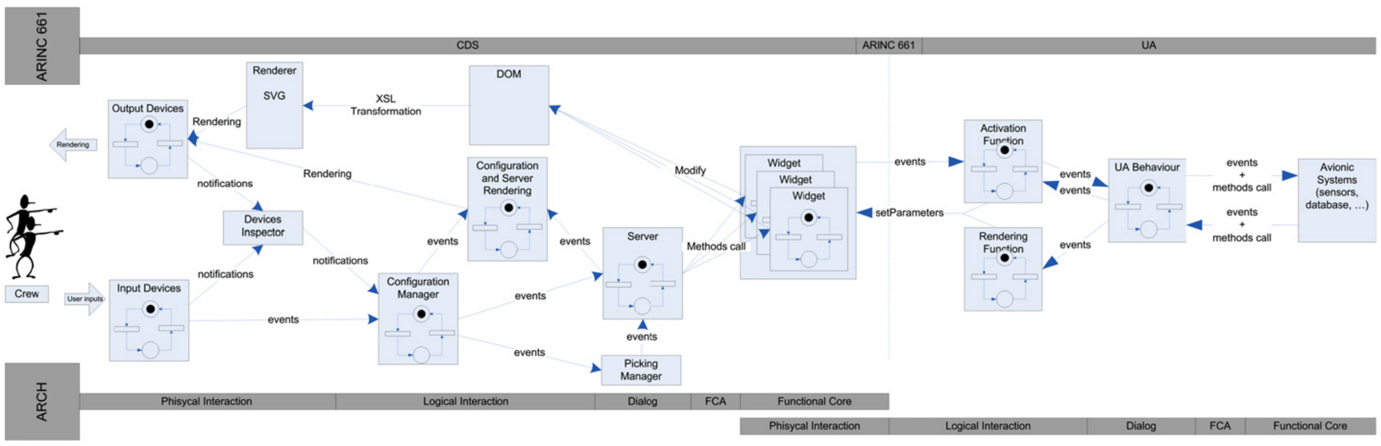


Fig. 6. Global architecture compliant with ARINC 661 specification and supporting interaction failures.

interaction technique according to failures. These failures are detected by a software module (called *Device Inspector*) testing on a regular basis the functioning of the input and output devices.

In Fig. 6 the dashed-line section highlights the improvements made with respect to the previous architecture:

- The left-hand part of the frame highlights the addition of ICO models dedicated to both input and output devices.
- The right-hand part presents the introduction of a new component named *configuration manager* responsible for managing the configuration of input and output devices.
- The *configuration and server rendering* component in charge of representing, on the user interface, the current configuration. In the case study, the current configuration is represented to the crew by different mouse cursors. This is why that component is connected both to the *server* (to access information about the position of the cursor) and to the *configuration manager* to access information about the current configuration.

The upper dark line on top of Fig. 6 positions the architecture according to the ARINC 661 decomposition while the lower dark line positions the various components according to the generic architecture presented in Fig. 4.

Even though modelling of input devices and interaction techniques has already been presented in the context of multimodal interfaces for military cockpits [10], it was not integrated with the previous architecture developed for interactive applications compliant with ARINC 661 specification. The rest of the paper thus focuses on the configuration manager that is dedicated to the dynamic reconfiguration of user interaction (both input devices and interaction techniques).

4. Configuration manager policy and modelling

This section presents the modelling of different policies to manage both input and output device configurations using the ICO formalism. As configuration management activities may occur at either runtime (while a user interacts with the application) or "pre-runtime" (e.g. just before starting an application or during a switchover of users), we present a pre-runtime policy involving explicitly input devices and a runtime policy explicitly managing output devices.

To support the illustration of both the input and output configurations, we use an application compliant with the "ARINC specification 661". This application features both simple

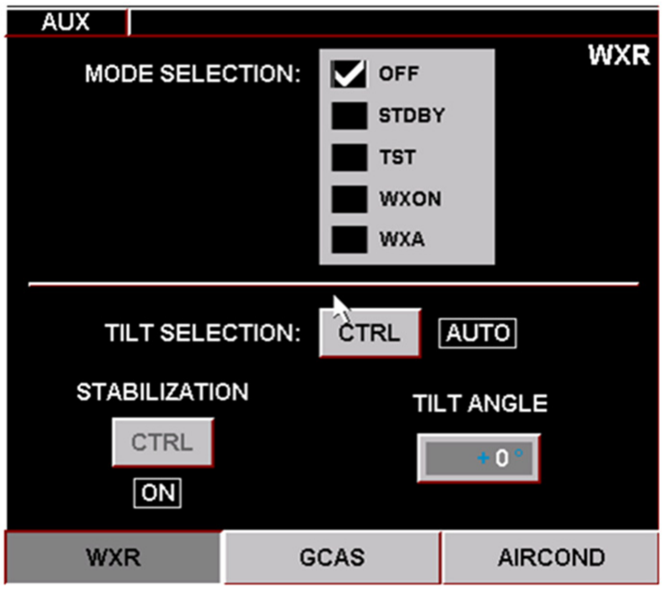


Fig. 7. The original MPIA application (with navigation by means of Tab at the bottom).

functionality and user interaction but is complicated enough to highlight the main points of this paper and to show how reconfigurations following hardware failures are managed. This user application (UA) is made up of 3 different pages containing 12 different widgets as defined by the "ARINC specification 661". It is important to note the usability or human factors issues are out of the scope of this paper as we focus here on engineering aspects of this application.

MPIA is a real user application (UA) aimed at handling several flight parameters and contains 3 pages called WXR, GCAS, and AIRCOND. WXR page is for managing weather radar information; GCAS is for ground anti-collision system parameters while AIRCOND deals with air conditioning settings. The application can be controlled (though not at the same time) by the pilot and the co-pilot via keyboard and mouse interaction using the KCCU (see Fig. 1). Each application window is made up of a set of widgets allowing users to modify the corresponding parameters. The original application only exploits one window and navigation between the 3 windows is made possible by means of a tab panel as shown in Fig. 7. The interested reader can find more information about MPIA and its specification using ICOs in [11,12].

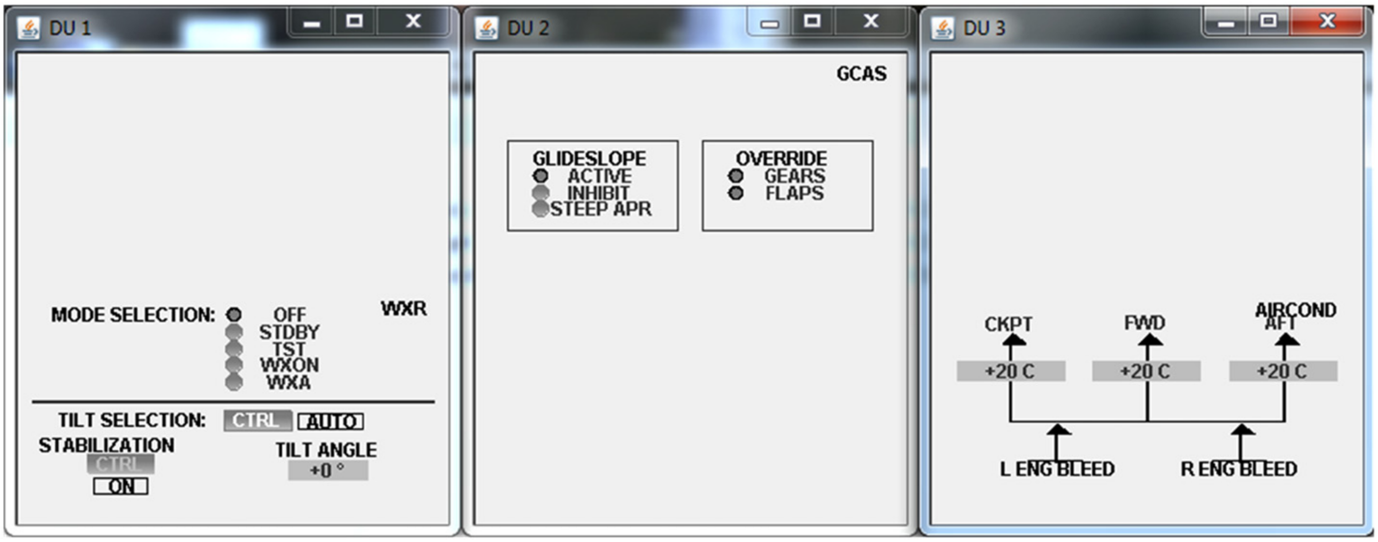


Fig. 8. The 3 panels of MPIA application (WXR, GCAS, and then AIRCOND).

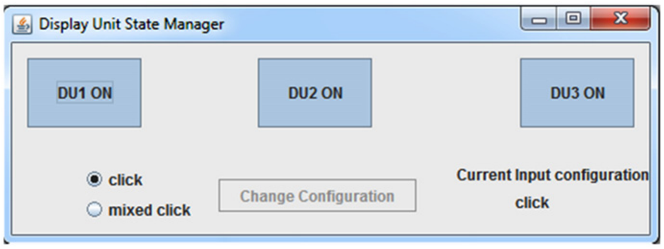


Fig. 9. The control panel for display unit failures (all DUs are On).

In the rest of the paper and in order to support the illustration of the output configuration, we propose a revised version of MPIA where each window is displayed on a different display unit (DU) in the cockpit making it possible to have the 3 pages of the application visible at the same time. In order to allow reconfigurations to occur, the DUs got enough space available for displaying the content of two MPIA windows. Indeed, in Fig. 8, both the upper part of DU1 and DU3, and the lower part of DU2 are able to accommodate additional content.

In order to simulate display units failures a controller has been designed and its control panel is presented in Fig. 9. This panel makes it possible to simulate failure both at input and output levels. The control panel features one button per display unit displaying the current status of the DU. Clicking on that button changes the state of the DU. For instance on that Figure the DUs are all in the state on. Clicking on the button named “DU1 ON” will shut down the DU1 and the button text will change to “DU1 OFF”. Using that control panel, by selecting one of the two predefined interaction techniques it is possible to switch between them. The current state of Fig. 9 shows that current input configuration allows users to interact with the MPIA application by using simple clicks.

4.1. *Input device configuration manager policy*

A possible use of reconfiguration is to allow customizing the interaction technique to make the application easier to manipulate. Even if it is out of the scope of the current version of the ARINC 661 specification, customization of interaction techniques may become necessary bringing a better user experience [14] in the same way as with personal computers. Customization becomes necessary too when continuity of interaction service has to be improved allowing users to carry on interacting with the system even though some input and output devices are out of order.

4.1.1. *Principle of the input device configuration manager policy*

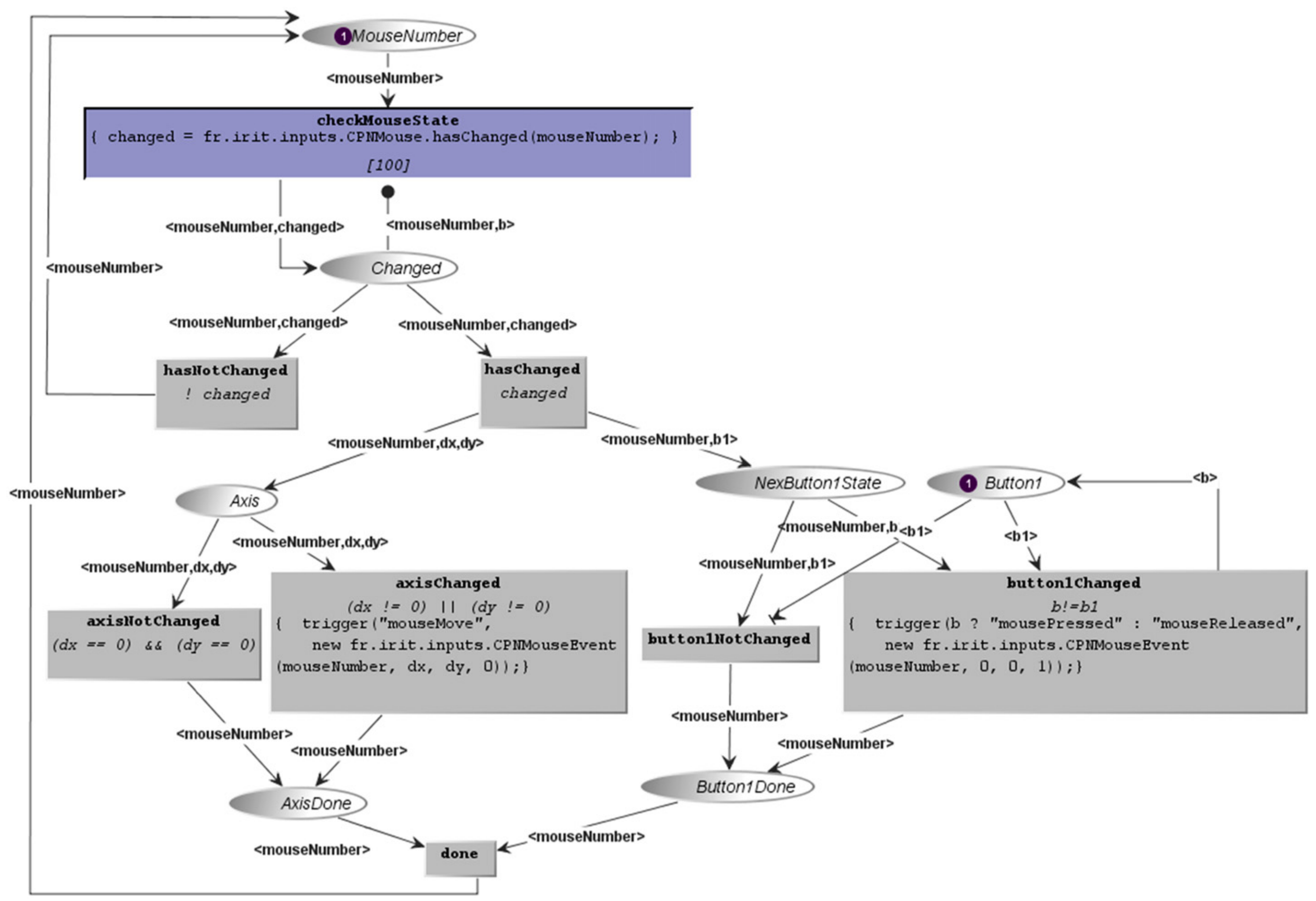
We focus only on a very simple scenario of input device configuration policy, which is based on the difference between a pilot and the associated co-pilot. The standard configuration allows the first officer (FO) and the pilot to interact at the same time on the various widgets of the applications running on the interactive display units of the cockpit. Selection of critical commands requires that the pilot and the FO interact within a short temporal window on the widget. While, on the user side, such interaction technique appears as a simple click for each user, on the system side it is handled as if one user was interacting with two mice and producing MixedClicks, i.e. a click with both mice on the same widget. If one KCCU fails then the interaction technique is reconfigured and MixedClicks are replaced by DoubleClicks for triggering critical commands.

The user interface server manages the set of widgets and the hierarchy of widgets used in the user applications. More precisely, the user interface server is responsible in handling the following:

- The creation of widgets.
- The graphical cursors of both the pilot and his co-pilot.
- The edition mode.
- The mouse and keyboard events and dispatching it to the corresponding widgets.
- The highlight and the focus mechanisms.
- …

As it handles much functionality, the complete model of the sub-server (dedicated in handling widgets involved in the MPIA User Application) is complex and difficult to manipulate without an appropriate tool, and cannot be illustrated with a figure.

Events received by the interaction server are in some way high-level events as they are not the raw events produced by the input devices drivers. In our architecture, the main role of an input configuration is the role of a transducer [1]; it receives raw events and produces higher-level events. The events used by the interaction server, and so produced by an input configuration are (*normalKey*, *abortKey*, *validationKey*, *pickup*,



```
{ changed = fr.irit.inputs.CPNMouse.hasChanged(mouseNumber); }
```

```
{  trigger("mouseMove",
   new fr.irit.inputs.CPNMouseEvent
(mouseNumber, dx, dy, 0));}
```

```
{  trigger(b ? "mousePressed" : "mouseReleased",
   new fr.irit.inputs.CPNMouseEvent
(mouseNumber, 0, 0, 1));}
```

Fig. 10. Model of the raw events handling for both configurations.

unPickup, *mouseDoubleClicked*, *mouseClicked*). These events are produced from the following set of raw events: *mouseMoved*, *mouseDragged*, *mousePressed*, *mouseReleased*, *mouseClicked*, and *mouseDoubleClicked* from the mouse driver, and *pickup* and *unPickup* from the picking manager.

4.1.2. Modelling of the input device configuration manager policy using ICO

Fig. 10 models the handling of raw events from the KCCU for the production of upper level events such as mouseMove, mousePressed, mouseReleased, etc. The model is common for the two interaction techniques, DoubleClick and MixedClick, each represented within their own model. Switching between these models is performed at the interaction technique level and not at the raw events level. This raw events model first tests the value of a variable "changed" defined in transition *CheckMouseState* (upper transition in Fig. 10) every chosen number of milliseconds (in this model, 100 ms) in order to verify if the state of the mouse has changed since the previous check. According to the value of the variable, transition *hasNotChanged* or *hasChanged* will fire.

Following this, there are two further tests, according to the movement of the mouse and the state of the mouse button. The movement test is modelled using transition *axisChanged* (left hand side of the model) according to *x*,*y* coordinates (mouseMove). Transition *buttonChanged* (right hand side of the model) checks to see if there has been a change in the state of the mouse button, which produces mousePressed or mouseReleased events. Only the left mouse button is considered in this example to reduce the complexity of the model. After the *axisChanged* and *buttonChanged* tests, transition *done* is fired placing a token in place *MouseNumber* ready to restart the simulation.

The model in Fig. 11 presents how low level events produced are combined at the interaction technique level to produce higher-level events. Transitions *mousePressed_t*1 and *mouseReleased_t*1 receives events from transition *buttonChanged* modelled in the "raw events" model shown in Fig. 10. The left part of this model produces a single click from a mousePressed and a mouseReleased from mouse1 (i.e. Pilot's mouse), while the right hand part of the model performs the same behaviour for mouse2 (i.e. first officer's mouse). The model states that if a MouseClick is performed (by either person) which starts a timer, and a second event MouseClick (performed with the other mouse) is received before the end of the timer, then the model produces a MixedClick event (transition *triggerMixedClick* at the bottom of the figure).

Fig. 12 represents the DoubleClick interaction technique in the degraded mode, i.e. when only one KCCU is available. The model receives events mousePressed, mouseReleased and mouseMoved from the raw events model presented in Fig. 10. They are then processed in order to be able to raise DoubleClick events, which occur when the KCCU has been pressed and released twice within a predefined temporal window and without any mouseMove event in-between.

Fig. 13 presents the model responsible for the management of the configurations. The basic principle of the model is that the current configuration has to be removed (unregistered part of the model on the right hand side of the figure) before the new desired configuration is set (register configuration part of the model on the left hand side of the figure).

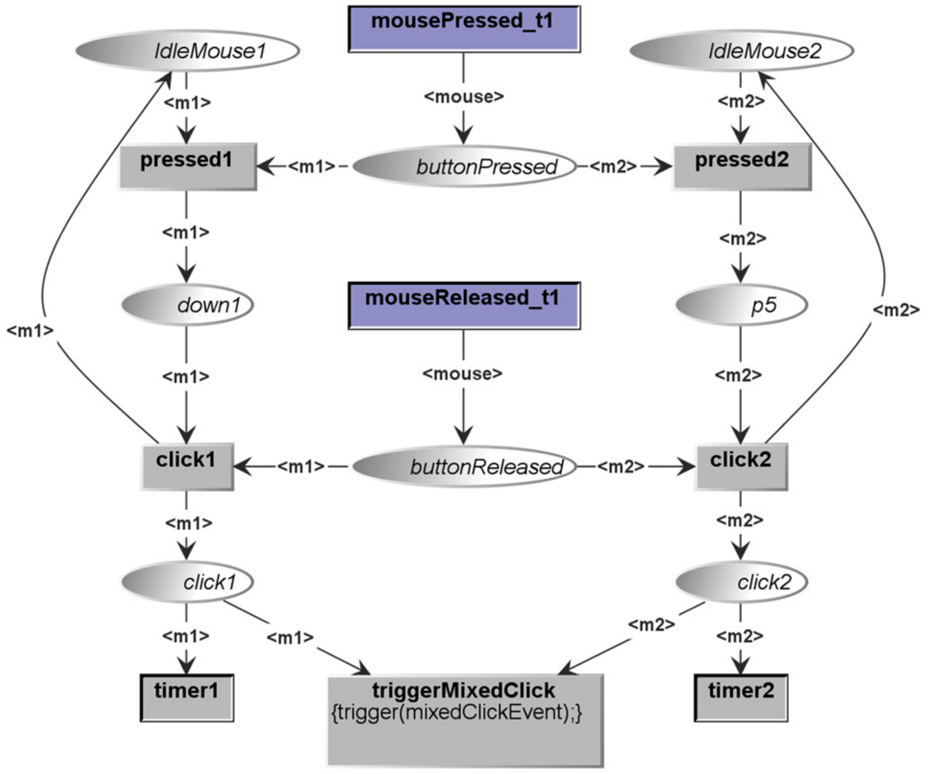


Fig. 11. Model of the mixed (both KCCU) click configuration.

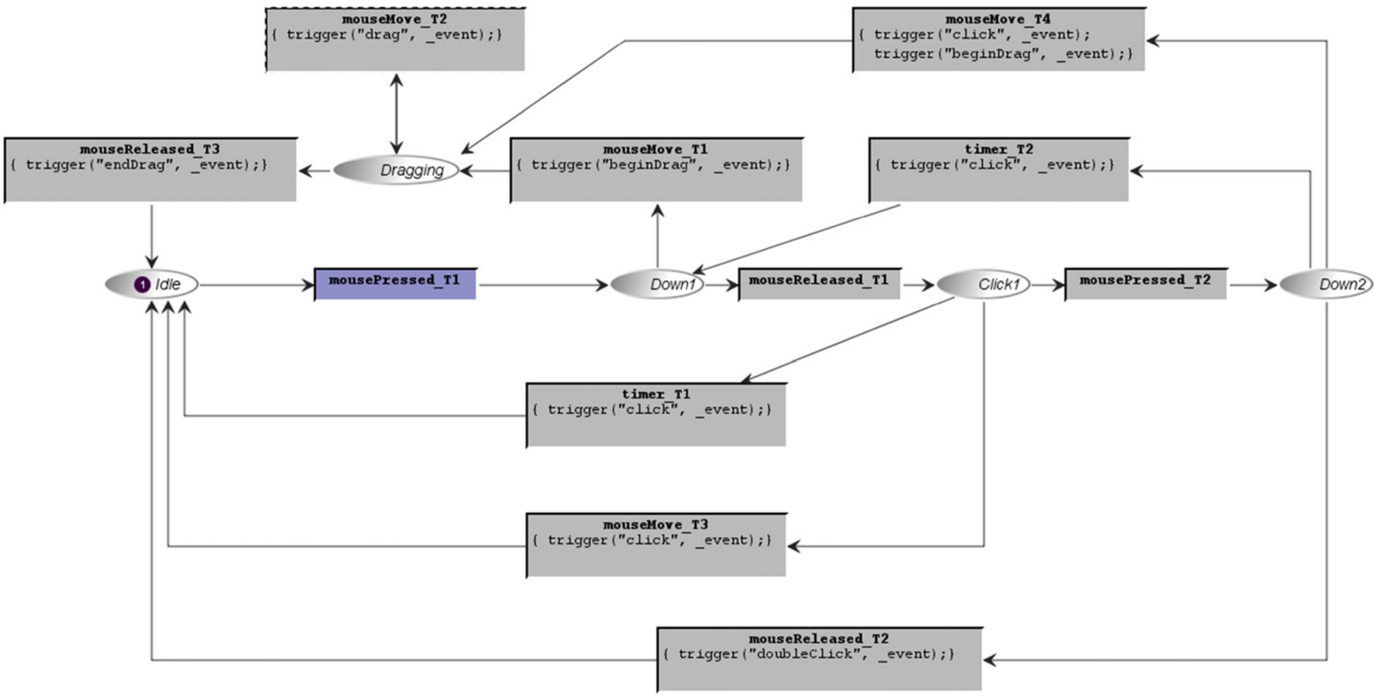

mouseMove_T2
{ trigger("drag", _event);}
mouseMove_T4
{ trigger("click", _event);
trigger("beginDrag", _event);}
mouseReleased_T3
{ trigger("endDrag", _event);}
Dragging
mouseMove_T1
{ trigger("beginDrag", _event);}
timer_T2
{ trigger("click", _event);}
1 Idle
mousePressed_T1
Down1
mouseReleased_T1
Click1
mousePressed_T2
Down2
timer_T1
{ trigger("click", _event);}
mouseMove_T3
{ trigger("click", _event);}
mouseReleased_T2
{ trigger("doubleClick", _event);}

Fig. 12. Model of the DoubleClick configuration.

The four places in the central part of Fig. 13 (*MouseDriver*, *KeyboardDriver*, *PickingManager*, and *InteractionServer*) contain a reference to the set of models corresponding to the input devices and to the interaction server. When a new configuration is requested to be set (that can be performed in our example by using the control panel presented in Fig. 9 in order to switch between the two predefined input configurations), a token with a reference to the new configuration is put in place *NewConfiguration*. Following this, the four transitions highlighted on the left hand side are fired in sequence (could be modelled as parallel behaviour as well) in order to register the new configuration as a listener of the events produced by the mouse driver, the keyboard driver, and the picking manager. The fourth transition registers the interaction server as a listener of the events produced by the new configuration.

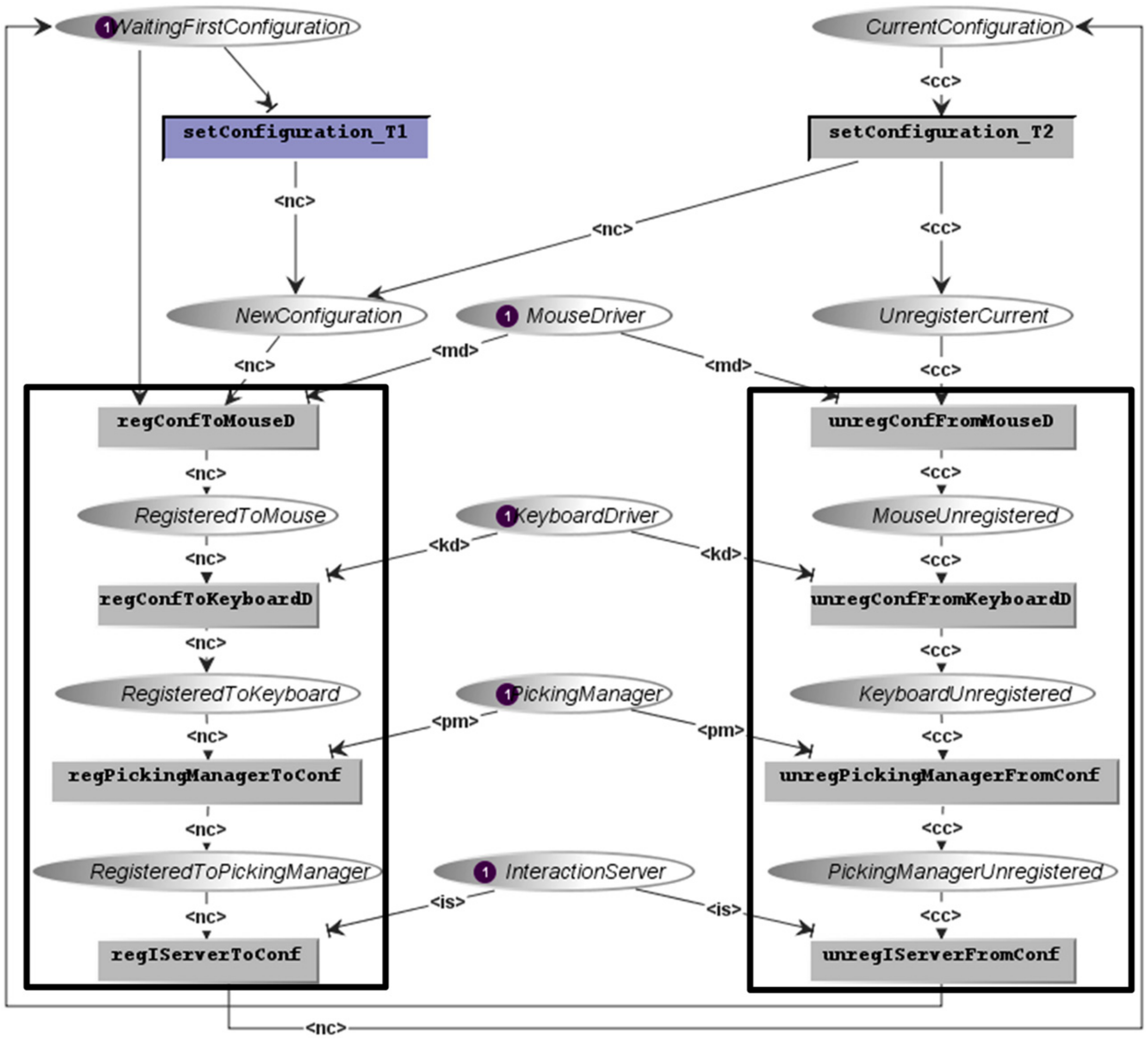


Fig. 13. ICO model of the configuration manager part dedicated to the input devices.

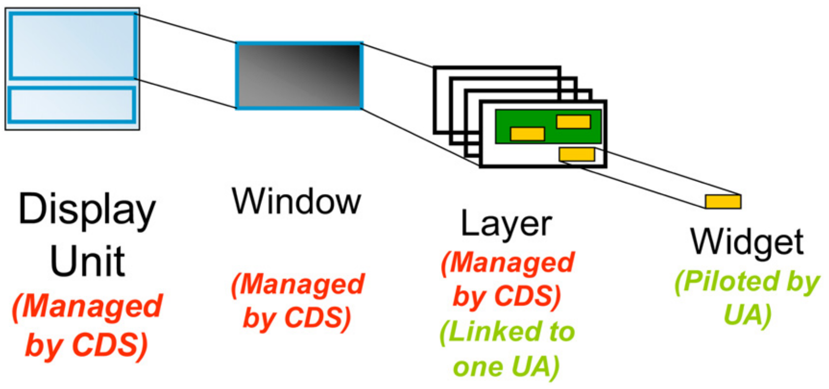


Fig. 14. ARINC 661 specification windowing architecture.

If a configuration is already set, when the new configuration is requested, a token is put in place *UnregisterCurrent* in order to fire the four transitions highlighted on the right hand side, corresponding to unregister from the different models, in parallel with registering the new configuration.

4.2. *Output device configuration manager policy*

A policy has to be defined on what kind of changes have to be performed when a display unit fails. This policy is highly based on the windowing system adopted by the standard ARINC 661 specification.

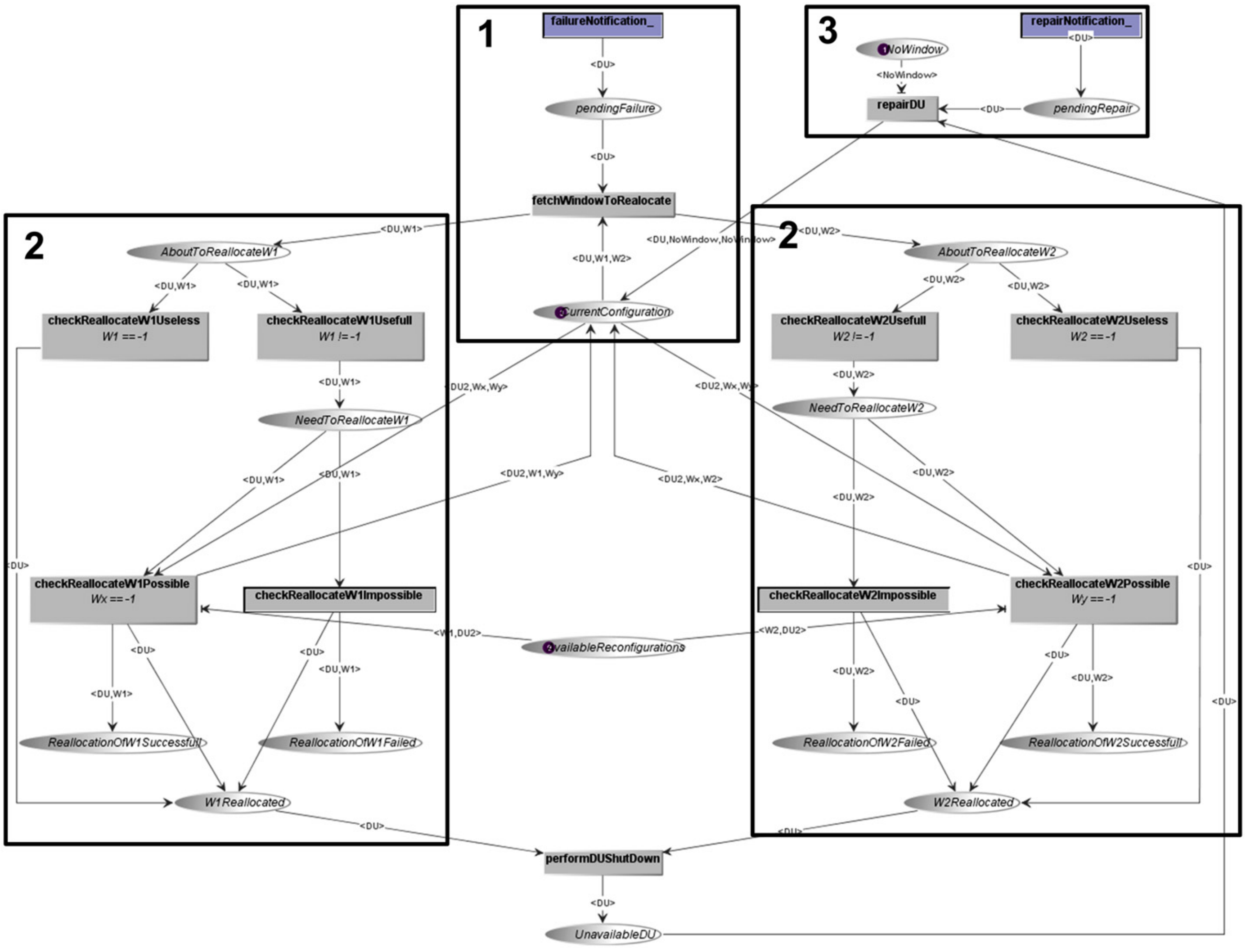

1
failureNotification_
pendingFailure
fetchWindowToRealocate
CurrentConfiguration
3
repairNotification_
NoWindow
repairDU
pendingRepair
2
AboutToReallocateW1
checkReallocateW1Useless W1 == -1
checkReallocateW1Usefull W1 != -1
NeedToReallocateW1
checkReallocateW1Possible Wx == -1
checkReallocateW1Impossible
ReallocationOfW1Successfull
ReallocationOfW1Failed
W1Reallocated
AvailableReconfigurations
AboutToReallocateW2
checkReallocateW2Usefull W2 != -1
checkReallocateW2Useless W2 == -1
NeedToReallocateW2
checkReallocateW2Impossible
checkReallocateW2Possible Wy == -1
ReallocationOfW2Failed
ReallocationOfW2Successfull
W2Reallocated
performDUShutDown
UnavailableDU

Fig. 15. An ICO model of a configuration manager.

The ARINC 661 specification uses a windowing concept, which can be compared to a desktop computer windowing system, but with many restrictions due to the aircraft environment constraints (see Fig. 14). The windowing system is split into 4 components:

- the display unit (DU) which corresponds to the hardware part,
- the format on a display unit (DU), consists of a set of windows and is defined by the current configuration of the CDS,
- the window is divided into a set of layers (with the restriction of only one layer activated and visible at a time) in a given window,
- the widgets are the smallest component on which interaction occurs (they correspond to classical interactors on Microsoft Windows system such as command buttons, radio buttons, check buttons, etc.).

4.2.1. Principle of the output device configuration manager policy

When a display unit fails, the associated windows might have to be reallocated to another display unit. This conditional assertion is related to the fact that

- There might be not enough space remaining on the other display units (DU).
- The other applications displaying information on the other DU might have a higher priority.

The ARINC 661 specification does not yet propose any solution to this particular problem but it is known as being critical and future supplements of the ARINC 661 specification may address this issue.[5] However, at the application level, the UADF (User Application Definition File) defines a priority ordering among the various layers included in the user application. At any given time, only one layer can be active. At runtime, the activation of a new layer must be preceded by the deactivation of the current layer.

The policy that we have defined lays in the definition of a set of compatible windows, i.e. windows offering a greater or equal display size. This is related to a strong limitation imposed by ARINC 661, which states that some methods and properties are only accessible at design time, i.e. (according to ARINC 661 specification vocabulary) when the application is initialized. Methods and properties related to widget size are not available at runtime and thus any reorganisation of widgets within a window is not possible.

[5] ARINC 661 specification is continuously evolving since the first proposal. The draft 2 of supplement 3 containing 374 pages has been released on August 15th 2007.

4.2.2. Modelling of the output device configuration manager policy using ICO

In Fig. 15, we present an implementation of the previously defined policy for handling output devices using the ICO formalism. This model is a subpart of the complete configuration manager that can be added to the previous modelling we have done and thus be integrated in the behaviour of our (cockpit display system) CDS model [13].

The model presented here is based on a very simple case:

- 1 layer per window.
- 2 windows per display unit.
- Each display unit is divided into two identical parts, one being the top part and the other one being the bottom part.

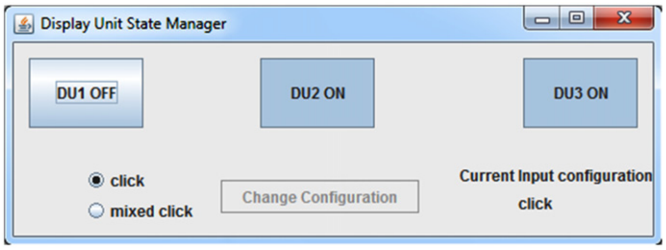


Fig. 16. The control panel for display unit failures (DU1 is Off and DU2 and DU3 are still On).

Fig. 15 presents the ICO model corresponding to the current state of the DUs. All the DUs are in the state "ON" (place *CurrentConfiguration* at the centre of the model holds 3 tokens and place *UnavailableDU* at the bottom of the model holds no token):

1. Each time one DU button is pressed on the control panel Fig. 9 (simulating an event that may be triggered by a sensor in charge of detecting DU failure), the transition failureNotification_ is fired depositing a token (holding the number of the DU on which a failure occurred) in place *PendingFailure*.
2. The configuration manager finds a compatible window for a reallocation of the contained layers (here all compatible windows are listed at creation time and stored in place *AvailableReconfigurations*). When a reconfiguration is chosen the current configuration is modified (the content of place *CurrentConfiguration* is updated). The model presented in Fig. 15 contains two information flows: one being responsible for reallocating the upper part of the DU (the left part of the model) and the other one being responsible for reallocating the bottom part of the DU (the right part of the model).
3. In the same way as the DU failure can be detected, the DU repairing can be notified. The modelled behaviour of the configuration manager updates the set of available DU for reallocation (by setting a token in place *CurrentConfiguration*) but does not immediately exploit the repaired DU. This is a

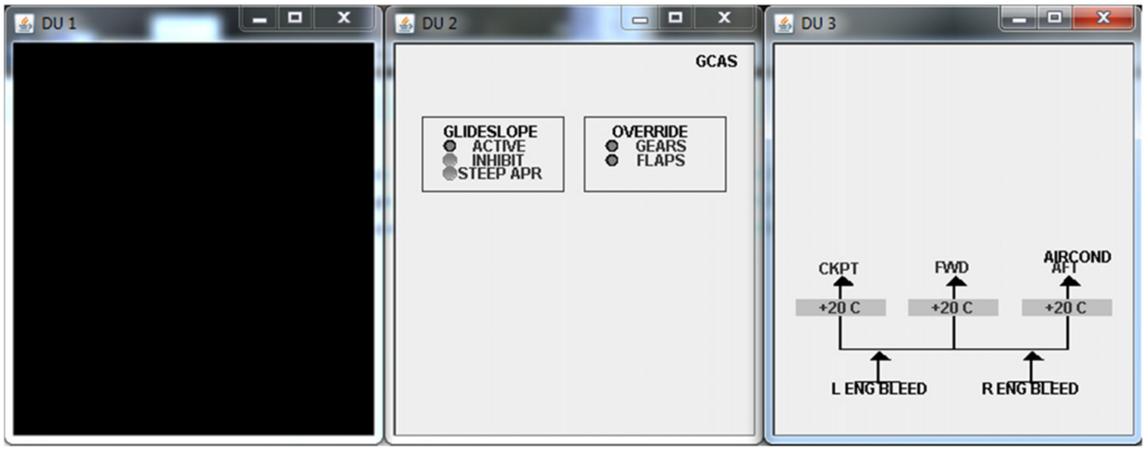


Fig. 17. Cockpit status after failure of DU1 (DU1 is Off and DU2 and DU3 are still On).

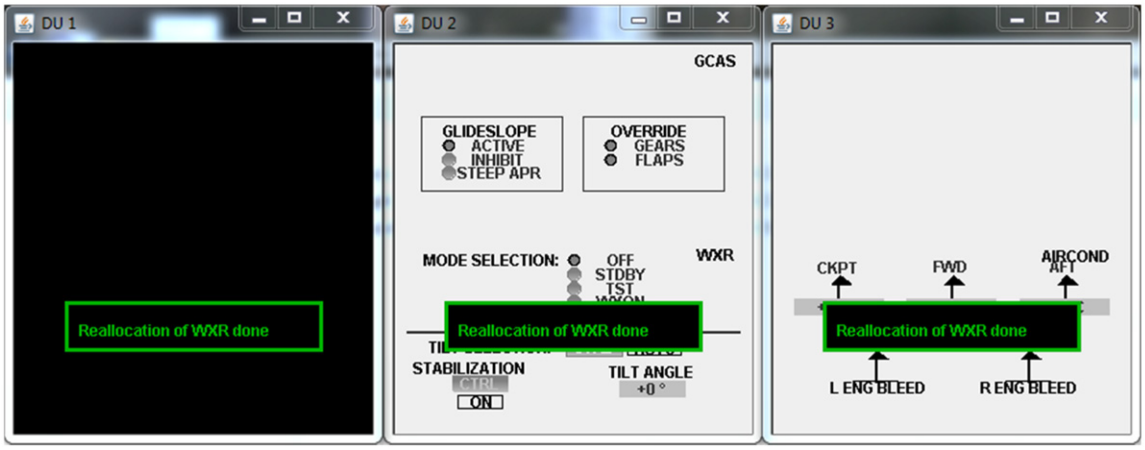


Fig. 18. Cockpit status after reconfiguration following the failure of DU1.

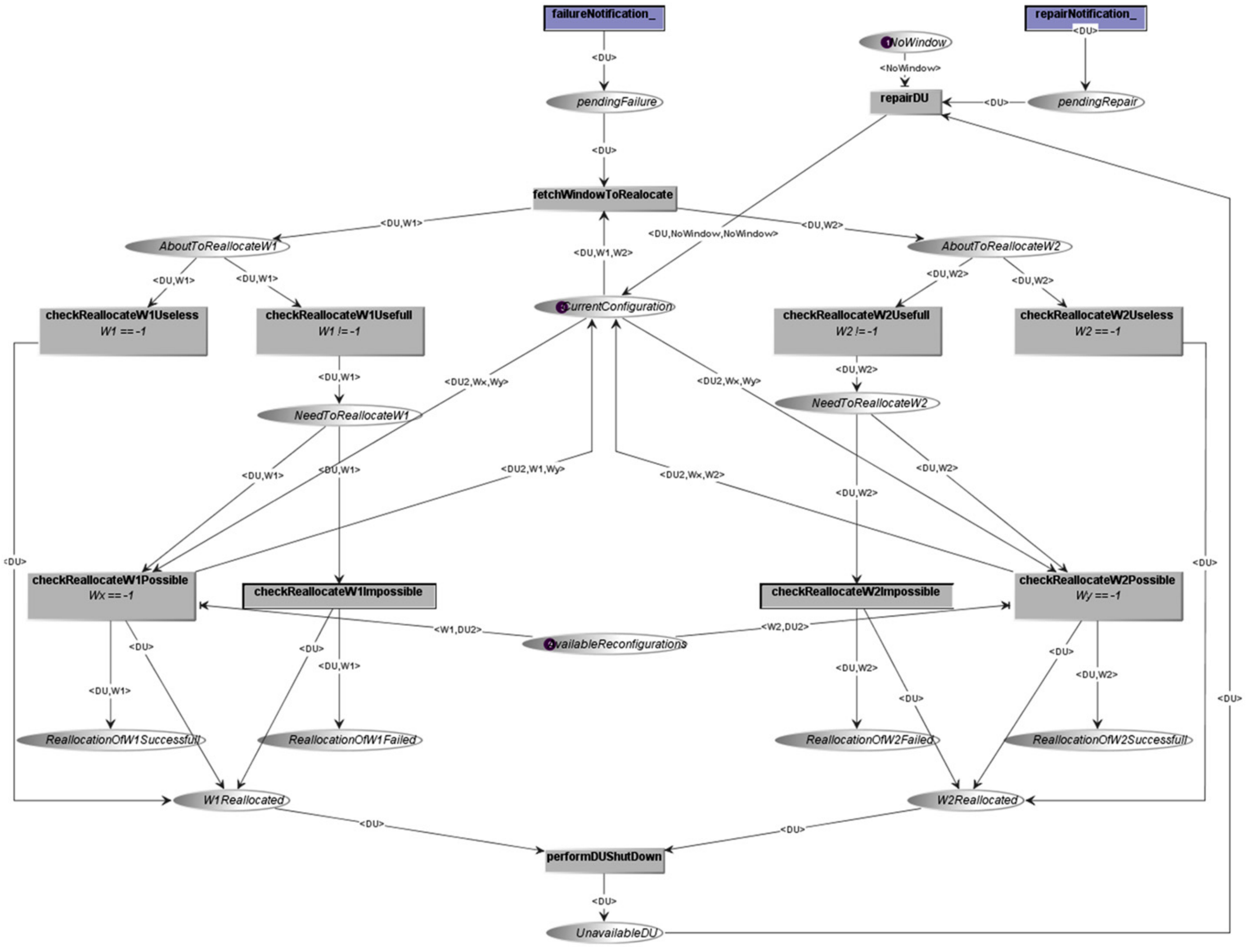


Fig. 19. Behavioural modelling of the system following reconfiguration after DU1 failure.

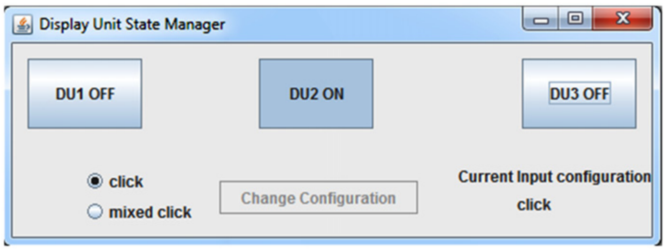


Fig. 20. The control panel for display unit failures (DU1 and DU3 are Off while DU2 is still On).

modelling choice and in that model only failure notifications trigger reallocation of layers on the DUs.

To make this configuration mechanism more concrete, the following paragraphs describe two scenarios of reconfiguration.

(1) *Simulation of failure on display unit 1 (DU1) and reconfiguration*

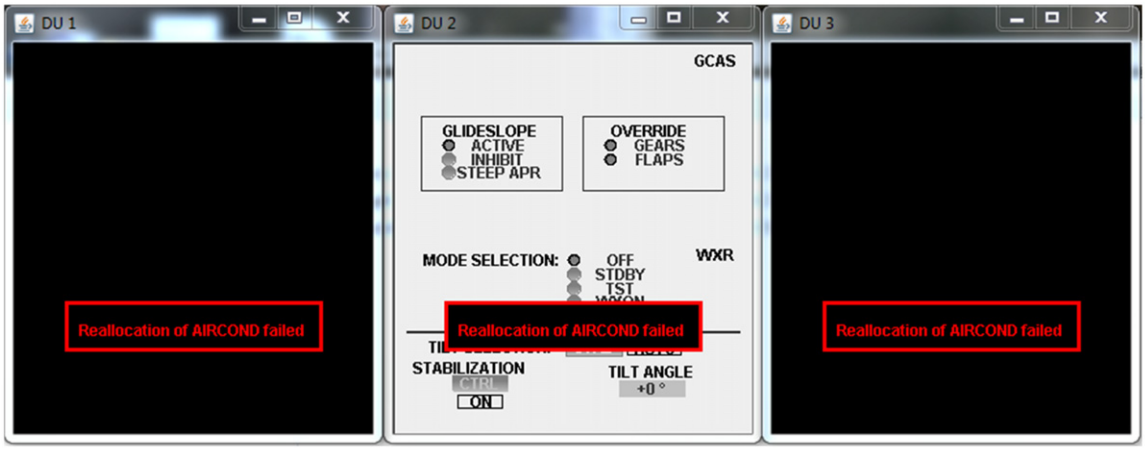
This section presents the evolution of the model according to a failure on display unit 1 and presents both how the reconfiguration is performed and how this is notified to the cockpit crew.

Fig. 16 shows the presentation of the control manager after the display unit 1 has been set to a failure mode. DU2 and DU3 are still On but DU1 is Off.

The corresponding state of the 3 DUs in the cockpit (as far as MPIA application is concerned) is presented in Fig. 17. Without reconfiguration of the content of the available DUs it is not possible anymore for the crew to access and modify parameters corresponding to the WXR part of MPIA.

Fig. 18 shows the reconfiguration of the DUs and the notification presented to the crew stating that WXR page has been allocated to a new DU. Here we only present information related to the reallocation while the ICO model contains all the information to provide crew with more information such as the following: which DU failed, on which DU WXR has been reallocated, and on which part of the DU it has been reallocated. However, in this kind of safety critical applications, the users are extensively trained on the behaviour of the system they are using including allowed reconfigurations. This is very different from other domains such as plastic interfaces [15] where reconfigurations are occurring without previous training of users.

The model in Fig. 19 presents the state of the application after reconfiguration. One DU is not available (there is one token in place *UnavailableDU* at the bottom of the model) and there are only 2 tokens in the place *CurrentConfiguration* (meaning that only 2 DUs are available). In addition, the right-hand upper part of the model now makes it possible for the failed DU to be repaired and exploited again (after a repairNotification is received).



Reallocation of AIRCOND failed

Reallocation of AIRCOND failed

Reallocation of AIRCOND failed

Fig. 21. Cockpit status after reconfiguration failure following the failure of both DU1 and DU3.

(2) *Simulation of an additional failure on display unit 3 (DU3) and reconfiguration impossible*

This section presents another aspect of reconfigurations, i.e. system behaviour when not enough resources are available (either at input or output level). In this example, we present a reconfiguration failure after the consecutive failure of both DU1 and DU3.

Fig. 20 presents the control panel for DU management right after the DU3 OFF button has been pressed. Only DU2 remains in the On mode. Fig. 21 shows the rendering of this state change in the cockpit. In that state, there was no "slot" available in DU2 for allocating the AIRCOND page (that was displayed in DU3). A message is thus displayed to the crew letting them know that AIRCOND page has not been successfully reallocated to any available DU.

As for the successful reallocation presented in previous section, here again the crew has been trained to unsuccessful reconfiguration and can follow recovery procedures to handle safely the aircraft. This is an important aspect of the contribution of this paper: we do not propose a method for guaranteeing reliable behaviour of interactive cockpit application. We only propose a mean for describing several configurations and for dynamically switching from one configuration to another one when an adverse event (such as an input and/or output device hardware failure) occurs.

In the case study presented in this section we have not presented how priorities between configurations can be handled. Indeed, in case of an insufficient number of resources, some priorities between the possible configurations should be defined and should be set. For instance in our case study, it could have been considered that AIRCOND has a higher priority that WXR and thus both AIRCOND and GCAS should be displayed if only one DU is available. This is easily modelled using our Petri net based approach but this is beyond the scope of this paper.

5. Conclusion and perspectives

This paper has addressed the issue of user interface reconfiguration in the field of safety critical command and control systems. The application domain was civil aircraft cockpit systems compliant with the ARINC 661 specification (which defines communication protocols and window management policy for cockpit displays systems). This work complements previous work we have done on this topic [13] by extending the behavioural model of cockpit display system with fault-tolerant behaviour and with a generic architecture allowing static configuration as well as dynamic reconfiguration of interaction techniques. Further work we currently carry out deals with several extensions with respect to what has been presented in the paper:

- Provide means for the specification of reconfiguration policies including the definition of priorities between user applications and within a given user application (as discussed at the end of previous section).
- Provide analysis methods and tools for assessing the efficiency of the policies both in terms of user behaviour (i.e. how possible is it for a pilot to perform a given set of tasks when a given reconfiguration policy is adopted) and in terms of system behaviour (i.e. how is it possible to identify the overall performance of a given policy).
- Take into account more complex failure modes such as partial failure of a device (e.g. a DU only displays a subset of colour) or difficult to identify failure (e.g. HeisenBugs [22]).

The validation of the work presented here is under investigation by means of a joint project involving additionally Airbus and LAAS. This is more complex and broader work as it involves other standards such as ARINC 653 [4] (which deals with operating system temporal and spatial partitioning) and embedding fault tolerant mechanisms.

It is important to note that the fault-tolerance addressed in the current paper is only related to the user interface part of the cockpit display system even though it takes into consideration input and output devices as well as the behaviour of the window manager.

While the safety aspects have not been at the centre of the paper the entire work presented here serves as a basis for supporting the design and construction of safer interactive embedded applications and to improve operations.

Acknowledgements

This work was partly supported by the EU funded Network of Excellence ResIST http://www.resist-noe.eu under Contract no. 026764. We would like to thank the reviewers for the detailed comments which definitively improved the structure of the paper.

References

[1] Accot J, Chatty S, Maury S, Palanque P. Formal transducers: models of devices and building bricks for highly interactive systems. In: Proceedings of the fourth eurographics workshop on "design, specification and verification of interactive systems", Spain: Springer Verlag; 5–7 June 1997.

[2] ARINC 661. Cockpit display system interfaces to user systems. ARINC Specification 661. Prepared by Airlines Electronic Engineering Committee; 2002.

[3] ARINC 661-2. Cockpit display system interfaces to user systems. ARINC Specification 661–2. Prepared by Airlines Electronic Engineering Committee; 2005.

[4] ARINC Specification 653P1-2. Avionics application software standard interface. Prepared by Airlines Electronic Engineering Committee; 2006.

[5] Bass L, et al. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. SIGCHI Bulletin 1992;24(1):32–37.

[6] Bastide R, Philippe Palanque, Ousmane Sy, Duc-Hoa Le, David Navarre. Petri net based behavioural specification of CORBA systems. In: Proceedings of the international conference on application and theory of Petri nets ATPN'99, Williamsburg (USA): LNCS Springer Verlag; 1999. p. 66–83.

[7] Bastide R, Palanque P, Sy O, Navarre D. Formal specification of CORBA services: experience and lessons learned. In: Proceedings of the 15th ACM SIGPLAN conference on object-oriented programming, systems, languages, and applications (Minneapolis, Minnesota, United States). OOPSLA '00. ACM, New York, NY; 2000.

[9] Bastide R, Palanque P, Sy O, Le D-H, Navarre D. PetShop a case tool for Petri net based specification and prototyping of Corba systems. Tool demonstration with application and theory of Petri nets ATPN'99. Williamsburg (USA): LNCS Springer Verlag; 1999.

[10] Bastide R, Navarre D, Palanque P, Schyn A, Dragicevic P. A model-based approach for real-time embedded multimodal systems in military aircrafts. In: Proceedings of the sixth international conference on multimodal interfaces (ICMI'04) October 14–15, 2004. Pennsylvania State University, USA: ACM Press; 2004.

[11] Barboni E, Navarre D, Palanque P, Basnyat S. Exploitation of formal specification techniques for ARINC 661 interactive cockpit applications. In: Proceedings of HCI aero conference, HCI Aero 2006, Seattle, USA; September 2006.

[12] Barboni E., David Navarre, Philippe Palanque & Sandra Basnyat. A formal description technique for interactive cockpit applications compliant with ARINC specification 661. In: Proceedings of the SIES 2007—IEEE second international symposium on industrial embedded systems July 4–6, 2007, Lisbon, Portugal. pp. 184–94.

[13] Barboni, E, Conversy, S, Navarre, D, Palanque, P. Model-based engineering of widgets, user applications and servers compliant with ARINC 661 specification. In: Proceedings of the 13th conference on design specification and verification of interactive systems (DSVIS 2006), LNCS: Springer Verlag; 2006.

[14] Csíkszentmihályi M. Flow: the psychology of optimal experience. New York: Harper and Row; 1990.

[15] Thevenin D, Coutaz J. Plasticity of user interfaces: framework and research agenda. In: Proceedings of interact'99, vol. 1, Edinburgh: IFIP TC 13, IOS Press; 1999. p. 110–7.

[16] Eirinaki M, Lampos C, Paulakis S, Vazirgiannis M. Web personalization integrating content semantics and navigational patterns. In: WIDM '04: Proceedings of the 6th annual ACM international workshop on web information and data management, New York, NY, USA: ACM Press; 2004. p. 72–9.

[18] Genrich HJ. Predicate/transitions nets. In: Jensen K, Rozenberg G, editors. High-levels petri nets: theory and application. Springer Verlag; 1991. p. 3–43.

[19] Hollnagel E, Woods D D, Leveson N. Resilience engineering: concepts and precepts. Ashgate; 2006. 397.

[20] Lakos C. Language for object-oriented Petri nets. #91-1, Department of Computer Science, University of Tasmania; 1991.

[21] MacKenzie S, Zhang SX, Soukoreff RW. Text entry using soft keyboards. Behaviour and information technology 1999;18:235–44.

[22] Musuvathi M., Shaz Qadeer, Thomas Ball, Gel̀ rard Basler, Piramanayagam A. Nainar, and Iulian Neamtiu. Finding and reproducing heisenbugs in concurrent programs. In: Proceedings of the eighth USENIX symposium on operating systems design and implementation, 2008. p. 267–80.

[23] Navarre D, Palanque P, Bastide RA. Tool-supported design framework for safety critical interactive systems in interacting with computers, vol. 15/3. Elsevier; 2001 pp. 309–328.

[24] Navarre D, Palanque P, Ladry J, Barboni E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Trans. Comput.-Hum. Interact. 2009;16(4): 1–56.

[25] Navarre D, Palanque P, Bastide R. A formal description technique for the behavioural description of interactive applications compliant with ARINC 661 specifications. HCI-Aero'04 Toulouse, France, 29 September–1 October 2004.

[26] Palanque P, Bernhaupt R Navarre D, Ould M, Winckler M. Supporting usability evaluation of multimodal man-machine interfaces for space ground segment applications using Petri net based formal specification. In: Proceedings of the ninth international conference on space operations, Rome, Italy; June 18–22; 2006.

[27] Petri CA. Kommunikation mit automaten. Darmstadt: Technical University; 1962.

[28] Reason J. Human error. Cambridge University Press; 1990.

[29] Ríos SA, Velásquez J D, Yasuda H, Aoki T. Web site off-line structure reconfiguration: a web user browsing analysis, in knowledge-based intelligent information and engineering systems. LNCS 2006;4252:371–8.

[31] Van Dam A. Post-WIMP user interfaces. Communications of the ACM, February 1997, 40(2); 1997. p. 63–7.