A Formal Description Technique for Interactive Cockpit Applications Compliant with ARINC Specification 661

Eric Barboni, David Navarre, Philippe Palanque & Sandra Basnyat
LIIHS-IRIT
Université Paul Sabatier (Toulouse III)
118, route de Narbonne, 31062 Toulouse Cedex 4, France barboni,navarre,palanque,basnyat@irit.fr

***Abstract*— The purpose of the ARINC specification 661 [1] is to define interfaces to a Cockpit Display System (CDS) targeting new aircraft installations. ARINC 661 provides precise information for communication protocols between application and user interface components (called widgets) as well as precise information about the widgets themselves. However, no information is given on the behavior of these widgets and on the behavior of an application made up of a set of such widgets. This paper presents a formal description technique called Interactive Cooperative Objects to define in a precise and non-ambiguous way such behaviors. This description technique also defines the relationships between the behavioral description and the user interface. We show the benefits of such a notation for the specification of interactive cockpit applications and we introduce each modeling concept on a small example.**

Keywords - ARINC 661 specification, formal description techniques, interactive software engineering, Interactive Cockpits.

I. INTRODUCTION

In order to provide pilots with additional means for handling more and more complex embedded equipments, new aircrafts are provided with interactive applications. This is not only true for military aircraft (that have always been ahead with new equipment) but also for civil aircrafts [9], [16].

Embedding interactive applications in civil and military cockpits is expected to provide significant benefits to the pilots by providing them with easier to use and more efficient applications increasing the communication bandwidth between pilots and systems. However, this technological enhancement brings inside the cockpit a set of problems that have been well known in the field of interactive systems. The more freedom given to the user, the more complex the system is in order to handle such freedom. Such problems have been identified in the beginning of the 80's through the "direct manipulation" paradigm [24] which raises the issues of state space explosion and very limited test coverage. The ARINC specification 661 (see next section), aims at providing a common ground for building interactive applications in the field of aeronautical industry. However, this standard only deals with part of the issues raised. The aim of this paper is to present a formal description technique to be used as a complement to ARINC 661 for the behavioral description of interactive applications.

The paper is structured as follows. The next section introduces the application domain of interactive cockpits. Section 3 presents the ICO formalism, a formal description technique for the design of safety critical interactive applications. The applicability of ICOs to cockpit display system and its compatibility with ARINC specification 661 is presented in section 4 by means of a case study. The last section of the paper deals with conclusions and perspectives to this work.

II. INTERACTIVE COCKPITS CONSTRAINTS

A. Constraints from standards

The purpose of the ARINC specification 661 [1] is to define software interfaces to a Cockpit Display System (CDS) used in interactive cockpits that have been deployed by several aircraft manufacturers including Airbus, Boeing and Dassault.

Among the objectives of this standard we find:

- The minimization of the cost of adding new display function to the cockpit during the life of an aircraft.
- The introduction of interactivity in the cockpit, providing a basis to standardize the Human Machine Interface (HMI) in the cockpit.

ARINC 661 defines two interfaces between the Cockpit Display System (CDS) and the aircraft systems to provide a clear separation between them. The first interface is between the avionics equipment and the display system graphics generators, and the second is a way to define the symbology and its related behavior. The CDS provides graphical and interactive services to user applications (UA) within the flight deck environment. The interactive applications will be executed on Display Units (DU) and interaction with the pilots will take place through the use of keyboard and graphical input devices like the Keyboard Cursor Control Unit (KCCU) developed by Thales Avionics[1].

[1] Description available and last accessed May 20th 2007 http://www.flightglobal.com/articles/2006/09/25/209189/airbus-a380-flight-test-in-full-our-report-from-toulouse.html

1-4244-0840-7/07/$20.00 ©2007 IEEE

A user application is then defined as a system that has a two-way communication with the CDS:

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

Figure 1. KCCU (Keyboard Cursor Control Unit) for interactive cockpits taken from AWST (September 27th 2004)

The ARINC specification 661 provides the definition of the software interface between the CDS and the UAs. It also represents the expression of airline desires in the form of guidance material. Designers should interpret this standard in terms of the "need" for specific design practices rather than practices that "must" be met under all circumstances.

The ARINC specification 661 does not provide the specification of the "look and feel" of any graphical information and does not document the activities to be carried out in order to design a CDS.

B. *Domain constraints*

In addition to the software engineering-related issues identified above, the fact that interactive applications are aimed at keeping the human in the loop introduces additional constraints such as, usability and error-tolerance. These properties are to be dealt with as carefully as reliability issues that this paper targets. Information on how to address usability, safety and reliability at the same time can be found in [21] and [19].

III. ICO FORMALISM

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 [18] for a complete presentation of this formal description technique.

The ICO formalism is a formal description technique dedicated to the specification of interactive systems [6]. 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 [12] to describe their dynamic or behavioral aspects.

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

A. *Informal Presentation*

ICOs are dedicated to the modeling 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) [6]. The following paragraphs briefly recall basics about Petri nets and their extensions in order to finally present the main features of the ICO formal description technique.

1) *Petri nets (PN)*

Petri nets were initially introduced in C.A Petri's PhD thesis in1962 and are used for modeling discrete event systems. Petri nets are a formalism that features a complete equivalence between a graphical and an algebraic representation. We present here the basics of Petri nets through their structural aspects and their dynamic behavior.

a) *Structure*

A Petri net is an oriented graph composed of two disjoint sets of nodes and a set of arcs:

- **Places** (represented by ellipses) symbolize states variables holding untyped tokens which symbolize values.
- **Transitions** (represented by rectangles) symbolize actions and state changes.
- **Arcs** link places to transitions (called input arcs), or transitions to places (called output arcs), and symbolize the flow of tokens through the Petri nets. Arcs may be given integer values which are described as the weight of the arc, e.g. the quantity of tokens that is consumed following this arc.(This sentence is hard to understand!!!)

The global state of the modeled system is fully represented by the distribution of tokens across the set of places (called marking).

b) *Dynamic behaviour*

Given a marked Petri net, its behavior is expressed in terms of a token game. The token game defines two basic rules, the enabling rule and the firing rule:

- **Enabling of transition:** the enabling rule involves input arcs of a transition. A transition is enabled if each input place (places linked to the transition with input arcs) contains at least as many tokens as the weight of the input arcs it is linked to.
- **Firing of transitions:** firing a transition leads to the removal of as many tokens as the weight of the corresponding input arcs from its input places and then setting into the output places as many tokens as the weight of the corresponding output arcs.

These rules illustrate one of the main properties of Petri nets, called locality of enabling and firing of transitions. This property makes Petri nets able to model true concurrent systems.

2) Object Petri Nets(OPN)

Classic Petri nets do not easily allow describing data. The introduction of Object Petri nets (OPN) [15], [25] provides a means to handle more complex data structure using the Object paradigm.

a) Structure

- **Typed place and tokens:** each place of an OPN is a typed tuple that may hold a set of tokens which are tuples of values of the corresponding type.
- **Transition with Actions and Preconditions:** Transitions may be constituted of a precondition (expressed as a predicate depending of the values held by the input places) and actions on the values held by the tokens from input places in order to produce the output values.
- **Variable names on arcs:** each arc is decorated while a tuple of variable names, in order to make actions and preconditions of a transition able to handle values held by token.
- **Inhibitor and Test arcs:** inhibitor arcs are used as zero tests or threshold tests that allow the enabling of transitions if the linked places are empty, and test arcs are used for simple tests on the tokens held by a place, without removing them from the place while firing the transition.

b) Dynamic behaviour

- **Enabling of transitions:** the enabling rule evolves to account the precondition and the new kind of arc. The rule remains basically the same as for classic Petri nets (e.g. enough tokens in each input places), and then the precondition is evaluated using the set of possible input values (called substitutions).
- **Firing of transitions:** the firing is the same as for classic Petri nets, and the value of the produced token may be the result of actions inside the transition.

3) Cooperative Objects formalism (CO)

A Cooperative Object states how the object reacts to external stimuli according to its inner state. This behavior, called the Object Control Structure (ObCS) is described by means of OPN.

a) Structure

- A **software interface** which describes the set of public methods provided by the object. The syntax used to express this set is the Java syntax as it is precise enough for describing method signature and basic types.
- Several **communication protocols**:
  - A **unicast synchronous** communication (e.g. method calls): Cooperative Objects may communicate among each other using method calls, where these methods are the ones from the associated software interface. A binding mechanism provides a translation of one signature of one method into a set of special places in the Petri net itself, which corresponds to input or output or exception parameters of the method. The semantic of this way of communication is based on a rebuild mechanism, which describes the communication as a Petri net.
  - A **multicast asynchronous** communication (e.g. event communication): Cooperative Objects may provide a set of events to which other Cooperative Objects may listen. The CO formalism defines ways to add/remove listeners, to trigger events and catch events using event handlers (represented by a set of particular transitions, called synchronized transitions). The semantics are not yet fully described, but we are investigating among related works in the Petri net community such as using Signal nets [14].

b) Dynamic behaviour

- **Enabling of transitions:** the enabling rule is exactly the same as for OPN.
- **Firing of transitions:** the firing rule is exactly the same as for OPN, and the action inside the transitions may result into a method call of another CO or into the raise of an event.
- **Observability:** the execution of the Petri net itself is fully observable, based on the event mechanism presented below. This feature is based on the design pattern called Observer [10], e.g. it provides means for the notification of state changes (e.g. marking changes for places), transitions and event handlers' availability and firing.

4) Interactive Cooperative Objects formalism (ICO)

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

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 events availability and event handlers availability.
- Links between state in the CO changes and rendering.

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 in the following way.. The presentation part is viewed as 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 for the CO interface description.

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 (via the Observer+event mechanism) and calls the corresponding activation rendering method from the presentation part with values for its parameters that only come from the event handler.

The activation function is fully expressed through a mapping as a CO which provides it with its semantic.

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 (via the Observer+event mechanism) and call the corresponding rendering method from the presentation part with values for its parameters that only come from the event handler.

As for the Activation function, the Rendering function is fully expressed as a CO class.

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).

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [17]. 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 [19], space command and control ground systems [21], or interactive military [8] or civil cockpits [2]. The example of civil aircraft is used in the next section to illustrate the specification of embedded systems.

B. *Advantages*

The main advantages of ICOs for the formal description of behavioral specification of interactive cockpit applications are related to their use of Petri nets. Indeed, by the representation of tokens.

IV. ILLUSTRATION OF ICO ON THE EXAMPLE OF ARINC 661 SPECIFICATION

In ARINC 661, a user application is defined as a system that has a two way communication with the Cockpit Display System (CDS):

- 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.

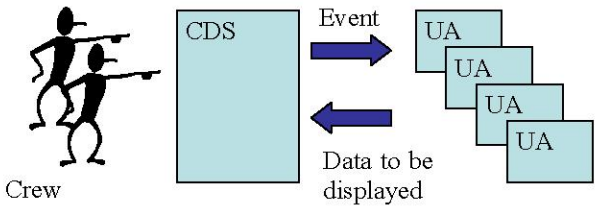

Figure 2. CDS / UA communication

As shown in Fig. 2 the CDS part 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.

Due to space constraints we only present in this paper an application featuring few widgets and supporting few tasks. However, the specification covers the entire application both at the widget and the User Application levels.

A. *Specification of a user application*

Modeling a user application using ICO is quite simple as ICO has already been used to model such kind of interactive applications. Indeed, UAs in the area of interactive cockpits correspond to classical WIMP interfaces.

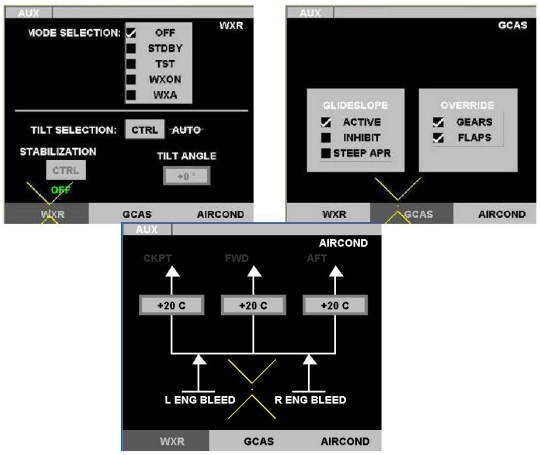

Figure 3. Snapshots of the 3 pages of the UA MPIA

The Multi Purpose Interactive Application (MPIA) is a User Application (UA) that aims at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND) between which a crew member is allowed to navigate using 3 buttons (as shown by Fig. 3). The WXR page is responsible for managing weather radar information; GCAS is responsible for the Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning.

The next sections present the 4 parts of the ICO specification of the page WXR extracted from the User Application MPIA.

1) *Presentation part*

The presentation part is made up of a set of widgets that are used for both rendering information and provides the user with means to interact with the interactive systems.

```
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);
}
```

Figure 4. Software interface of the page WXR from the user application MPIA

The layout of the presentation part (the upper left window in Fig. 3) is out of the scope of the ICO specification, but this presentation part is seen as a collection of rendering methods and ways to provide events as shown in Fig. 4.

2) *Behaviour*

As stated in the section describing the Cooperative Objects formalism, a model may is composed of a software interface and a behavior described using 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. 5 shows the entire behavior of page WXR which is made up of two non connected parts:

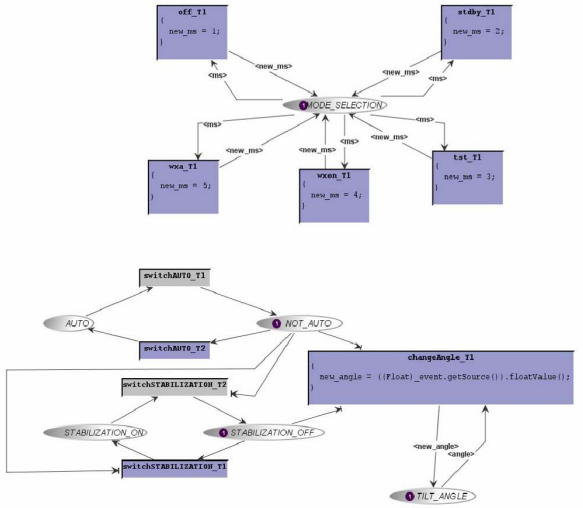

Figure 5. Behaviour of the page WXR

- The upper part aims at handling events from the 5 CheckButtons and the modification implied of the MODE_SELECTION that may be one of five possibilities (OFF, STDBY, TST, WXON, WXA). Value changes of tokens stored in place *Mode-Selection* are described in the transitions while variables on the incoming and outgoing arcs play the role of formal parameters of the transitions.
- The lower part concerns the handling of events from the 2 PicturePushButton and the EditBoxNumeric. Interacting with these buttons will change the state of the application.

3) *Activation function*

Fig. 6 shows an excerpt of the activation function for the WXR page.

User Events	Event handler	ActivationRendering
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

Figure 6. Activation Function of the page WXR

This table may be read line by line, as one line describes the three objects linked in the activation process, for instance the

user event *ask_off*, the event handler *off* from the behavior and the activation rendering method *setWXRModeSelectEnabled* from the presentation part. The signification of this line is:

- 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.
- When the button OFF of the presentation part is pressed, the presentation part raises the event called *asked_off*. This event is catch by the activation function which asked the behavior part to fire the event handler *off*.

4) Rendering Function

The modeling of the rendering function is shown in Fig. 7).

ObCSNode 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

Figure 7. Rendering Function of the page WXR

This table may be read line by line too, as one line describes the three objects linked in the rendering process, for instance the place *MODE_SELECTION*, the event linked to this place and in which we are interested *token_enter* and the rendering method *showModeSelection* from the presentation part. 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 about the new marking of the place.

B. *Specification of the CDS*

One of the goals of the work done on interactive cockpits compliant with ARINC Specification 661 was to define an architecture that clearly identifies each part of this architecture and their communication, as shown in Fig. 8. The aim of this architecture is also to clearly identify which components will be taken into account in the modeling process and which ones are taken into account in a different way (for instance rendering is done using SVG facilities).

Such architecture presents two main advantages:

1. Every component that has an inner behavior (server, widgets, User Applications (UA), and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modeled using the ICO formal description technique.
2. The rendering part is delegated to a dedicated language and tool (such as SVG).

As written in the description of ARINC 661, the need for precise specification occurs at both widget and UA levels.

At a widget level, ICO must be used to describe the inner behavior of the widgets and to describe the impact of state changes on their external presentation.

At a user application level, ICO must be used to describe the behavior of the application itself and the impact of state changes in term of widget parameters modification.

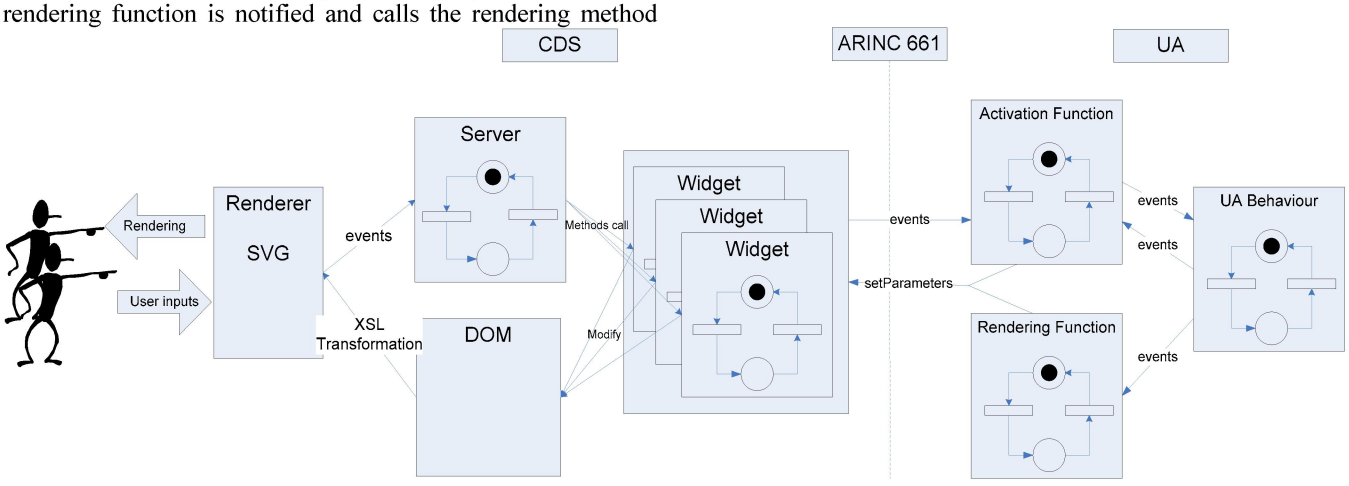

Figure 8. Detailed architecture to support ARINC 661 specification

We briefly present hereafter the definition of widgets in ARINC 661 and then show how ICO can be fruitfully used to formally represent their detailed behavior. Due to space constraints we do not present the Server part of which an overview is given in [4].

C. *Specification of a widget*

A widget is defined with an identifier (widget type, widget identifier and widget parent), states (informal description of the relationship between these states) and a six other description parts that provides information about parameters, data needed

for creation, events raised… such as the parameters table shown in TABLE I.

TABLE I. EXCERPT OF THE PUSHBUTTON PARAMETERS

Parameter	Change	Description
Commonly used parameters		
PosX	D	The X position of the widget reference position
Enable	DR	Ability of the widget to be activated
Specific parameters		
LabelString	DR	String of the PushButton
MaxStringLength	D	Maximum length of the label text

The main drawback of this description is the lack of specification of state changes and their impact on the presentation of the widget. In order to be able to build reliable and certifiable interactive software a precise and unambiguous specification is required. We exploit here the ICO formal description technique presented above. As stated earlier, the behavior of a widget is made up with a software interface, as shown in Fig. 9, and a high-level Petri net, as shown in Fig. 10.

```
public interface A661_PUSH_BUTTON {
        void setEnable(char A661_ENABLE);
        void setVisible(char A661_VISIBLE);
        void setStyleSet(short A661_STYLE_SET);
        void setLabelString(string A661_STRING);
};
```

Figure 9. Software interface for ARINC 661 PushButton

For the description of ARINC 661 widgets, this software interface defines the run-time modifiable parameters: one definition of a "set" method for one run-time modifiable parameter. For instance, the ARINC 661 PushButton provides a run-time modifiable parameter, called Enable of the type "char", as shown in Table 4. In its definition, this parameter is represented by the method "`void setEnable(char A661_ENABLE)`".

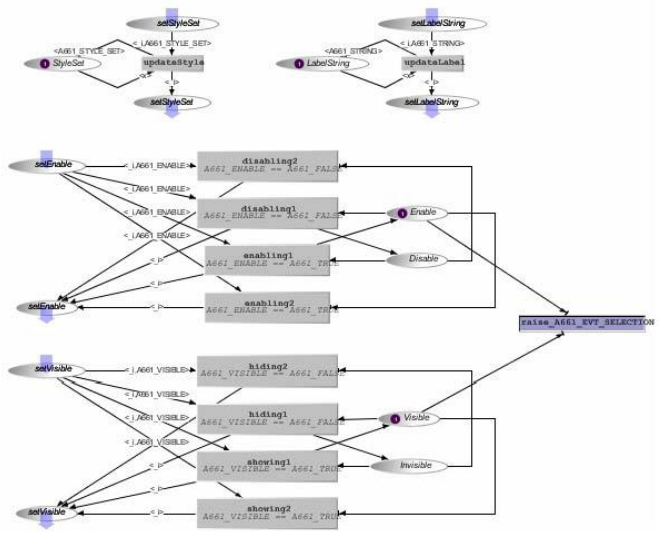

Figure 10. Behaviour of the ARINC 661 PushButton

In the High-level Petri net model the complete and unambiguous description of the widget is given. Widget parameters are held by tokens in places. Depending on the repartition and value of these tokens, special transitions (called synchronized transitions) may be fired, and when fired, these transitions raise an event (those described in the corresponding ARINC specification). Therefore, by specifying the widget behavior, we clearly define the conditions under which a widget raises an event.

As for the user application part, we do not describe here the "look and feel" of the widget, but show what must be represented. Indeed, the important element is what information has to be presented to the pilot and when to present it.

TABLE II. shows an excerpt of the rendering method of the PushButton. For instance, it describes that when a token enters the place *Enabled*, the PushButton must be shown as enabled.

TABLE II. EXCERPT OF PUSHBUTTON RENDERING FUNCTION

ObCS Item	Event	Rendering method
Place LabelString	Token <x> enters	Display <x>
Place Enabled	Token enters	Show as enabled
Place Enabled	Token exit	Show as disabled

D. What has not been presented through this example

Due to space constraints, we have not presented the part of the work linked to both ICO and the interactive cockpit. We use the next paragraphs to list some of these points.

1) ARINC 661 Specification

A big part of the interactive cockpits is the user interface server which 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 creation of widgets, graphical cursors of both the pilot and his co-pilot, the mouse and keyboard events and dispatching it to the corresponding widgets, … The corresponding ICO models have been done and presented in [4] and illustrates how to handle huge and complex models.

2) ICO formal description technique

The Activation functions and Rendering functions are fully describe using Petri nets, legitimates the use of the table notation as a readable way to express the connection between the dialog and the presentation parts.

As it does not bring more sense to the presentation of the notation, we did not present all the initialization mechanisms such as creation of models, initialization of the communication using events (registers for listening some events, …).

3) Execution of models

A well-known advantage of Petri nets is their executability. This is highly beneficial to our approach, since as soon as a behavioral specification is provided in term of ObCS, this specification can be executed to provide additional insights on the possible evolutions of the system. The environment used to support the edition, verification and execution of models is called PetShop, and more details may be found in [3].

The execution of the models of the user application MPIA leads to another interesting part of the work done about ARINC specification 661. The models of the user application MPIA can both be connected to the modeled CDS or to an

implemented CDS, using a special API, as it respects the ARINC 661 specification. As testing an implemented user application is still a problem that has to be solved, especially when the UA is connected to a real CDS, a model based approach may support testing at different levels:

1. Test a modeled user application on the modeled CDS.
2. Test the modeled user application on the CDS implemented by the manufacturer.
3. Code and test the user application on the implemented CDS.

The first step promotes a very iterative prototyping process where both the User Application and the CDS may be modified, as the second step allows user testing on the real interactive system (CDS), with classical prototyping facilities provided by the models expressed in ICO of the User Application.

V. Conclusion and Perspectives

This paper has presented the use of a formal description technique for describing interactive components in ARINC specification 661. One of the advantages of using the ICO formal description technique is that it provides additional benefits with respect to other notations such as Statecharts as proposed in [23]. Thanks to its Petri nets basis the ICO notations makes it possible to model behaviors featuring an infinite number of states (as states are modeled by a distribution of tokens in the places of the Petri nets).

Acknowledgements

This work was supported by the EU funded ResIST Network http://www.resist-noe.eu. The work was also partly funded by DPAC (Direction des Programmes de l'Aviation Civile) under contract #00.70.624.00.470.75.96.

References

[1] ARINC, ARINC 661 specification: Cockpit Display System Interfaces To User Systems, Prepared by AIRLINES ELECTRONIC ENGINEERING COMMITTEE, Published by AERONAUTICAL RADIO, INC, april 22, 2002.

[2] Barboni E., Navarre D., Palanque P. & Basnyat S.. Exploitation of Formal Specification Techniques for ARINC 661 Interactive Cockpit Applications. Proceedings of HCI aero conference, (HCI Aero 2006), Seatle, USA, Sept. 2006.

[3] Eric Barboni, David Navarre, Philippe Palanque & Didier Bazalgette. PetShop : A Model-Based Tool for the Formal Modelling and Simulation of Interactive Safety Critical Embedded Systems. Proceedings of HCI aero conference (Demonstration) (HCI Aero 2006), Seatle, USA, Sept. 2006.

[4] Eric Barboni, Stéphane Conversy, David Navarre & Philippe Palanque. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. Proceedings of the 13th conference on Design Specification and Verification of Interactive Systems (DSVIS 2006), Dublin, 2006, LNCS, Springer Verlag.

[5] Bastide, R., Palanque, P. A Petri Net Based Environment for the Design of Event-Driven Interfaces. 16th International Conference on Application and theory of Petri Nets (ATPN'95), Italy, 20-22 June 1995.

[6] Bastide, Rémi, Palanque, Philippe, Le, Duc-Hoa and Muñoz, Jaime. Integrating Rendering Specifications into a Formalism for the Design of Interactive Systems. in 5th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'98, Abingdon, U. K. Springer-Verlag (1998)

[7] Bastide R., Barboni E., Lacaze X., Navarre D., Palanque P., Schyn A. & Bazalgette D. Supporting INTUITION through formal specification of the User Interface for military aircraft cockpit. In procedings of HCI International 2005, Las Vegas, July 2005.

[8] Bastide, R., Palanque, P., Ousmane, S, Duc-Hoa Le., Nvarre, D.. Petri Net Based Behavioural Specification of CORBA Systems. International Conference on Application and Theory of Petri nets ATPN'99, Williamsburg (USA), LNCS Springer Verlag, 1999.

[9] Faerber R. Vogl T. & Hartley D. Advanced Graphical User Interface for Next Generation Flight Management Systems. In proceedings of HCI Aero 2000, pp. 107-112.

[10] Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software.Addison Welsey #63361, 1994.

[11] Gram, Christian; Cockton, Gilbert, Editors. Design principles for interactive software. Chapman et Hall ed.1995.

[12] Genrich, H. J. Predicate/Transitions Nets. High-Levels Petri Nets: Theory and Application . K. Jensen and G. Rozenberg (Eds.)Berlin: Springer Verlag (1991) pp. 3-43.

[13] ICAO, Guidance Material on CNS/ATM Operations in the Asia/Pacific Region, http://www.icao.org, International Civil Aviation Organization, DOC 4444 PANS/RAC, 1991.

[14] Juhás Gabriel, Lorenz Robert, Neumair Christian: Modelling and Control with Modules of Signal Nets. 585-625 in proceedings of 24th International Conference on Applications and Theory of Petri Nets, ICATPN'2003

[15] Lakos, C. Language for Object-Oriented Petri Nets. #91-1. Department of Computer Science, University of Tasmania, 1991.

[16] Marrenbach J. & Kraiss K-F. Advanced Flight Management System: A New Design and Evaluation Results. In proceedings of HCI Aero 2000, pp. 101-106.

[17] Navarre, David, Palanque, Philippe, Bastide, Rémi and Sy, Ousmane. Structuring Interactive Systems Specifications for Executability and Prototypability. 7th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000, Limerick, Ireland. Lecture Notes in Computer Science.

[18] Navarre, D. Contribution a l'ingénierie en Interaction Homme Machine - Une technique de description formelle et un environnement pour une modélisation et une exploitation synergiques des tâches et du systčme. PhD Thesis. Uniersity of Toulouse I. Defended July 2nd 2001. 2001.

[19] Navarre, David; Palanque, Philippe & Bastide, Rémi. Reconciling Safety and Usability Concerns through Formal Specification-based Development Process HCI-Aero'02 MIT, USA, 23-25 October, 2002.

[20] OMG. The Common Object Request Broker: Architecture and Specification. In CORBA IIOP 2.2.Framingham 1998.

[21] 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. Ninth International Conference on Space Operations, Rome, Italy, June 18-22, 2006.

[22] User Interface Management Systems, Eurographics Seminar, Seeheim, 1983. Gunther Ptaff, editor. Springer Verlag, 1983.

[23] Sherry L., Polson P., Feary M. & Palmer E. When Does the MCDU Interface Work Well? Lessons Learned for the Design of New Flightdeck User-Interface. In proceedings of HCI Aero 2002, AAAI Press, pp. 180-186.

[24] Shneidermann B. Direct manipulation: a step beyond programming languages. Computer 16 (Aug. 1983), 57-69.

[25] Sy, Ousmane, Bastide, Rémi, Palanque, Philippe, Le, Duc-Hoa and Navarre, David. PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems. 20th International Conference on Applications and Theory of Petri Nets, ICATPN'99.

[26] Valk, R. Petri Nets as Token Objects: an Introduction to Elementary Object Nets. 19th International Conference on Application and Theory of Petri Nets, ICATPN'98, Lissabon, Portugal, June 1998, Springer, 1998.