Model-based engineering of widgets, user applications and servers compliant with ARINC 661 specification

Eric Barboni, Stéphane Conversy, David Navarre, Philippe Palanque

Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification

Barboni Eric[1], Conversy Stéphane[1,2], Navarre David[1] & Palanque Philippe[1]

[1] LIIHS – IRIT, Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex 4
{barboni, conversy, navarre, palanque}@irit.fr
http://liihs.irit.fr/{barboni, navarre, palanque}

[2] ENAC – DTI/SDER – Ecole Nationale de l'Aviation Civile
7, avenue Edouard Belin, 31055 Toulouse.
conversy@enac.fr

Abstract. The purpose of ARINC 661 specification [1] is to define interfaces to a Cockpit Display System (CDS) used in any types of aircraft installations. ARINC 661 provides precise information for communication protocol between application (called User Applications) and user interface components (called widgets) as well as precise information about the widgets themselves. However, in ARINC 661, no information is given about the behaviour of these widgets and about the behaviour of an application made up of a set of such widgets. This paper presents the results of the application of a formal description technique to the various elements of ARINC 661 specification within an industrial project. This formal description technique called Interactive Cooperative Objects defines in a precise and non-ambiguous way all the elements of ARINC 661 specification. The application of the formal description techniques is shown on an interactive application called MPIA (Multi Purpose Interactive Application). Within this application, we present how ICO are used for describing interactive widgets, User Applications and User Interface servers (in charge of interaction techniques). The emphasis is put on the model-based management of the feel of the applications allowing rapid prototyping of the external presentation and the interaction techniques. Lastly, we present the CASE (Computer Aided Software Engineering) tool supporting the formal description technique and its new extensions in order to deal with large scale applications as the ones targeted at by ARINC 661 specification.

1 Introduction

Interactive applications embedded in cockpits are the current trend of evolution promoted by several aircraft manufacturer both in the field of civil and military systems [7, 10]. Embedding interactive application in civil and military cockpit 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 comes along with

several problems that have to be taken into account with appropriate precautions. 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 propose a formal description technique to be used as a complement to ARINC 661 for the specification, design, implementation and validation of interactive application.

The paper is structured as follows. Next section introduces ARINC 661 specification to define software interfaces for a Cockpit Display System. It presents informally the content of the specification but also its associated architecture that has to be followed in order to build ARINC-661-compliant interactive applications. Section 3 presents the ICO formalism, a formal description technique for the design of safety critical interactive applications. This description technique has already been applied in various domains including Air Traffic Control applications, multimodal military cockpits or multimodal satellite ground segments. Its applicability to cockpit display system and its compatibility with ARINC specification 661 is discussed and extensions that had to be added are also presented in section 4. Section 5 presents the use of the formal description technique on an interactive application called MPIA (Multi Purpose Interactive Application) currently available in some cockpits of regional aircrafts. Last section of the paper deals with conclusions and perspectives to this work.

2 ARINC 661 specification

This section presents, in an informal way, the basic principles of ARINC 661 specification. The purpose of this section is to provide a description of the underlying mechanisms of ARINC 661 specification and more precisely how its content influences the behaviour and the software architecture of interactive applications embedded in interactive cockpits.

2.1 Purpose and Scope

The purpose of ARINC 661 specification (ARINC 661, 2002) is to define interfaces to a Cockpit Display System (CDS) used in interactive cockpits that are now under deployment by several aircraft manufacturers including Airbus, Boeing and Dassault. The CDS provides graphical and interactive services to user applications (UA) within the flight deck environment. Basically, 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).

ARINC 661 dissociates, on one side, input and output devices (provided by avionics equipment manufacturers) and on the other side the user applications (designed by aircraft manufacturers). Consistency between these two parts is maintained through a communication protocol:

- 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.

In the field of interactive systems engineering, interactive software architectures such as Seeheim [14] or Arch [9] promote a separation of the interactive system in at least three components: presentation part (in charge of presenting information to and receiving input from the users), dialogue part (in charge of the behaviour of the system i.e. describing the available interface elements according to the current state of the application) and functional core (in charge of the non interactive functions of the system). The CDS part may be seen as the presentation part of the whole system, provided to crew members, and the set of UAs may be seen as the merge of both the dialogue and the functional core of this system.

2.2 User Interface Components in ARINC 661

The communication between the CDS and UAs is based on the identification of user interface components hereafter called widgets. ARINC 661 defines a set of 42 widgets that belong to 6 categories. Widgets may be any combination of "container", "graphical representation" of one or more data, "text string" representations, "interactive", dedicated to "map management" or may "dynamically move".

In ARINC 661, each widget is defined by:

- a set of states classified in four levels (visibility, inner state, ability, visual representation),
- a description in six parts (definition section, parameters table, creation structure table, event structure table, run-time modifiable parameter table, specific sections).

The main drawback of this description is the lack of description of the behaviour itself. Even if states are partially described, dynamic aspects such as state changes are informally described. As stated in ARINC 661 (section 1.0 introduction), the main paradigm is here based on this comment:

> "A UA should not have any direct access to the visual representations. Therefore, visual presentations do not have to be defined within the ARINC 661 interface protocol. Only the ARINC 661 parameter effects on graphical representation should be described in the ARINC 661 interface. The style guide defined by the OEM should describe the "look and feel" and thus, provide necessary information to UAs for their HMI interface design."

An additional textual description called SRS (for Software Requirement Specification), informally defines the look and feel of a CDS (Cockpit Display System). This SRS is designed by each manufacturer of airline electronic equipment (we worked with a draft document provided by Thales Avionics). This kind of document describes both the appearance and the detailed expected behaviour of each graphical or interactive component.

2.3 Overview of our contribution to ARINC 661

One of the goals of the work presented in this paper is to define an architecture that clearly identifies each part of this architecture and their communication, as shown on Fig 1. The aim of this architecture is also to clearly identify which components will be taken into account in the modelling process and which ones are taken into account in a different way by exploiting SVG facilities. The architecture has two main advantages:

1. 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.
2. The rendering part is delegated to a dedicated language and tool (such as SVG).

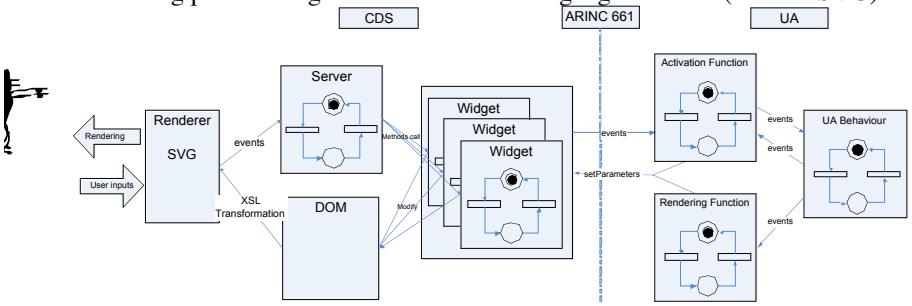

Fig 1. Detailed architecture to support ARINC 661 specification

The following section recalls the basics of ICO notation and presents a new extension that has been required in order to be able to address all the modelling challenges put forward by interactive cockpit applications compliant with ARINC 661 specification, and then present the connection to SVG. Lastly, a real case study illustrates this architecture and how modelling all the elements of ARINC 661 specification are addressed using ICOs formal description technique.

3 ICO modelling of ARINC 661 components

We use the ICO formalism to describe formally the behaviour of the ARIC components. This section first briefly recalls the main features of the ICO formalism. We encourage the interested reader to look at [13, 11] for a complete presentation of the formal description technique and the environment supporting it. The second part is dedicated to the extensions that had to be defined in order to address the specificities of interactive applications compliant with ARINC 661 specifications.

3.1 Overview of the ICO formalism

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [4, 11]. It uses concepts borrowed from the object-oriented approach to describe the structural or static aspects of systems, and uses high-level Petri nets [8] to describe their dynamic or behavioural aspects. ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by Petri nets. The formalism made up of both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO).

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). Time-out transitions are special transitions that do not belong to the categories above.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [12]. The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

3.2 ICO improvements

Two main issues have been raised while working with ARINC 661 specification that have not been encountered in previous work we have done in the field of interactive systems' specification and modeling.

- The first one is related to the management of rendering information in a more independent and structured way in order to be able to dissociate as much as possible the graphical appearance of interactive components from their behavior. This is one of the basics of interactive cockpit applications compliant with ARINC 661 specification as (as stated above) these two sides of the interactive cockpit applications are described in two different documents (communication protocol and abstract behavior in ARINC 661 specification while presentation and detailed behavior are described in the SRS (System Requirement Specifications)).
- The second one is related to the fact that ARINC 661 specification does not exploit current windows manager available in the operating system (as this is the case for Microsoft Windows applications for instance). On the opposite, the manufacturer in charge of developing the entire ARINC 661 architecture is also in charge of developing all the components in charge of the management of input devices, device drivers and to manage the

graphical structure of the interactive widgets. In order to handle those aspects we have defined a denotational semantics (in terms of High-level Petri nets) of both the rendering and the activation functions. Beforehand, these functions were only partly defined (relying on the underlying mechanisms provided by the window manager) and implemented using a particular java API thus making much more limited the verification aspects of theses aspects of the specification. Indeed, the work presented here addresses at the same level of formality, applications, widgets and user interface server (also called window manager). Besides, the connections and communications between these three parts are also formally described.

Next section presents in details the various mechanisms that have been defined in order to handle the low level management of input devices and focuses on one specific aspect called picking which correspond to the window manager activating of finding the interactive component that was the target of the user when an event has been produced. The case study in section 4 shows on a concrete example how those elements are combined for describing User Applications, Widgets and User Interface servers.

4 MPIA case study

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 2). WXR page is in charge managing weather radar information; GCAS is in charge of the Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning.

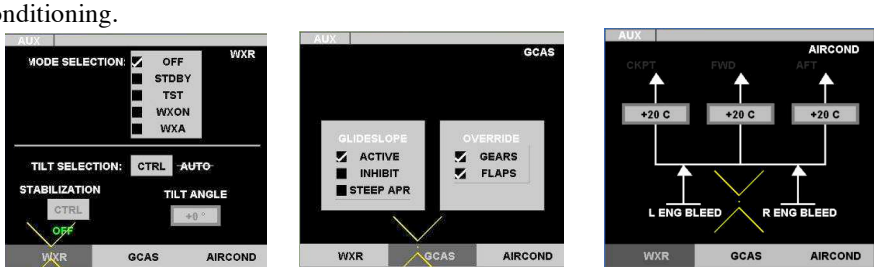

Fig 2. Snapshots of the 3 pages of the UA MPIA

In this section, we present the modelling of a simple widget and its link to SVG rendering, then we briefly present the classical modelling of a user application to show the extension made to ICOs, and finally we present parts of the server. The purpose is not here to present the whole specification which is made up of about 40 models, but only to present brief extracts to show all bricks of the modelling.

4.1 Modelling ARINC 661 interactive widgets

The whole modelling process of ARINC 661 interactive components using ICO is fully described in [12]. The additional feature consists in using the rendering process described above, based on replacing the classical code-based rendering methods with rendering methods that modify the SVG Document Object Model. Rendering is the process of transforming a logical description (conceptual model) of an interactive component to a graphical representation (perceptual model). In previous similar works, we specified rendering with Java code, using the Java2D API., However, describing graphics with an imperative language is not an easy task, especially when one tries to match a particular look. Furthermore, the java code for graphics is embedded into the model, which makes it hard to change for another look. This is even more difficult when several components share a common part of the graphical representation, for instance when components must have a similar style and when this style has to be changed.

To overcome these two problems, we changed for an architecture that uses declarative descriptions of the graphical part and that supports transformations from conceptual models to graphical representations. These two elements exploit XML-based languages from the W3C: the SVG language for graphical representation, and the XSLT language for transformation. SVG is an xml-based vector graphics format: it describes graphical primitives in terms of analytical shapes and transformations. XSLT is an xml-based format that describes how to transform an xml description (the source) to another xml description (the target). An XSLT description is called a "stylesheet". Due to space constraints this work is not presented in the next section as we focus on the behavioural aspects of models.

4.2 Modelling User Applications

Modelling 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,

As the detailed specification is not necessary to expose the modification of ICO, we only present an excerpt of the models that have been produced to build the MPIA application. This excerpt is the first page (WXR) of the application (left part of Fig 2).

4.2.1 Behaviour

Fig 3 shows the entire behaviour of page WXR which is made up of two non connected parts:

- The upper part aims at handling events from the 5 CheckButtons and the modification implied of the MODE_SELECTION that might be one of five possibilities (OFF, STDBY, TST, WXON, WXA). Value changes of token 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.

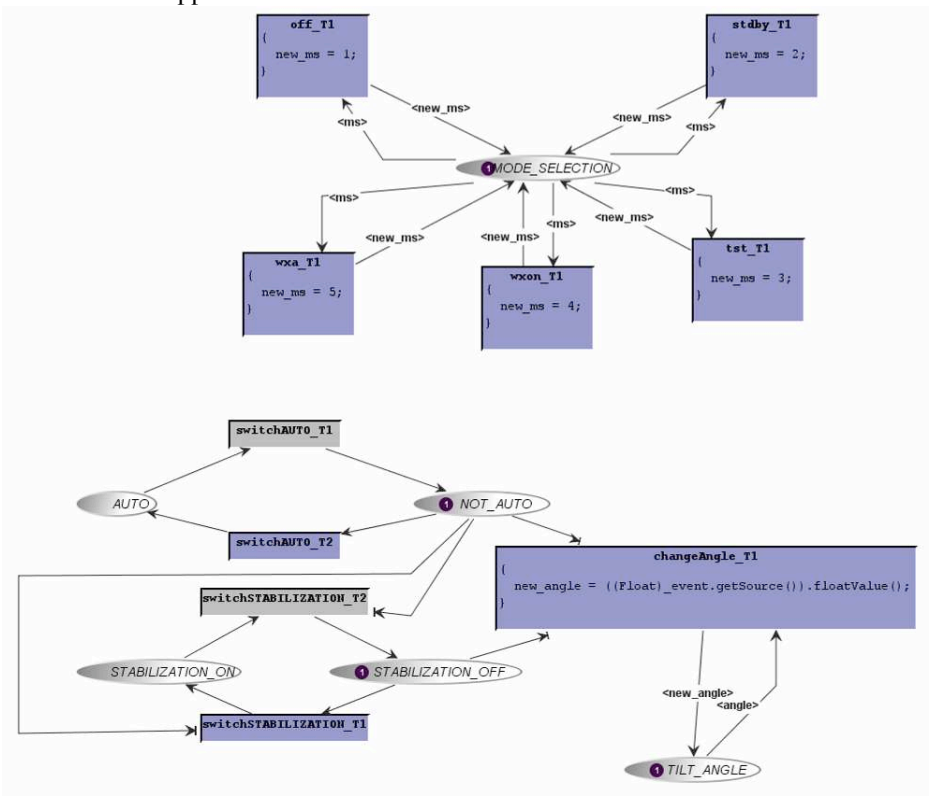

Fig 3. Behaviour of the page WXR

4.2.2 Activation function

Fig 4 shows an excerpt of the activation function for page WXR.

	Widget	Event	UserService	ActivationRendering
wxrOFFAdapter	off_CheckButton	A661_INNER_STATE_SELECT	off	setWXRModeSelectEnabled
wxrSTDBYAdapter	stdby_CheckButton	A661_INNER_STATE_SELECT	stdby	setWXRModeSelectEnabled
wxrTSTAdapter	tst_CheckButton	A661_INNER_STATE_SELECT	tst	setWXRModeSelectEnabled
wxrWXONAdapter	wxon_CheckButton	A661_INNER_STATE_SELECT	wxon	setWXRModeSelectEnabled
wxrWXAAdapter	wxa_CheckButton	A661_INNER_STATE_SELECT	wxa	setWXRModeSelectEnabled
autoAdapter	auto_PicturePushButton	A661_EVT_SELECTION	switchAUTO	setWXRTiltSelectionEnabled
stabAdapter	stab_PicturePushButton	A661_EVT_SELECTION	switchSTABILIZATION	setWXRTiltSelectionEnabled
tiltAngleAdapter	tiltAngle_EditBox	A661_STRING_CHANGE	changeAngle	setWXRTiltSelectionEnabled

Fig 4. Activation Function of the page WXR

From this textual description, we can derive the ICO model shown on Fig 5. The left part of this figure presents the full activation function, which is made up of as many sub Petri nets as there are lines in the textual activation function. The upper right hand side of the figure emphasises on of these sub Petri nets. It describes how the availability of the associated widget is modified according to some changes in the WXR behaviour. The lower right hand part of the Figure shows the general pattern associated to one line of the activation function: It describes the handling of the event raised par the corresponding widget, and how it is linked to an event handler in the WXR behaviour.

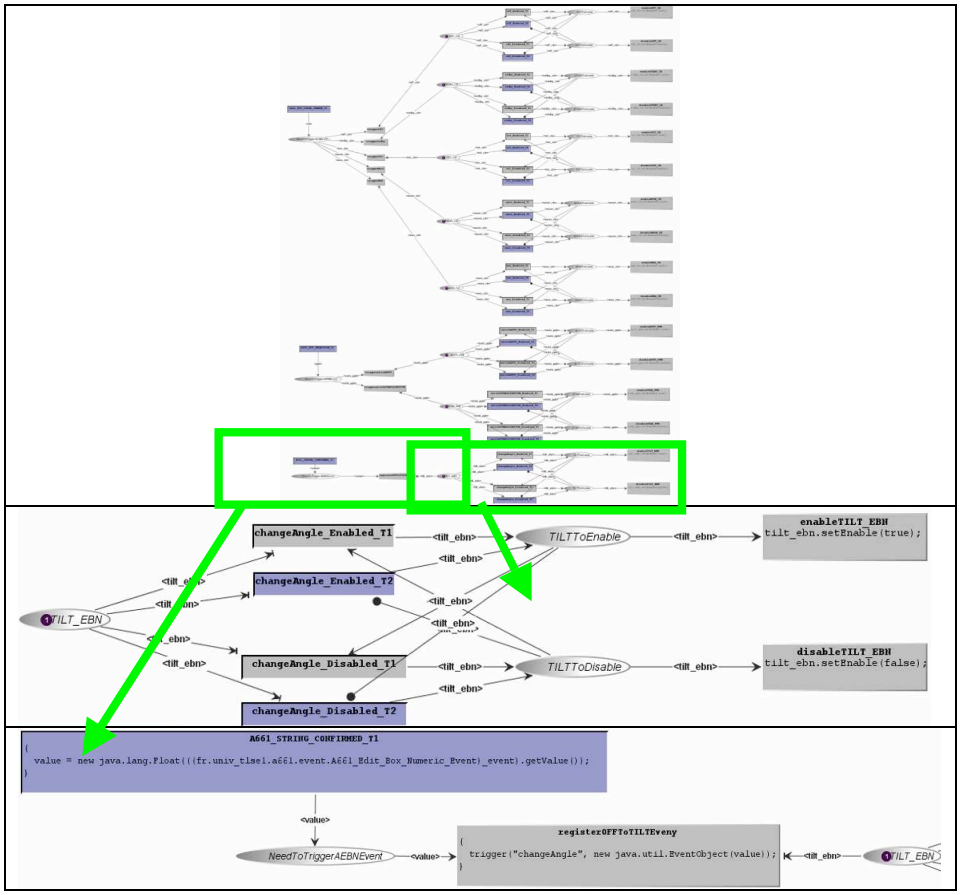

Fig 5. Activation Function of the page WXR expressed in Petri nets

The use of Petri nets to model the activation function is made possible thanks to the event communication available in the ICO formalism. As this kind of communication is out of the scope of this paper, we do not present the models responsible in the registration of events-handlers needed to allow the communication between behaviour, activation function and widgets. More information about this mechanism can be found in [2].

4.2.3 Rendering Function

The modelling of the rendering function (shown on Fig 6) into Petri nets (shown on Fig 7) works the same way as for the activation function, i.e. for each line in the rendering function, there is a pattern to express that in Petri nets. This is why we do not detail more the translation.

	ObCSNode name	ObCS event	Rendering method
modeSelectionAdapter	MODE_SELECTION	token_enter <int m>	showModeSelection(m)
tiltAngleAdapter	TILT_ANGLE	token_enter <float a>	showTiltAngle(a)
initAutoAdapter	AUTO	marking_reset	showAuto(true)
autoAdapter	AUTO	token_enter	showAuto(true)
notAutoAdapter	AUTO	token_remove	showAuto(false)
initStabAdapter	STABILIZATION_ON	marking_reset	showStab(true)
stabAdapter	STABILIZATION_ON	token_enter	showStab(true)
notStabAdapter	STABILIZATION_ON	token_remove	showStab(false)

Fig 6. Rendering Function of the page WXR

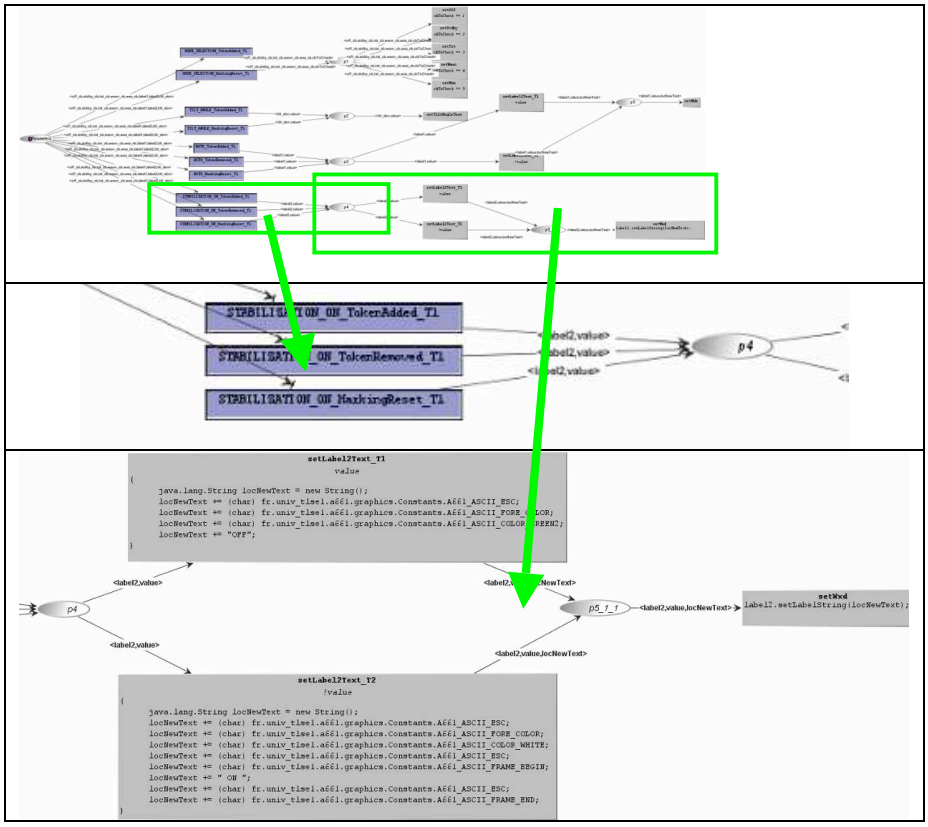

Fig 7. Rendering Function of the page WXR expressed in Petri nets

4.3 Modelling User Interface Server

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 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 many functionalities, 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. As the detailed model is out of the scope of this paper, Fig 8 only present an overview of the complete model.

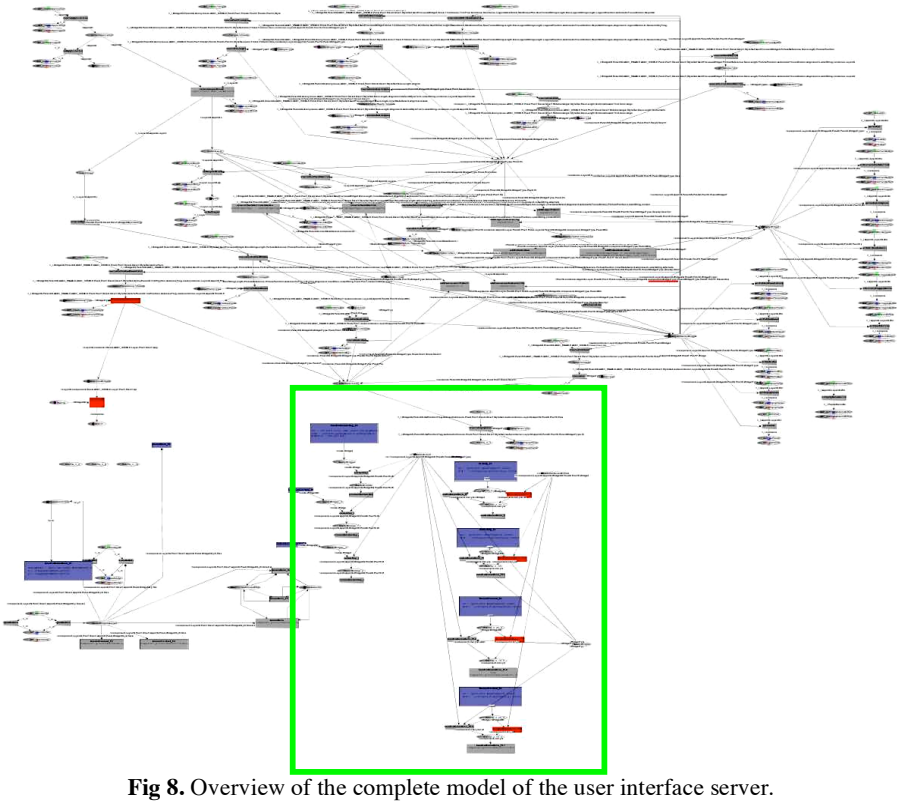

Fig 8. Overview of the complete model of the user interface server.

The rectangle at the bottom of Fig 8 represents the part of the model of the server in charge of the interaction technique and input devices management. The rest of the model corresponds to the management of the widgets.

4.4 Modelling the complete MPIA User Application

We do not present here the full model of the user application MPIA neither the one of the user interface server, but the formal description technique ICO has been used to model in a complete and non ambiguous way all the pages and the navigation between pages for such user application, and still produces low-sized and readable models. Modelling Activation functions and Rendering functions 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.

Another issue is that the models of the user application MPIA can both be connected to the modelled 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 modelled user application on the modelled CDS.
2. Test the modelled 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.

The MPIA application has been fully modelled and can be executed on the CDS modelled using the ICO formalism. However, it has also been connected on a CDS developed on an experimental test bench as shown in Fig. 9.

Fig. 9. The MPIA application modelled using ICO connected to experimental CDS at THALES

5 Conclusions and Perspectives

This paper has presented the use of a formal description technique for describing interactive components in ARINC specification 661. Beyond that, we have shown that this formal description technique is also adequate for interactive applications embedding such interactive components. 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 [15]. Thanks to its Petri nets basis the ICO notations makes it possible to model behaviours featuring an infinite number of states (as states are modelled by a distribution of tokens in the places of the Petri nets). Another advantage of ICOs is that they allow designers to use verification techniques at design time as this has been presented in [3]. These verification techniques are of great help for certification purposes.

We are currently developing techniques for providing support to certification processes by allowing verification of compatibility between the behavioural description of the interactive application and task model describing nominal or unexpected pilots behaviour. Support is also provided through the verification of interactive system safety and liveness properties such as the fact that whatever state the system is in there is always at least one interactive element available.

6 Acknowledgements

The work presented in the paper is partly funded by DPAC (Direction des Programmes de l'Aviation Civile) under contract #00.70.624.00.470.75.96. Special thanks are due to our colleagues at THALES P. Cazaux and S. Marchal.

7 References

1. 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. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004 Pennsylvania State University, USA.
3. Bastide R., David Navarre & Philippe Palanque. Tool Support for Interactive Prototyping of Safety Critical Interactive Applications. In Encyclopedia of HCI, C. Gaoui (Ed.). ISBN: 1-59140-562-9. Hard Cover. Publisher: Idea Group Reference Pub Date: July 2005. Pages: 650.
4. Bastide R., Ph. Palanque A Petri Net Based Environment for the Design of Event-Driven Interfaces. 16th International Conference on Application and theory of Petri Nets (ATPN'95), LNCS, Springer Verlag, Torino, Italy, 20-22 June 1995.

5. Beaudoux O., 2005. XML Active Transformation (eXAcT): Transforming Documents within Interactive Systems. Proceedings of the 2005 ACM Symposium on Document Engineering (DocEng 2005), ACM Press, pages 146-148.
6. Blanch R., Michel Beaudouin-Lafon, Stéphane Conversy, Yannick Jestin, Thomas Baudel and Yun Peng Zhao. INDIGO : une architecture pour la conception d'applications graphiques interactives distribuées. In Proceedings of IHM 2005, pages 139-146, Toulouse - France, September 2005.
7. 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.
8. Genrich H. J.. (1991). Predicate/Transition Nets, in K. Jensen and G. Rozenberg (Eds.), High-Level Petri Nets: Theory and Application. Springer Verlag, Berlin, pp. 3-43.
9. Gram C., Cockton G. (Editors). Design principles for interactive software. Chapman et Hall ed.1995.
10. Marrenbach J., Kraiss K-F. Advanced Flight Management System: A New Design and Evaluation Results. In proceedings of HCI Aero 2000, pp. 101-106.
11. Navarre D., Palanque, Philippe, Bastide, Rémi. A Tool-Supported Design Framework for Safety Critical Interactive Systems in Interacting with computers, Elsevier, Vol. 15/3, pp 309-328, 2003.
12. Navarre D., Philippe Palanque & Rémi Bastide. A Formal Description Technique for the Behavioural Description of Interactive Applications Compliant with ARINC 661 Specification. HCI-Aero'04 Toulouse, France, 29 September-1st October 2004.
13. Palanque P., R. Bastide. Petri nets with objects for specification, design and validation of user-driven interfaces. In proceedings of the third IFIP TC 13 conference on Human-Computer Interaction, Interact'90. Cambridge 27-31 August 1990 (UK).
14. Pfaff, G. E. (Hrsg.): User Interface Management Systems, Proceedings, Workshop on User Interface Management Systems, Seeheim,(1. - 3.11.1983); Springer Verlag 1983.
15. 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.
16. Souchon, N., Vanderdonckt, J., A Review of XML-Compliant User Interface Description Languages, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
17. UsiXML, http://www.usixml.org/?view=news.
18. Villard, L. and Layaïda, N. 2002. An incremental XSLT transformation processor for XML document manipulation. In Proceedings of the 11th international Conference on World Wide Web (Honolulu, Hawaii, USA, May 07 - 11, 2002). WWW '02. ACM Press, New York, NY, 474-485.