UsiXML Concrete Behaviour with a Formal Description Technique for Interactive Systems

Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, Marco Winckler

▶ **To cite this version:**

Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, Marco Winckler. UsiXML Concrete Behaviour with a Formal Description Technique for Interactive Systems. IFIP WG 2.7/13.4 Workshop on User Interface Description Languages (UIDL 2011), IFIP: International Federation for Information Processing, Sep 2011, Lisbonne, Portugal. pp.(electronic medium). hal-03651207

HAL Id: hal-03651207
https://hal.science/hal-03651207

Submitted on 27 Apr 2022

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

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

UsiXML Concrete Behaviour with a Formal Description Technique for Interactive Systems

Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, Marco Winckler
Institute of Research in Informatics of Toulouse, University of Toulouse
Interactive Critical Systems (ICS) team
118 route de Narbonne
31042 Toulouse Cedex 9, France
{barboni, martinie, navarre, palanque, winckler}@irit.fr

ABSTRACT
In the last years User Interface Description Languages (UIDL) such as UsiXML appeared as a suitable solution for developing interactive systems. So far, there have been several attempts for exploring the potential of UsiXML as a language for describing user interface components for multi-target platforms. In this paper we are concerned by the behavioural aspect of interactive system built using UsiXML. In order to implement reliable and efficient applications, we propose to employ a formal description technique called ICO (Interactive Cooperative Objects) that have been developed to cope with complex behaviours of interactive systems including event-based and multimodal interaction. Our approach offers a bridge between UsiXML descriptions of the user interfaces components and a robust technique for describing behaviour using ICO modelling. Beyond that, this paper highlights how it is possible to take advantage from the two approaches to make possible to provide a model-based approach for prototyping interactive systems. The approach is fully illustrated by a case study using the ARINC 661 specification for User Interface components embedded into interactive aircraft cockpits.

Keywords
Interactive systems, Behavioural modelling, User Interface Description Languages (UIDLs), UsiXML, ARINC 661.

1. INTRODUCTION

In the last years User Interface Description Languages (UIDL) appeared as a suitable solution for developing interactive systems [7][8][18]. In this scenario UsiXML [10] appears as an emergent standard for describing interactive system, in particular those sought to be deployed in different platforms [21]. It is widely agreed that a UIDL must cover three different aspects of the User Interface (UI): to describe the static structure of the user interfaces (i.e. presentation part which ultimately includes the description of user interface elements, e.g. widgets, and their composition), to describe the dynamic behaviour (i.e. the dialog part, describing the dynamic relationships between components including event, actions, and behavioural constraints) and to define the presentation attributes (i.e. look & feel properties for rendering the UI elements). Among the models involved in User Interface (UI) development, dynamic behaviour is one of the most misunderstood and one of the most difficult to exploit [6][23]. Dialog models play a major role on UI design by capturing the dynamic aspects of the user interaction with the system which includes the specification of: relationship between presentation units (e.g. transitions between windows) as well as between UI elements (e.g. activate/deactivate buttons), events chain (i.e. including fusion/fission of events when multimodal interaction is involved) and integration with the functional core which requires mapping of events to actions according to predefined constraints enabling/disabling actions at runtime. These problems related to the description of behavioural aspects of interactive systems have been discussed in detail in [15]. Among the techniques presented, it is worth of mention the Interactive Cooperative Objects (ICO) formalism which is a formal description technique designed to the specification, modelling and implementation of interactive systems. ICO has been demonstrated efficient for describing several techniques including 3D, multimodal interaction techniques and dynamic reconfiguration of interactive systems [16]. ICO models are executable and fully supported by the CASE tool PetShop [4] which has been shown effective for prototyping interactive techniques [14].

In this paper we propose a model-driven approach to integrate behaviour described using ICO models and user interface components described with UsiXML. By using ICO models is possible to run the Petshop environment to control the execution of the application. This approach has already been demonstrated efficient to model the behaviour of user interface components based on the standard ARINC 661 for interactive aircraft cockpits [1][2][3][13]. In section 2 we present an overview of behavioural aspects in UsiXML and how these issues have been treated by the research community. Section 3 introduces the standard ARINC 611 and how user interface components described by this standard can be implemented using UsiXML. Section 4 introduces the case study. Section 5 is devoted to the specification of the behaviour of user interface components. In section 6 we present a proposal for extending the concrete behavioural description within UsiXML. Finally, section 7 presents conclusions and future work.

2. USIXML AND BEHAVIOURAL DESCRIPTIONS

UsiXML (USer Interface eXtensible Markup Language) is defined in a set of XML schemas where each schema corresponds to one of the models containing attributes and relationships in the scope of the language [10]. UsiXML schemas are used to describe at a high level of abstraction the constituting elements of the UI of an application including: widgets, controls, containers, modalities, interaction techniques, etc. The UsiXML language is structured according to the four levels of abstractions as proposed by the framework Cameleon [7], as follows: *task models, abstract user interfaces (AUI), concrete user interface (CUI)* and *final user interface (FUI)*. Several tools [12] exist for editing specification using UsiXML at different levels of abstraction. Notwithstanding, developers can start using UsiXML schemas at the abstraction level that better suits their purposes.

As far as the behaviour is a concern, there are some dedicated schemas in UsiXML. At the task level, behaviour is covered by task models featuring operators in a similar way as it is done by CTT [11]. At the AUI and CUI levels several schemas allows to describe basic elements of the dialog behaviour including *events, triggers, conditions, and source* and *target* components. These elements can be refined at the FUI level to reach final constructs implemented by the target platform.

So far there is limited support for UsiXML schemas related to behavioural aspect of interactive systems beyond the task model level. Some extensions have been proposed to describe high level dialog behaviours such as those implemented by transitions between windows [22] and between states of workflow-based applications [9]. However, all these extensions are more or less related to task models. The description of fine-grained behaviour in UsiXML is awkward as the behavioural aspect and the user interface composition are interleaved in a single description. So that, the description of events, triggers and actions is scattered along the components of the user interface with makes extremely difficult to visualize the behaviour of the current state of the application being modelled. Another conceptual issue with dialog modelling with UsiXML is related to the different levels of abstraction; whilst abstract containers can be easily mapped to windows, it is not so easy to envisage abstract behaviour and how to refine them into more concrete actions on the user interface.

A few works [17][23] have addressed the behaviour aspect of interactive system described with UsiXML. Schaefer, Bleul, and Mueller (2006) [17], propose an extension of UsiXML by the means of a dedicated language called *Dialog and Interface Specification Language* (DISL). The main contribution of that work is to propose clear separation between presentation, user interface composition and dialog parts of the interface. Winckler et al (2008) [23] suggest there is no need of new dialog language as UsiXML can be coupled with existing dialog modelling techniques such as StateWebCharts (SWC) [24] to deal with the behaviour of interactive systems. Those authors propose a set of mappings that allows SWC specification to be used as running engine for the behaviour of UsiXML specifications. Notwithstanding, the work was limited to navigation between web pages in Web-based user interfaces.

3. ARINC 661 SPECIFICATION AND USIXML

Even if the main topic of this contribution is to make a bridge between a description of the user interface using UsiXML and an external behavioural description, we firstly propose an overview of a similar work done on an aircraft standard for interactive application. Making a parallel with this previous work, we then highlight the basic bricks making possible to enhance UsiXML with a behavioural description. As illustrated in the next paragraphs, services offered by the ARINC 661 widgets and the definition of User Application (UA) are very close to UsiXML Concrete User Interface model.

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 [1][2]. In ARINC 661, a user application is defined as a system that has two-way communication with the CDS:

- Transmission of data to the CDS, possibly 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 [5], the CDS part (in Figure 1) 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.

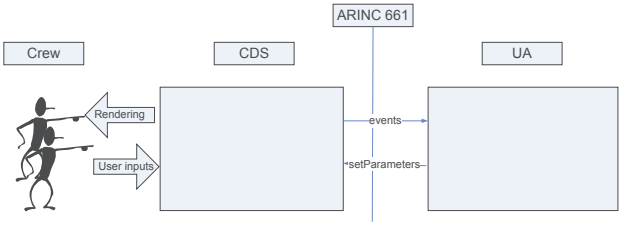

Figure 1. Abstract architecture and communication protocol between Cockpit Display System and a User Application.

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 Figure 2).

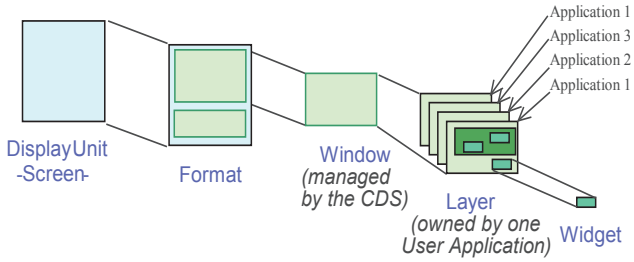

Figure 2. ARINC 661 Specification windowing architecture.

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 corresponds to classical interactors on Microsoft Windows system such as command buttons, radio buttons, check buttons, among others).

In ARINC 661, a widget is defined with an identifier (widget type, widget identifier and widget parent), states (informal description of the relationship between these states) and some other descriptions:

- **A definition section** provides general information on the widget such as the categories it belongs to, a functional description of its behaviour and restrictions (if any) with respect to ARINC 661 principles.
- **A parameter table** provides the list of the widget parameters (position, size, availability…).
- **A creation structure table** presents the parameters required for the instantiation of the widget (kind, restrictions…).
- **An event structure table** presents the event notification structure. It describes the parameters that may be held by the events.
- **A run-time modifiable parameter table** presents the sets of parameters that may be changed at run-time.

For instance, a *PushButton* is defined as followed (only a subpart of the entire description is provided hereafter):

Categories:

Graphical representation, Interactive, Text string.

Description:

A PushButton widget is a momentary switched button, which enables a crew member to launch an action. A PushButton has only one inner state, so there is no need for an inner state parameter.

Restriction:

None.

PushButton event structure:

Event structure	Size(bits)	Value/Description
EventId	16	A661_EVT_SELECTION

PushButton Runtime Modifiable Parameters:

Parameter	Type	Size	Parameter Ident	Type of structure
Enable	Uchar	8	A661_ENABLE	…
Visible	Uchar	8	A661_VISIBLE	…
…				

…

In ARINC 661, a UA communicates with the CDS asking for modification of widgets parameters and receiving events from them. On the CDS side, the set of widgets is created and their layout is related to the use of the User Application Definition File (UADF). The content of this file, as well as the description of widgets is really close to the UsiXML model for a Concrete User Interface (even if it is not expressed using an XML-based format).

4. INFORMAL DESCRIPTION OF THE CASE STUDY

In order to illustrate our approach, we briefly introduce the MPIA application (which stands for Multi-Purpose Application) that we employ as case study (see Figure 3).

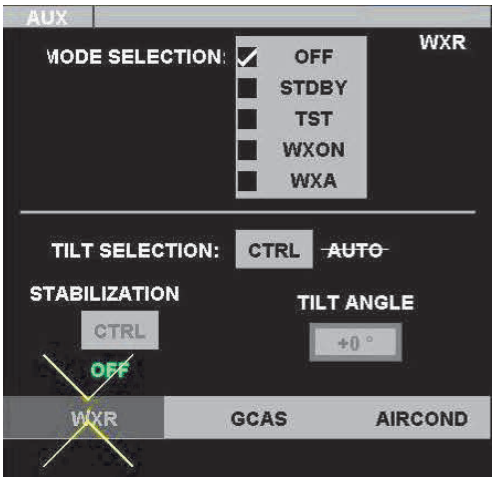

Figure 3. WXA User Interface of the MPIA application.

The MPIA is an application embedded into aircraft cockpits (see Figure 4) and it aimed for handling several flight parameters. It is made up of three pages (called WXR, GCAS and AIRCOND) between which a crew member is allowed to navigate. 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. Due to space reasons, we only focus on the WXR page. For the same reasons, we only on the ARNC 611 component *PushButton* that is used to build the buttons *WXR*, *GCAS* and *AIRCOND* as shown in the bottom-side of Figure 3.

Figure 4. The MPIA application in aircraft cockpit.

5. BEHAVIOURAL DESCRIPTION OF ARINC 661 WITH ICO

Such as UsiXML CUI model, ARINC 661 does not provide an explicit description of both the application and widgets behaviour. Previous works based on the ICO formal description technique [15] have been done in order to enhance ARINC 661 specification. In [13] we provide the basis for mapping parts of the ARINC 661 Specification into ICO constructs used to describe the behaviour of both widgets and UA. In [3] we present architecture to explicit rendering concerns based on SVG [19]. In [16] we improve the previous architecture to support both multimodal interaction and reconfiguration of input and output devices. In this section, we present an overview of this work.

5.1 The ICO formalism

The Interactive Cooperative Objects (ICO) formalism is based on concepts borrowed from the object-oriented approach (i.e. dynamic instantiation, classification, encapsulation, inheritance, and client/server relationships) to describe the structural or static aspects of systems, and uses high-level Petri nets to describe their dynamics or behavioural aspects. In the ICO formalism, an object is an entity featuring five components: a cooperative object (CO), an available function, a presentation part and two functions (the activation function and the rendering function) that correspond to the link between the cooperative object and the presentation part.

The **Cooperative Object** (CO) models the behaviour of an ICO. It states (by means of a high-level Petri net) how the object reacts to external stimuli according to its inner state. Figure 5 shows the concepts of the Cooperative Object models including: *places* (i.e. used as variables for tracking the system state), *transitions* (i.e. elements processing changes in the system state) and *arcs* (i.e. connecting places and transitions in a graph). *Arcs* can indicate input/output for tokens circulating in the graph; notice that an input arc (i.e. *InputArc*) can be extended to feature preconditions such as testing the availability of tokens in a place (*i.e. TestArc*) or preventing the movement of token accordingly to special conditions (i.e. *InhibitorArc*). The variables associated to an arc are expressed by the concept *EString*. Tokens can hold values of any class in the system. The types of tokens that can circulate in a given place are denoted through the relationship with the concept *EClass*.

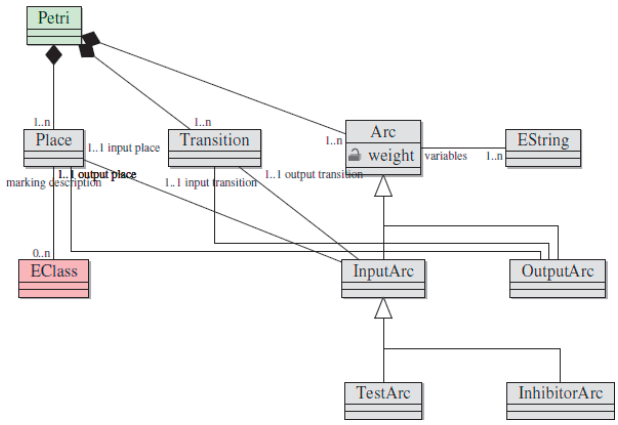

Figure 5. The Cooperative Objects meta-model.

The **presentation part** describes the external appearance of the ICOs. It is a set of widgets embedded into a set of windows. Each widget can be used for interacting with the interactive system (user interaction → system) and/or as a way to display information about the internal state of the object (system → user interaction).

The **activation function** (user inputs: user interaction → system) links users' actions on the presentation part (for instance, a click using a mouse on a button) to event services.

The **rendering function** (system outputs: system → user interaction) maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes through functions calls.

Additionally, an **availability function** is provided to link a service to its corresponding transitions in the ICO, i.e., a service offered by an object will only be available if one of its related transitions in the Petri net is available.

5.2 Architecture

The architecture presented in Figure 6 proposes a structured view on the findings from of a project dealing with formal description techniques for interactive applications compliant with the ARINC 661 specification.

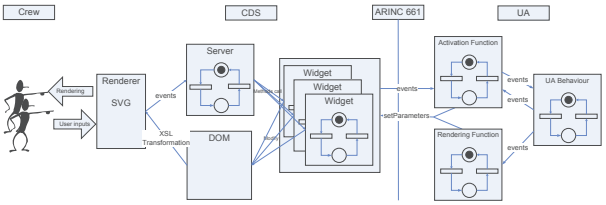

Figure 6. Detailed architecture compliant with ARINC 661 specification.

The ICOs notation is exploited to model the behaviour of all the components of an interactive application compliant with ARINC 661 specification. This includes each interactive component (i.e. widgets), the user application (UA) 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 Figure 6 are:

- 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,
- The rendering part is delegated to a dedicated language and tool (such as SVG, *Scalable Vector Graphics* [19]), 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. In this architecture the basic principle is to associate a document object model (DOM) to the set of widgets and to produce a SVG document using an XSLT transformation [26].

5.3 Overview of the formal description using ICO

As illustrated by the above architecture, ICO is used to model several parts of the entire interactive system. In this section, we present the modelling of a simple widget and its link to the SVG rendering, then we briefly present the classical modelling of a user application, and finally we present parts of the server. The purpose here is to present a brief extracts to show all bricks of the modelling.

Modelling ARINC 661 widgets

For each widget in ARINC 661 specification document, we model:

- Its behaviour using a Petri net.
- Its states (by the distribution of tokens in the places of the Petri net).
- The transition between the states.
- The rendering and activation function (which links the behaviour to the presentation part).

Modelling a widget follows the following process:

- Extract from ARINC 661 specification document the list of all the parameters
- Extract from ARINC 661 specification document the list of all the events it raises
- Build a software interface that exposes its run-time modifiable parameters, by providing an accessor for each parameter (i.e. a setXXX method for each XXX run-time modifiable parameter)
- Edit the Petri net model for which a skeleton has been generated from the previous information.

By applying this process, we modelled 12 widgets (from classical buttons, to complex containers such as a *Tabbed_Panel_Group*). Hereafter we present the modelling of a widget called *Picture_Push_Button* as an example. A *Picture_Push_Button* is a widget that is made up of 5 run-time modifiable parameters (*Enable*, *Visible*, *StyleSet*, *LabelString* and *PictureReference*) and raises 1 event (*A661_EVT_SELECTION*).

The upper side of the Figure 7 presents a zoom on the behaviour of this widget that handles the modification of the two parameters *Visible* and *Enable*. The bottom part of Figure 7 shows the connections of this widget and model describing the whole behaviour of the WXR application.

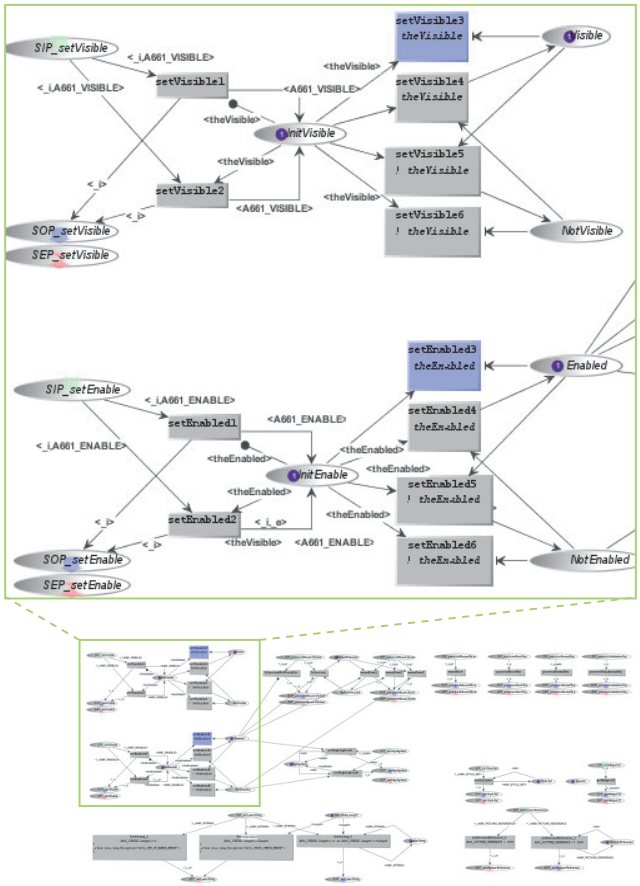

Figure 7. Behaviour model of the *PicturePushButton*.

Figure 8 presents the rendering function associated to the widget *Picture_Push_Button*. The third column presents the DOM attribute modified when the inner state of the button changes (e.g. when the state of the Petri net changes). An XSLT transformation is then used to produce the SVG document that renders the widget.

ObCSNode	ObCS event	Modified DOM attribute
Visible	*token_enter*	Visible = true
Visible	*token_remove*	Visible = false
Enabled	*token_enter*	Enabled = true
Enabled	*token_remove*	Enabled = false
…		

Figure 8. Rendering function of the *PicturePushButton*

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-based user interfaces[1].

Figure 9 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, allowing changing the tilt angle of the weather radar.

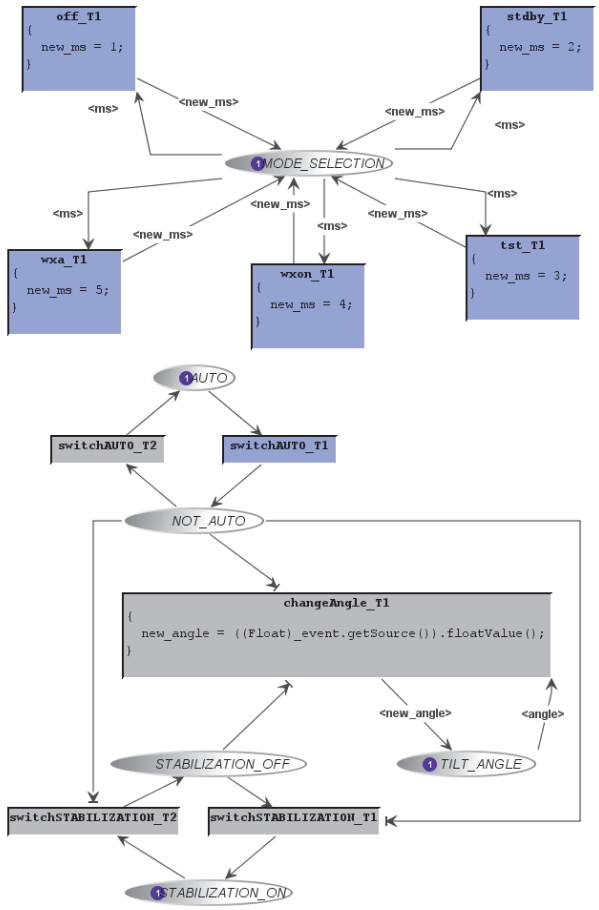

Figure 9. Behaviour of the page WXR

[1] WIMP stands for Window, Icon, Menu, Pointing device.

Figure 10 shows an excerpt of the activation function for page WXR, which describes the link between events availability and triggering and the behaviour of the application. For instance, the first line represents the link between the event A661_EVT_SELECTION produced by the button *auto_PicturePushButton* and the event handler switch from the behavioural model of WXR (see Figure 9). If the event handler is available, the corresponding event producer (the button) should be enabled.

Widget	Event	Event Handler
auto_PicturePushButton	*A661_EVT_SELECTION*	*switchAUTO*
stab_PicturePushButton	*A661_EVT_SELECTION*	*switchSTABILIZATION*
tiltAngle_EditBox	*A661_STRING_CHANGE*	*changeAngle*
…		

Figure 10. Activation Function of the page WXR

From this textual description, we can derive the ICO model as presented in [3]. 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.

Figure 11 shows an excerpt of the rendering function, which describes how state changes within the WXR behaviour lead to rendering changes. For instance, when a token (*<float a>*) enters (i.e. *token_enter*) the place *TILT_ANGLE,* it calls the rendering method *showTiltAngle(a)* which displays the angle value into a textbox.

ObCSNode name	ObCS event	Rendering method
MODE_SELECTION	*token_enter <int m>*	*showModeSelection(m)*
TILT_ANGLE	*token_enter <float a>*	*showTiltAngle(a)*
…		

Figure 11. Rendering Function of the page WXR

The modelling of the rendering function into Petri nets 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 (the interested reader may find more details in [3]).

Modelling User Interface Server

An important part of the above architecture is the user interface server that 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 much functionality, the complete model of such a server is complex and difficult to manipulate without an appropriate tool, and cannot be illustrated with a figure.

In previous works [16], this server has been improved to support reconfiguration policies for both input and output devices and it has been enhanced too to support multiple mice interaction.

6. A PROPOSAL FOR CONCRETE BEHAVIOURAL DESCRIPTION WITHIN USIXML

Beyond the obvious link that exists between the domain model of UsiXML and the behavioural description of an ICO, the work presented in the previous sections shows that there are common concerns between UsiXML CUI model and ARINC 661 specification (such as description of high level widgets and user interface, independent from implementation), and it shows that it is possible to enhance such descriptions with behavioural aspects.

With respect to the UsiXML architecture, the work done with ARINC 661 may be divided into two distinct parts, making possible to ease the design path from the concrete user interface to the final user interface.

6.1 An architecture making the bridge between ICO and UsiXML

As stated when discussing the architecture of Figure 6, it is possible to clearly separate behavioural aspects from rendering aspects. Figure 12 presents a first proposal for making UsiXML and ICO cooperate.

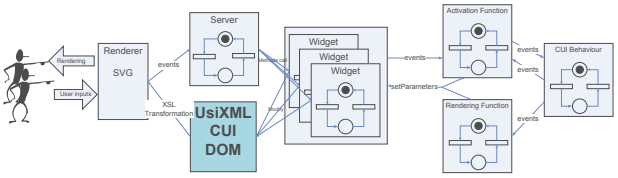

Figure 12. Detailed architecture compliant with UsiXML CUI.

As with ARINC 661, the main idea is to explicitly introduce behavioural models and make a clear link with the graphical representation. A successful integration should then lead to a UsiXML-based prototyping approach, inheriting from the prototyping capability of ICO.

6.2 Introducing behaviour at CUI level

Mapping state changes described using ICO description technique with UsiXML model attributes can be done easily. We illustrate the principle of introducing behavioural aspects at the CUI level with the example of the WXR application. These illustrations provide the key features allowing integration of ICO and UsiXML.

Figure 13 introduces a subpart of the CUI model of the WXR application, showing only a classical text box and a button:

- The *inputText* element *txt_tiltAngle* aims at containing a number representing a tilt angle. In order to include such as information into the description of the user interface built using UsiXML we propose the inclusion of an attribute "*text*" that does not exist in the current version of UsiXML. Thus attribute "*text*" is used to host the corresponding rendering function as shown by Figure 15.
- When clicked, the button *btn_switchAUTO* produces an event "*switchAUTO*". Both the availability of this event and its occurrence are related to the behaviour of the application (as stated by the next paragraphs).

```
<cuiModel id="WXR-cui_1" name="WXR-cui">

<window id="window_component_0"
name="window_component_0" width="456"
height="416">

<inputText id="txt_tiltAngle"
name="txt_tiltAngle" isVisible="true"
isEnabled="true" textColor="#000000"
maxLength="50" numberOfColumns="15"
isEditable="true" text=""/>

<button id="btn_switchAUTO" name="btn_switchAUTO"
isVisible="true" isEnabled="true"
textColor="#000000">

<behavior>

<event id="switchAUTO" eventType="action"
eventContext=""/>

</behavior>

</button>

…

</window>

</cuiModel>
```

Figure 13. Part of the CUI model of the WXR application

Making the link between the behaviour of the application expressed using ICO (as illustrated by Figure 9) is quite easy as there can be a direct mapping of the event produced by the button ("*switchAUTO*") and the available event handler of the behaviour of WXR ("*switchAUTO*"), as shown by Figure 14.

Widget	Event	Event Handler
btn_switchAUTO	*switchAUTO*	*switchAUTO*
…		

Figure 14. Activation Function of the page WXR

When the event handler is enabled, the attribute enabled of the button is thus set to *"true"*, *"false"* otherwise.

Describing the rendering of the application is linked to attribute modification of the CUI DOM such as described by Figure 15.

ObCSNode name	ObCS event	CUI attribute
TILT_ANGLE	*token_enter <float a>*	*"text" of txt_tiltAngle*
…		

Figure 15. Rendering Function of the page WXR

When the token enters the place *TILT_ANGLE*, the attribute "text" of the *inputText* element of the CUI is modified with the value hold by the token.

6.3 An executable CUI as a prototype for FUI

Thanks to the possibility of executing Petri nets, ICO allows prototyping when connected to the graphical representation of an application [14]. For instance, the MPIA application (from which WXR is extracted) 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 Figure 4.

Providing a graphical representation of the CUI makes possible to build a prototype based our approach. Associating ICO and the CUI model has been discussed in the previous section, but it is possible too, in a similar way, to do this association at widget level, while proposing a way to render a CUI model based on a previous work integrating SVG [3].

Such a work should then allow the prototyping of the final UI (FUI) based on the bridge between ICO and a CUI model, shortening the design path to the FUI.

Rendering based on SVG

As stated in the previous section, any state change of the application is rendered via the modification of the CUI DOM, based on the mapping described by both the rendering and activation function. Figure 16 illustrates the run-time architecture that supports FUI prototyping based on the association of UsiXML and ICO.

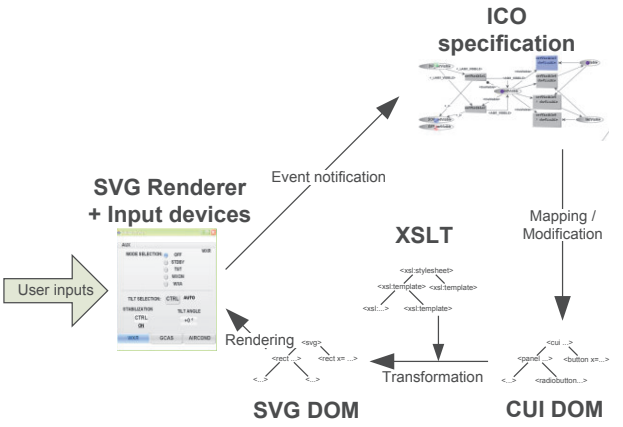

Figure 16. The run-time architecture

To provide a rendering to each CUI element, we propose the use of declarative descriptions of the graphical part that support transformations from conceptual models to graphical representations. The approach exploits both the SVG language [19] for graphical representation, and the XSLT language for transformation (called a "*stylesheet*").

In order to write a stylesheet, one has to design the rendering of a particular widget, using Illustrator for example. When ready, the textual description of the widget is included in the stylesheet.

In our case, the source is the CUI DOM, built at start-up time, together with the instantiation of the ICOs components. Before running the application, the system must compile the stylesheet to an XSLT transformer. While running the application, every time the state of a CUI DOM variable changes, it is transformed into a DOM SVG tree, which in turn is passed to the SVG renderer and displayed.

Introduction of behaviour for widgets

To go further with a precise prototyping of the FUI, it is necessary to describe each widget, including its behaviour (such as already done with ARINC 661). As illustrated by Figure 7 and Figure 8, it is possible to describe the fine grain behaviour of a widget and the link of its inner state changes with rendering.

In its current state, UsiXML, via the CUI model, describes widgets as a type and a set of attributes (a button is defined by an id, a name…), making it abstract enough to be independent from the targeted platform for the FUI. But when considering prototyping, it may be interesting to provide a finer description of the kind of widget that is expected, and a less coarse grain description of the widgets attributes (for instance, it is possible to introduce rendering for any inner state of a button: armed, pressed…). Another interesting point when dealing with widget is the introduction of new widgets that may request a precise description of how it should work on the targeted platforms.

One possible way to allow such description within UsiXML could be to enhance the current platform model of the context model with a precise widget description. Even if no effort has already been put on it, this way is an important part of our future works.

7. DISCUSSION AND OUTLOOK

Most of the recent work on UsiXML have been focused on mapping UsiXML schemas between several levels of abstraction [11][20] or proving automatic user interface generation of components to multi-target devices [12][21]. Indeed, very few works have focused on the behavioural aspect of interactive systems modelled with UsiXML.

This paper has presented a bridge between an already existing formal description technique for behavioural aspects of interactive systems and an approach for describing the presentation part of such system. Beyond

that, it highlights how it is possible to take advantage from the two approaches to make possible to provide a model-based approach for prototyping interactive systems.

Such as highlighted by the Arch architecture, this approach allows a clear separation between graphical aspects, behavioural aspects and functional aspects. It allows too a clear separation with tasks such as with the work done in [4]. Such a separation is necessary as, depending on the functional part of the interactive system, constraints independent from task concerns can appear. In the example used in this paper, the value of the tilt angle must meet the system requirements and the dialogue is thus specially designed to support this constraint. If the functional part changes, the dialog part must be modified, but not the user's tasks.

It is noteworthy that the use of ICO models to describe the behaviour of user interfaces allows overcoming of some of the limitations of other UIDL languages such as SCXML [25] XUL [27] such as the easier management of infinite states, the encapsulation of variables as objects of any kind and dynamic instantiation of objects. Moreover, properties of UI descriptions can be formally assessed using the underlying Petri Net formalism.

Three ways of improvement for this work could be:

- As presented in the previous section, a possible extension of our work is to introduce a notation or to enhance the current context model of UsiXML with a precise widget description, including its behaviour, making possible to build prototypes of the FUI.
- As presented with classical widgets, such an approach can be used to precisely describe new interactive components.
- A link from task models and abstract UI to concrete UI could be done based on the work we have already done about putting into correspondence task models and system model [4]

To make this work more "concrete" a particular effort has to be performed to integrate already existing tool support or to point out new developments. These issues are currently being addressed by our team at the IRIT (*Institute of Research in Informatics of Toulouse*) and the CNES (*Centre National d'Etudes Spatiales*) in a recently started Research & Technology project called ALDABRA.

ACKNOWLEDGEMENTS

This work is supported by the Research & Technology Project (RT) ALDABRA (IRIT-CNES).

REFERENCES

1. ARINC 661, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. ARINC Specification 661. (2002).
2. ARINC 661-2, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. ARINC Specification 661-2; (2005).
3. Barboni, E., Conversy, S., Navarre, D., Palanque, P. 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), LNCS, Springer Verlag.
4. Barboni, E., Ladry, J-F, Navarre, D., Palanque, P., Winckler, M. Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models. In Proc. Of the ACM SIGCHI conference Engineering Interactive Computing Systems (EICS 2010), Berlin, Germany, June 19-23, 2010, ACM SIGCHI, p. 143-152.
5. Bass L. et al. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. SIGCHI Bulletin, vol. 24(1):32–37, 1992.
6. Book, M., Gruhn, V.: Fine-Grained Specification and Control of Data Flows in Web-based User Interfaces. Journal of Web Engineering (JWE) Vol. 8, No. 1. Rinton Press, Paramus, NJ, USA 2009, pp. 48-70.
7. Calvary, G., Coutaz J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A. Unifying Reference Framework for Multi-Target User Interfaces, Interacting With Computers, Vol. 15/3, pp 289-308, 2003.
8. Guerrero-Garcia, J., Gonzalez-Calleros, J. M., Vanderdonckt, J., Munoz-Arteaga, J. 2009. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In Proceedings of the 2009 Latin American Web Congress (LA-WEB '09). IEEE Computer Society, Washington, DC, USA, 36-43. DOI=10.1109/LA-WEB.2009.40
9. Guerrero-García, J., Vanderdonckt, J., González-Calleros, J. M. FlowiXML: a step towards designing workflow management systems. In: Int. J. Web Eng. Technol., Vol. 4, Nr. 2 (2008), p. 163-182.
10. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Víctor López Jaquero, UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004), Springer LNCS 3425. pp. 200-220.
11. Montero, M., López-Jaquero, V., Vanderdonckt, Gonzalez, P. Lozano, Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML, Proc. of DSV-IS'2005 (Newcastle upon Tyne, 13-15 July 2005), S.W. Gilroy, M.D. Harrison (eds.), Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, 2005, pp. 161-172.
12. Michotte, B., Vanderdonckt, J., GrafiXML, A Multi-Target User Interface Builder based on UsiXML, Proc.

of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008 (Gosier, 16-21 March 2008), IEEE Computer Society Press, Los Alamitos, 2008, pp. 15-22.

13. 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-1st October 2004
14. Navarre, D., Palanque, P.; Bastide, R., and Sy, O. A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. 12th IEEE, International Workshop on Rapid System Prototyping; Monterey (USA). IEEE; 2001.
15. Navarre, D., Palanque, P., Ladry, J.F., Barboni, E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability, in Journal ACM Transactions on Computer-Human Interaction (TOCHI), Volume 16 Issue 4, November 2009, ACM New York, NY, USA
16. Navarre, D., Palanque, P., Ladry, J.F., Basnyat, S. An Architecture and a Formal Description Technique for User Interaction Reconfiguration of Safety Critical Interactive Systems. The XVth International Workshop on the Design, Verification and Specification of Interactive Systems (DSVIS 2008). Kingston, Ontario, Canada. July 16-18 2008.
17. Schaefer, R., Bleul, S., Mueller, W. 2006. Dialog modeling for multiple devices and multiple interaction modalities. In Proceedings of the 5th international conference on Task models and diagrams for users interface design (TAMODIA'06), Karin Coninx, Kris Luyten, and Kevin A. Schneider (Eds.). Springer-Verlag, Berlin, Heidelberg, 39-53.
18. Shaer, O., Green, M., Jacob, R.J.K, and Luyten, K., User Interface Description Languages for Next Generation User Interfaces. In Proc. of Extended Abstracts of CHI'08, ACM Press, New York (2008), pp. 3949-3952.
19. SVG W3C 2003: Scalable Vector Graphics (SVG) 1.1 Specification http://www.w3.org/TR/SVG11/
20. Tran, V., Vanderdonckt, J., Kolp, M., Wautelet, Y., Using Task And Data Models For User Interface Declarative Generation, Proc. of 12th International Conference on Enterprise Information Systems ICEIS'2010 (Funchal, 8-10 June 2010), J. Filipe, J. Cordeiro (Eds.), Vol. 5, SciTePress, 2010, pp. 155-160.
21. Trindade, F. M., Pimenta, M. S. RenderXML - A Multi-platform Software Development Tool. TAMODIA 2007, Springer LNCS 4849, p. 293-298.
22. Vanderdonckt, J., Limbourg, Q., Florins, M. Deriving the Navigational Structure of a User Interface, Proc. of Interact' 2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 455–462.
23. Winckler, M., Trindade, F.M., Stanciulescu, A., Vanderdonckt, J., Cascading Dialog Modeling with UsiXML, Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008 (Kingston, July 16-18, 2008), Lecture Notes in Computer Sciences, Vol. 5136, Springer, Berlin, 2008, pp. 121-135.
24. Winckler, M.; Palanque, P. StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications. International Workshop on Design, Specification and Verification of Interactive Systems - DSVIS'2003, Funchal, Portugal, June 2003.
25. World Wide Web Consortium. State Chart XML (SCXML): State Machine Notation for Control Abstraction. Working Draft 26 April 2011 Available at: http://www.w3.org/TR/2011/WD-scxml-20110426/
26. XSL Transformations (XSLT). Version 1.0. W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt
27. XUL (XML User Interface Language). Available at: http://www.mozilla.org/projects/xul/ (August 10, 2011).