Engineering mixed-criticality interactive applications

Camille Fayollas, Célia Martinie, David Navarre, Philippe Palanque

Engineering Mixed-Criticality Interactive Applications

Camille Fayollas, Célia Martinie, David Navarre, Philippe Palanque
ICS-IRIT, University of Toulouse 3
Toulouse, France
{fayollas, martinie, palanque, navarre}@irit.fr

ABSTRACT
In the field of critical systems, safety standards such as DO-178C define Development Assurance Levels (DALs) for software systems (or sub-systems). The higher the consequence of a failure the higher DAL is required by certification authorities. Developing a system at a DAL A requires the use of formal description techniques and is thus expensive. For lower DALs, standard software development is accepted. While operating such systems, reaching a given goal might require operators to perform tasks using sub-systems of different DALs. Operations thus take place via mixed-criticality systems developed using several different techniques. In order to guarantee the effectiveness of the developed systems, it is necessary to ensure the compatibility of the operators' tasks and the system (whatever technique has been used for its development). While DAL identification is outside the scope of the paper, this article presents a task-model based approach for addressing multiple DALs for mixed-criticality interactive software. That approach proposes a systematic process for engineering mixed-criticality interactive applications. This process is supported by a software modeling and development environment integrating both formal description techniques and standard software programming techniques. The process and the development environment are illustrated with a case study of a mixed-criticality interactive cockpit application.

Author Keywords
Interactive systems, task models, formal models, co-execution, Development Assurance Levels, critical systems.

ACM Classification Keywords
D.2.2 [Software] Design Tools and Techniques - Computer-aided software engineering (CASE).

INTRODUCTION
In the field of critical systems, safety standards such as DO-178C or IEC 61508 define Development Assurance Levels for software systems (or for functions of software systems). These levels are based on the analysis of consequences or effect of a malfunction. For instance, if a function failure has high consequences such as multiple fatalities, it is called catastrophic and certification authorities will require that the system manufacturer will provide a Development Assurance Level A (DO-178C standard for aeronautics [11]). If consequences are lower, the required level will decrease. Developing a system of a DAL A is extremely resource consuming and expensive and, as far as software is concerned, the use of formal description techniques is required [7]. In lower DALs, such expensive approaches are not required and for reaching levels such as DAL D rigorous software engineering approaches are sufficient.

One of the key issues in the development of critical systems is to identify the required level of DAL for each function which is usually performed by safety analysts. Once this is performed system designers have to apply development approaches compatible with the identified DALs. In the area of safety critical interactive systems, there is a lack of methods for designing systems with functions of different Development Assurance Levels. One of the key aspects of operating such systems is that operator tasks are orthogonal to Development Assurance Levels. This means that reaching a given goal might require operators to perform tasks using functions of different DALs. Ensuring the compatibility between the description of tasks to be performed by operators and the systems on which those tasks are performed is necessary in order to assess the effectiveness of a system. Beyond that, such compatibility provides additional benefits such as construction of a training program based on task descriptions, assessment of task complexity, and impact of systems evolutions on workload (among others).

In this article, we present a task-model based approach for addressing multiple DALs for the interactive software. Such software is qualified as mixed-criticality software [27]. The approach supposes that DALs have been previously defined by safety analysts and proposes a process for the development of mixed-criticality interactive applications as well as a software modeling and development environment supporting that process. It integrates both formal description techniques and standard software programming techniques. The process and the development environment are illustrated with a case study of a mixed-criticality interactive cockpit application.

The remainder of the paper is structured as follows. The next section presents an overview of the Development Assurance Levels for interactive systems, in order to highlight the main principles of this classification, exemplify it, and provide insights on the way it is used. The third section presents a process for addressing multiple Development Assurance Levels during the design and development of mixed-criticality interactive applications. The fourth section

EICS'16, June 21-24, 2016, Brussels, Belgium

DOI: http://dx.doi.org/10.1145/2933242.2933258

presents the main elements of the architecture of the CASE tool supporting the proposed process. The fifth section illustrates the process and the CASE tool with an excerpt from a case study of an aircraft interactive cockpit application.

DEVELOPMENT ASSURANCE LEVELS FOR INTERACTIVE SYSTEMS

In the aviation industry, the criticality level of a system is classified by means of five Development Assurance Levels (DAL) introduced in the DO-178C standard [11] for software components and in the DO-254 standard [12] for electronic hardware components. These levels are directly linked to failure condition categories defined by certification authorities such as EASA (European Aviation Safety Agency[1]) or FAA (Federal Aviation Administration[2]). Table 1 presents the five Development Assurance Levels associated with their failure condition category and its description (summarized from the EASA CS-25 standard [10]). As presented in the first row of Table 1, a failure having catastrophic consequences (failure condition column) must not occur more often than once per 10^9 hours of functioning (failure rate column). Such software must be developed following DAL A processes and methods as defined in DO-178C [11].

Development Assurance Level	Failure condition categories	Description of the failure conditions	Failure rate (failures/hour)
A	Catastrophic	Failure conditions that may cause a crash	Extremely improbable 10-9 +fail safe
B	Hazardous	Failure has a large negative impact or performance, Or reduces the ability of crew to operate the plane	Extremely remote 10-7
C	Major	Failure is significant, but has lesser impact than hazardous	Remote 10-5
D	Minor	Failure is noticeable, but has lesser impact than Major	Probable 10-3
E	No safety effect	No impact on dependability	Any range

Table 1. System Development Assurance Level for civil aircrafts

According to this standard the first two rows of Table 1 (colored in grey) correspond to so-called **critical** systems while systems in the lower rows are called **non critical**.

Requirements and costs for DALs

The last column of Table 1 presents the requirements of the different DALs in terms of acceptable rates of occurrence of failures as defined by certification authorities. At catastrophic level (DAL A in first row), a *Fail Safe* requirement is added to components in order to ensure that *no single failure can have catastrophic consequences on the aircraft*.

Table 2 presents the DALs with respect to what has to be done at development time and who is in charge of assessing what has been done. What has to be done is called Objectives (see column 3 of Table 2) and if the objective has to be assessed by an external body, it is listed in the column four ("with independence"). For instance, in DO-178C, for DAL A (row one of Table 2) 71 objectives have been identified and 33 of them have to be assessed by external bodies.

Development Assurance Level	Failure condition categories	Objectives	With independence
A	Catastrophic	71	33
B	Hazardous	69	21
C	Major	62	8
D	Minor	26	5
E	No Safety Effect	0	0

Table 2. Development Assurance Levels and their associated number of objectives to satisfied (from DO-178C [11])

Figure 1 presents an example of those objectives that are listed in the DO-178C. That objective refers to the need for ensuring that high-level requirements have been taken into account in the design. Such an objective is required with assessment by an external body for DAL A and DAL B software (see black dots in column "Applicability by Software Level"). The white dot for DAL C software means that such an objective does not require external body assessment. DO-178C only identifies what has to be done while annexes to that standard identify how thing have to be done. For instance, DO-333 annex [7] (page 101) states that high level requirements should be expressed in temporal logic while low-level requirements should be expressed by state-based description techniques. Compliance between these two representations has to be done using model-checking techniques.

	Objective		Activity	Applicability by Software Level			
	Description	Ref	Ref	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2.a	6.3.2	●	●	○	

Figure 1. Example of objectives from DO-178 C page 81

The main difference between critical and non-critical software is that a lot of objectives (e.g. verification activities) have to be satisfied with independence, thus leading to very expensive software development. Identifying accurately the correct DAL is thus very important in order to invest adequately development resources (not overspend on low DAL systems and not underspend on DAL A software).

[1] http://easa.europa.eu/

[2] http://www.faa.gov/

Example of interactive systems with different DALs

In a cockpit such as the one of Airbus A380, three types of interactive systems co-exist: physical hardware panels (composed of physical buttons, knobs and dials), numeric displays and interactive numeric user interfaces (the crew can interact with them using a keyboard and a track ball unit).

In current aircraft, all the critical interactive systems are of the first two types only:

i) hardware panels such as the Flight Control Unit: a physical panel enabling the command and control of the auto-pilot parameters and which is categorized DAL A;

ii) numeric displays such as the Primary Flight Display (PFD) which is categorized DAL B and which is associated with a monitoring software (categorized DAL A), in charge of verifying correct functioning of the display but only for some parameters identified as critical. It is interesting to note that this system (even though not interactive) involves mixed-criticality software.

All interactive software in cockpits is categorized as DAL C and thus can only be used for the command and control of non-critical systems (e.g. the Flight Warning System in charge of presenting alarms and their corresponding procedures to the pilot).

Operations on mixed-criticality systems

As explained previously, the CS-25 standard [10] is the standard for certification specification of the EASA for large aeroplanes. This document identifies requirements dedicated to the cockpit in the 1302 section called "*Installed systems and equipment for use of the flight crew*". Following this standard, the cockpit must allow the crew to safely perform all their tasks and avoid error prone behaviors.

However, during operations, some of the pilot tasks may involve several aircraft systems with different DALs. For instance, after perceiving an alarm on the ECAM display the pilot might decide to decrease flight level. In that case, he will successively use information displayed by the Flight Warning System (DAL C) and trigger commands using the Flight Control Unit (DAL A). In order to comply with CS 25 1302 it is thus important to develop methods and tools capable of ensuring that goals can be reached and tasks can be performed on systems of various DALs.

DEALING WITH LEVELS OF CRITICALITY IN INTERACTIVE SYSTEM (RELATED WORK)

Many techniques exist to design and develop interactive systems. In this section, they have been classified in two main groups:

- **Formal methods in HCI**: this group encompasses techniques and approaches that provide support for verification of user interfaces and interaction techniques. The techniques belonging to this group provide support for proving the correctness and robustness of a design and are therefore very eligible to be used for the development of critical parts of an interactive system.
- **Non formal user interface management systems**: this group encompasses all user interface management systems that are not focused on ensuring the correctness and reliability of an interactive system.

Formal methods in HCI

Several contributions in the field of formal methods in HCI provide support for dealing with criticality in interactive applications. Paterno and Santoro [26] proposed an approach based on formal model-checking (with CADP[3] toolset) of LOTOS [16] specifications of dialogue between the user and the system. Oliveira et al [25] presented how models following a PAC architecture can be used to verify ergonomics properties. The following set of cited approaches are based on the formal verification of state charts. Campos and Harrison [5] proposed an approach based on SMV [21] model-checking of Interactor specifications. Kamel and Ait Ameur [18] also propose an approach to verify properties for multimodal interactions with the SMV model checker. Combéfis et al. [9] propose to translate models into Java programs which can then be verified using JavaPathFinder[4] model checker. All of these approaches can target a wide range of properties verification on models [15] and are very suitable to describe and verify interactive applications with a low number of states.

Other approaches based on the executability of formal models provide augmented support for describing the conceptual model of the system and for analysis and verification at earlier stage in the process [14]. ICO-Petshop [24] and APEX-CPN Tools [28] frameworks are examples of these kind of approaches. ICO (Interactive Cooperative Object) [24] is a formal description technique based on high-level Petri nets which enables the specification of interactions between the user and the system as well as the system's behavior. This notation is associated with a CASE tool which provides editing facilities as well as a runtime environment to execute the user interface with its underlying ICO behavioral models (this formal notation and associated tool and properties verification capabilities are detailed in the third section). APEX-CPN Tools is a framework based on colored Petri nets [17], which provides support for rapid prototyping of ubiquitous environments and a predefined set of algorithms for properties verification. All of these contributions target the application of formal methods to the whole interactive system, but do not provide explicit support for designing and developing mixed-criticality interactive applications.

[3] http://cadp.inria.fr/

[4] http://babelfish.arc.nasa.gov/trac/jpf/

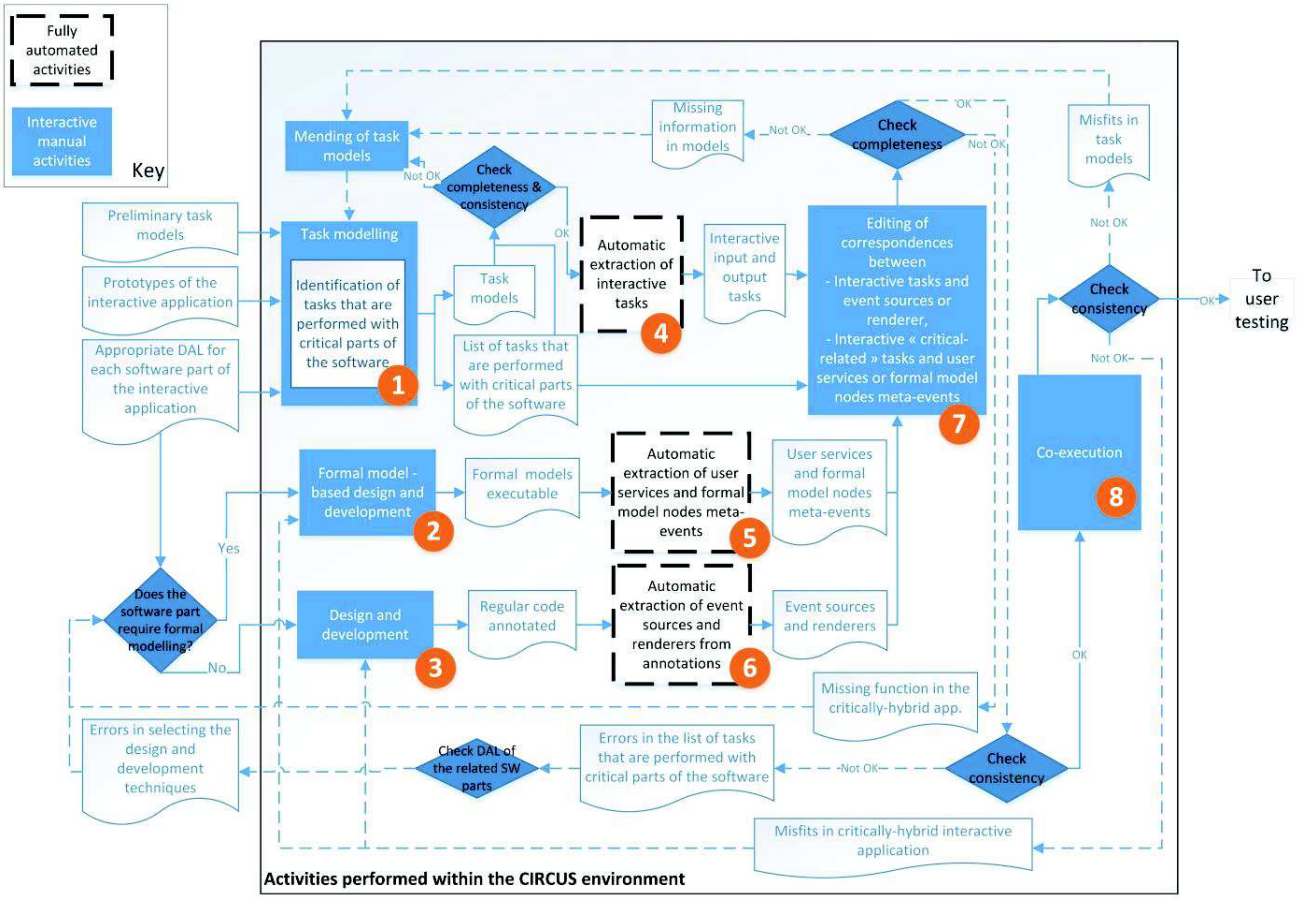

Figure 2. A process for addressing multiple Development Assurance Levels in mixed-criticality interactive applications

Non formal user interface management systems

Frameworks and tool suites dedicated to the design and development of user interfaces have been available for several decades [23]. A large set of these frameworks aim at providing support for faster development of graphical interfaces thanks to the introduction of graphical toolkits [4]. But they can also target specific types of user interfaces such as distributed user interfaces [20] and/or specific contexts of user interaction such as multimodal user interactions [6].

These frameworks and tool suites neither explicitly deal with the integration of formal and non-formal techniques nor target the development of mixed-criticality interactive applications. In the field of software engineering, approaches that integrate the use of formal and informal modeling languages have been proposed such as the model driven approach presented in [19]. In the field of dependable computing, targeted systems embed critical components and non-critical components, and a set of techniques take into account the integration of both types of components at design and development time such the one proposed in [27]. But these two last types of solution are dedicated to non-interactive software.

A PROCESS FOR ADDRESSING MULTIPLE DEVELOPMENT ASSURANCE LEVELS IN MIXED-CRITICALITY INTERACTIVE APPLICATIONS

The proposed process provides support for ensuring completeness and consistency between a mixed-criticality interactive application and users' tasks. As depicted in Figure 2, preliminary task models and appropriate DALs are taken for each software part of the interactive application. Three main steps can be carried out in parallel (discs 1, 2 and 3 in Figure 2)

- Task modeling refining the models in order to identify the tasks that are performed with critical parts of the software. Several task models are produced (using structuring mechanisms to handle large sets of users' activities [13]) and they are reworked until they are complete and consistent with each other.
- Formal model-based design and development of the critical parts of the interactive application.
- Design and development of the software components that do not require the application of formal methods.

Three types of artefacts are produced from these three steps: task models, executable formal models and annotated code.

CIRCUS (Computer-aided-design of Interactive, Resilient, Critical and Usable Systems), merges system modeling, software programming and task modeling tools into a single framework. Three types of artefacts can then be produced with CIRCUS tool. While all the steps can be performed with CIRCUS, they can be interactive manual activities (plain rectangles with discs 1, 2, 3, 7 and 8 in Figure 2) or fully-automated activities (dashed rectangles with discs 4, 5 and 6 in Figure 2). From the produced artefacts, automatic extractions are performed on them:

- Interactive input and output tasks are extracted from the task models.
- Event sources and renderers are extracted from annotated code.
- User services and formal model nodes meta-events are extracted from formal models.

A correspondence window containing the extracted input/output elements can then be used to systematically map:

- Each interactive input task to an event source or a user services,
- Each interactive output task to a renderer or a formal model node meta-event.

During this operation, if an interactive task cannot be mapped to an input/output element of the mixed-criticality interactive application or, in the other case, if an input or output element of the mixed-criticality interactive application cannot be mapped to an interactive task, it means that there is a completeness issue. In the first case, this completeness issue highlights the fact that, a function is missing in the mixed-criticality interactive application and it has to be updated (loop back from the "Check completeness" rhombus to the relevant design and development step according to the software DAL in Figure 2). In the second case, this completeness issue highlights the fact that an interactive task is missing in the task model, which has to be updated (loop back from the "Check completeness" rhombus to the task modeling step in Figure 2).

Once all the completeness issues have been solved, the consistency between interactive tasks and DAL of the part of the interactive application software with which the interactive task is mapped is assessed. For example, if an interactive input task is listed as having to be executed with a critical part of the software, and that from the extracted input elements of the interactive application, only a matching event source is found instead of a user service, it means that there is an error in the list of tasks that have to be performed with critical parts of the software (loop back from the bottom rhombus "Check consistency" going through artefact "Errors in the list of tasks that are performed with critical parts of the software" in Figure 2). In that case, the DAL of the related software part has to be checked and depending on the outcome of this verification, the task models will be modified or the mixed-criticality application will be modified.

Once all the mapping of the extracted elements have been completed (i.e. is complete and consistent), the task models can be co-executed with the mixed-criticality interactive application. During this co-execution step, behavioral inconsistencies can be detected between the sequence of executed tasks and the behavior of the different software parts of the mixed-criticality interactive application. If such inconsistency is detected, either the task model will have to be modified (loop back from the "Misfits in task models" artefact in Figure 2) or the mixed-criticality interactive application will have to be modified (loop back from the "Misfits in mixed-criticality interactive application" artefact in Figure 2).

AN ARCHITECTURE AND A MODELING AND DEVELOPMENT ENVIRONMENT FOR ADRESSING MULTIPLE DEVELOPMENT ASSURANCE LEVELS IN MIXED-CRITICALITY INTERACTIVE APPLICATIONS

The architecture of the CIRCUS software environment that supports the presented process allows both correspondence editing and co-execution of task models and mixed-criticality systems is presented on Figure 3.

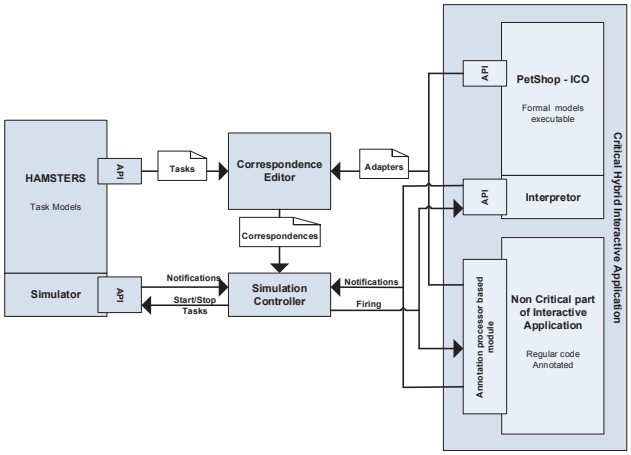

Figure 3. Design time/ runtime architecture of the CIRCUS environment.

Design time Architecture

This architecture is inspired by the work presented in [3] where the integration of the tools relies on a specific API provided by both HAMSTERS and PetShop, and since [21] on the API provided for handling annotated code.

On the left side of Figure 3, the integration relies on the Hamsters environment that provides a set of tools for engineering task models. It allows the editing and the simulation of task models this building interaction scenarios from the task models:

- From task specification a set of interactive tasks (input and output tasks) are extracted that represent a set of manipulations that can be performed by the user on the system and outputs from the system to the user.

- While building a scenario (e.g. simulating the task model) the Hamsters environment notifies the evolution of this scenario through a dedicated application programming interface (API) that allows data to be received from the simulator.

Similarly, on the right side of Figure 3, the integration relies on the ICO environment (PetShop) and on the annotated code that provide means for editing (PetShop or Java IDE) and executing the system (models played by PetShop and/or execution of the code):

- From the ICO specification we extract the activation and rendering function that may be seen as the set of inputs and outputs of the system model.
- From the annotated code we extract the set of events and graphical property changes produced by annotated widgets.
- While using the application (result of the execution of the specification within the PetShop environment and/or the annotated code) dedicated APIs provide the means for the notification of the evolution of this execution.

The principle of editing the correspondences between the two sides is to put together interactive input tasks (from the task model) with system inputs (from the system model or code) and system outputs (from the system model or code) with interactive output tasks (from the task model).

Property Verification of the correspondences

As described in the process presented in the third section, the CIRCUS framework provides support for verifying the following properties:

- **Completeness:** The former versions of the tool already provide means to assess the task model coverage by correspondences. For verification purpose, a particular focus must be put on having a correspondence for each critical task.
- **Consistency:** As tasks may be critical, they should be performed using a critical part of the application (higher DALs), while standard tasks may be performed without any particular attention put on the criticality of the interactive software part they are performed with (lower DALs).

Runtime Architecture

At runtime, a two way communication exists between the task side and the system side.

- **Between HAMSTERS and the Simulation Controller:** on one side HAMSTERS notifies changes in the current scenario to the Simulation. On the other side the Simulation Controller is able to ask to perform the corresponding task (according to the correspondence provided by the Correspondence editor), simulating the user action.
- **Between PetShop, annotated code and the Simulation Controller:** on one side the PetShop interpreter or the execution of the annotated code notify the Simulation Controller with the evolution of the current execution of the mixed-criticality application (notifications come from rendering and activation functions of the ICO models or the events triggered by the execution of the code). On the other side, the Simulation Controller fires the corresponding activation adapter (according to the correspondence provided by the Correspondence editor) simulating the user action.

As was the case in the previous version of the framework [3], this proposed architecture allows the task model driven execution of the system part (using PetShop or annotated code), and it conversely allows system driven execution and recording of the user actions in task based scenarios:

- In the case of task driven execution, executed tasks are translated by the Simulation Controller into user actions within PetShop (a sequence of transition firings) or the annotated code (making widgets trigger events or changing graphical properties).
- In the case of system driven execution, user actions are directly linked to the corresponding tasks from the task model and the user's action with the user interface of the application is recognized and mapped to an interactive task within the task model.

This runtime architecture behaves in the same way as presented in [3, 21] and is not illustrated in the case study section.

ILLUSTRATIVE EXAMPLE FROM ENGINEERING A MIXED-CRITICALITY INTERACTIVE COCKPIT APPLICATION

The presented example has been extracted from a case study in the avionics application domain. The FCU Software is a case study application that aims at allowing the aircraft crew members to interact with the Auto-Pilot and to configure flying and navigation displays. The FCU Software is a mixed-criticality interactive application as it enables the pilot to command and control DAL A software applications (such as pressure configuration) as well as DAL C software applications (such as waypoints display configuration).

The purpose of this section is to illustrate the main steps of the proposed design and development process. The case study presentation focuses on the engineering aspects of the approach. The three first sub-sections (discs 1, 2, 3 in Figure 2) illustrate the production of models and of regular code. Automatic extraction steps (discs 4, 5 and 6 in Figure 2) outputs elements that are used during the correspondence editing step (disc 7 in Figure 2) and presented in the dedicated sub-section. The co-execution step (disc 8 in Figure 2) is the same as that described in [3] and [21]. This section highlights the architectural aspects of the mixed-criticality FCU Software within the CIRCUS environment, the artefacts produced during the process and the correspondence edition step.

```
@EventSource(name="wpt", event="a661EvtSelection")
@Renderer(name="wpt", property="pictureReference")
private PicturePushButton buttonWPT;
…
private void ppbWPTA661EvtSelection (fr.irit.examples.a661.swing.A661EvtSelectionEvent evt){
    setND_Option1(ND_Options1.WPT);
}
```

Figure 4. Excerpt of regular code (widget declaration and event-handler)

a)b)

Figure 5. EFIS control panel (with WPT button activated)

Presentation of the FCU Software

In the Airbus A380, the Flight Control Unit is a hardware panel (i.e. several electronic devices such as buttons, knobs, displays …) providing two services: one managing aircraft information called EFIS (Electronic Flight Information System) and the other one managing the autopilot called AFS (Auto Flight System). Several of the actions that can be achieved by the pilot through the use of the Flight Control Unit are critical. The FCU Software is considered as a graphical interactive application for replacing the FCU hardware panel by graphical interfaces.

The crew members can interact with the application via the Keyboard and Cursor Control Units which gathers in a single hardware component a keyboard and a trackball. The upper part of the left panel of the EFIS (depicted in Figure 5) is dedicated to the configuration of the barometer settings. The top right panel of the EFIS page enables the display of several navigation information (such as waypoints).

Task modeling of the start descent activities

In this illustrative example, we focus on the activities that have to be carried out in order to start descent. The main activities that have to be carried out by the aircraft crew members to initiate a descent are a sequence of tasks that are depicted in the task model in Figure 6:

- The pilot in charge first receives the clearance that contains information about the flight level to be set and the baro settings to be configured.
- The pilot then has to configure the barometer settings (standard settings or particular settings corresponding to the current atmospheric pressure of the area the aircraft is entering) using the upper left part of the left panel in the EFIS page.
- S/he configures the flight level.
- S/he decides to check waypoints.
- S/he configures the display of waypoints in the Navigation Display by pressing the "WPT" button widget in the upper left part of the right panel of the EFIS_CP page.
- S/he analyses the planned trajectory.

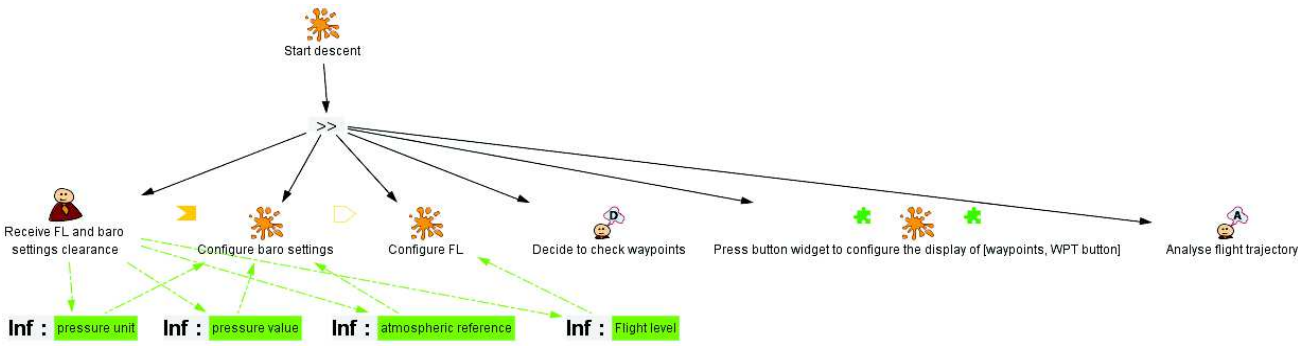

Figure 6. "Start descent" task model

Structuring mechanisms (previously proposed in [8]) have been used to produce all the task models of the case study. In Figure 6, the task "Configure baro settings" is a subroutine, meaning that it is composed of several tasks that are described in another task model (that the reader can find in Annex A). In Figure 6 as well, the task "Press button widget to configure the display of [waypoints, WPT button]" is an instance of a component task, meaning that it is composed of a set of tasks that can be performed in the same way with all the instances of a user interface component. This set of tasks is described in the task model (presented in Annex A). Concerning the "Start descent" activities, the task "Configure baro settings" has to be performed with critical functions of the avionics system, whereas the task "Press button widget to configure the display of [waypoints, WPT button]" has to be performed with a non-critical function of the avionics system.

Design and development of the mixed-criticality interactive application

Figure 8 presents the software architecture of this application. The lower part of Figure 8 corresponds to the Cockpit Display System (CDS) which is responsible for handling both graphical element of widgets and user inputs. This part of the application is considered reliable and is thus out of the scope of the case study (not directly involved in the correspondence editing). The upper part of Figure 8 corresponds to the mixed-criticality application involved in initiating a descent. Left-hand side is critical, handled and played using PetShop-ICO (management of baro settings), while the right-hand part is not critical and is edited and played as regular code (management of waypoints).

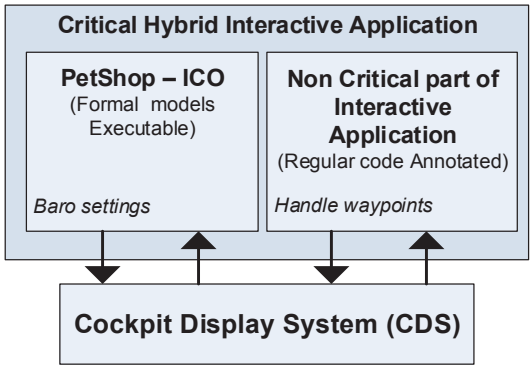

Figure 8. A mixed-criticality application architecture

Formal model executable

The critical parts of the software are designed and developed with using the ICO notation. ICO models are produced and then connected to the static UI elements following a process introduced in [24]. The right part of Figure 9 presents an excerpt of the ICO model describing the behaviour of the barometer control panel. In this excerpt, a token is present in the place "InHg" and transition "switchTo HPA" is available. This transition will be fired if the user clicks on the "InHG to HPA" radio button.

Regular code annotated

The non-critical part of the software is designed and developed using the Java language. Figure 4 presents an extract of the Java code that defines the behaviour of the waypoints display configuration. In this extract, the last line is the declaration of the WPT button widget and the two

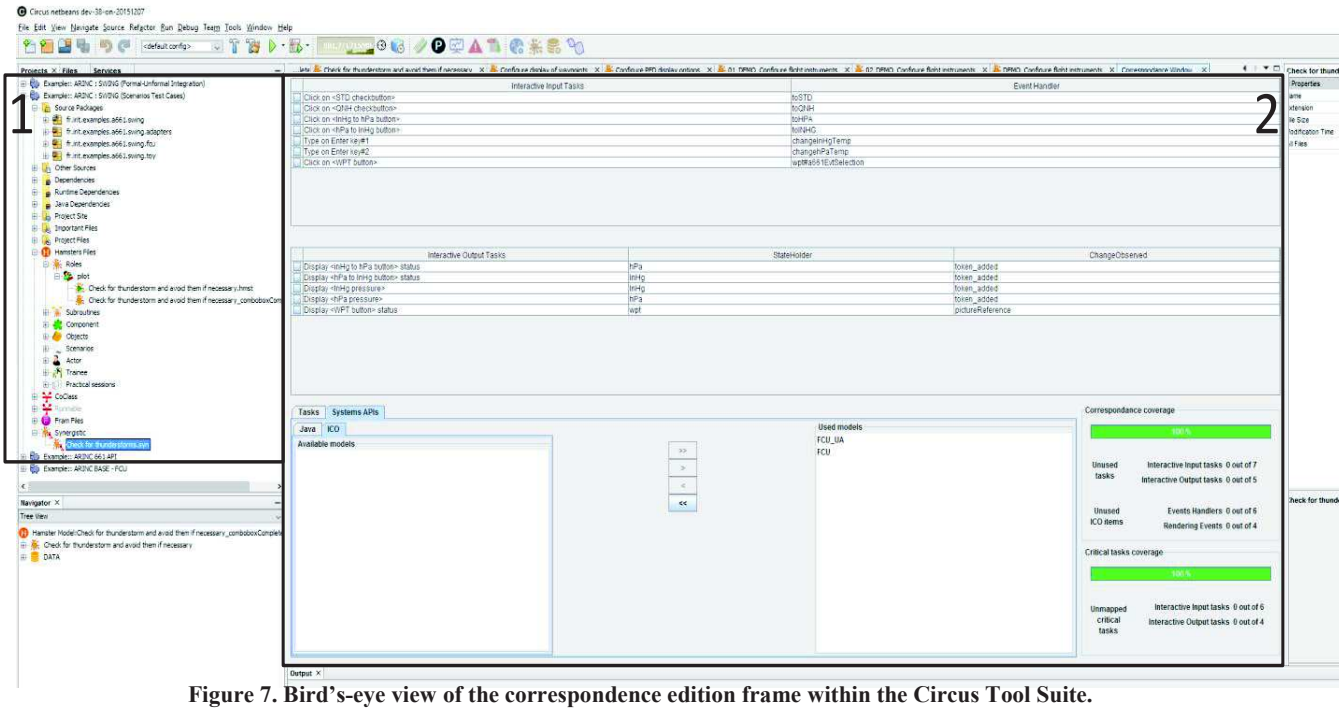

Figure 7. Bird's-eye view of the correspondence edition frame within the Circus Tool Suite.

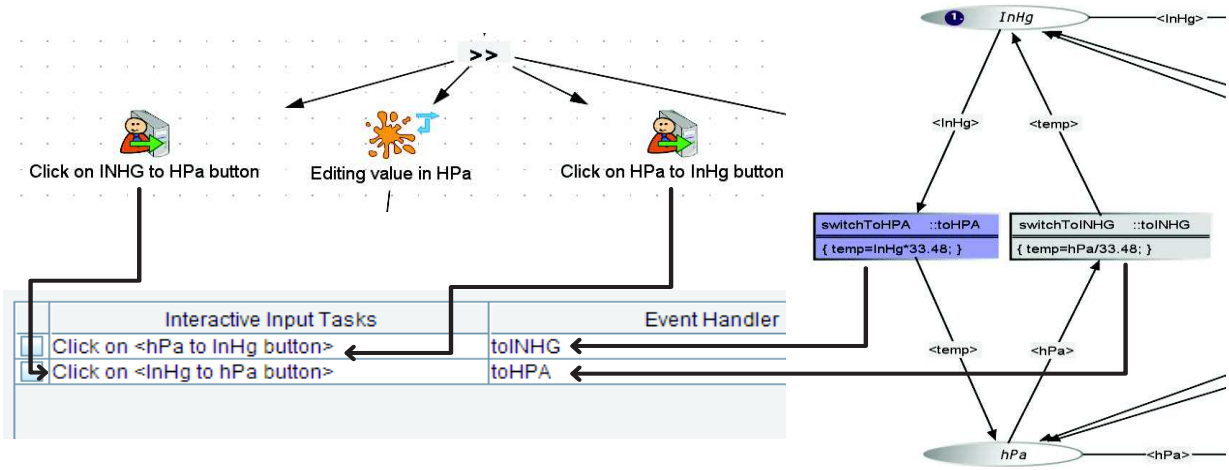

Figure 9. Correspondence editing between Hamsters model and ICO model

previous lines are the annotations that will be used by the CIRCUS environment for preparing the correspondence editing [21].

Correspondence editing between tasks and mixed-criticality interactive system

Both the task models and the interactive application (ICO models and the regular code) are handled within the Circus tool suite as illustrated in Figure 7. The left part (numbered 1) in Figure 7 displays the project set of files of the application (Java source files, Hamsters models, ICO models and correspondence files). The right part (numbered 2) in Figure 7 is the correspondence editing frame where the upper part is dedicated to edit the input and output correspondences between task models and the system part, and where the lower part is dedicated to the models and files selection (left) and to the correspondence coverage summary (right).

Editing correspondences between Hamsters models and the mixed-criticality application is done thanks to a table containing data extracted from task models, ICO models and annotated code as presented in Figure 9 and Figure 10. Figure 9 illustrates the editing of the correspondence between the task model "Configure baro settings" and the corresponding ICO model. The two critical interactive input tasks are put into correspondence with the two corresponding event handlers from the ICO model (*Click on <hPa to InHg button>* corresponds to the event handler *toInHg* and *Click on <InHg to hPa button>* corresponds to the event handler *tohPa*). In Figure 10, this correspondence is done with the widgets extracted from the annotated regular code (right part) and the task model "Press button widget to configure the display of [waypoints, WPT button]". For instance, the task *Click on <WPT button>* is put into correspondence with the triggering of the event *a661EvtSelection* of the widget called *wpt* (*wpt#a661EvtSelection* in the column Event Handler of the input table). When editing such correspondences, one could put into correspondence a critical task and a regular code element as illustrated in Figure 11. Putting inconsistent elements together triggers a warning amongst three possible one (the three numbered orange items in Figure 11):

1. In the first table, the orange background of the fourth line is a visual cue for warning that a critical task has been put into correspondence with a non-critical element.
2. In the correspondence coverage panel, it is displayed that some elements have not been used in a correspondence (here the event handler from the ICO model that should have been used instead of the regular code element).

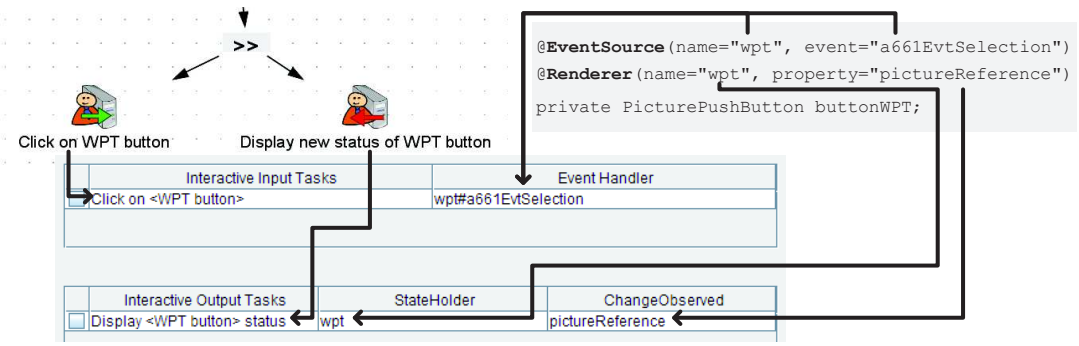

Figure 10. Correspondence editing between Hamsters model and regular annotated code

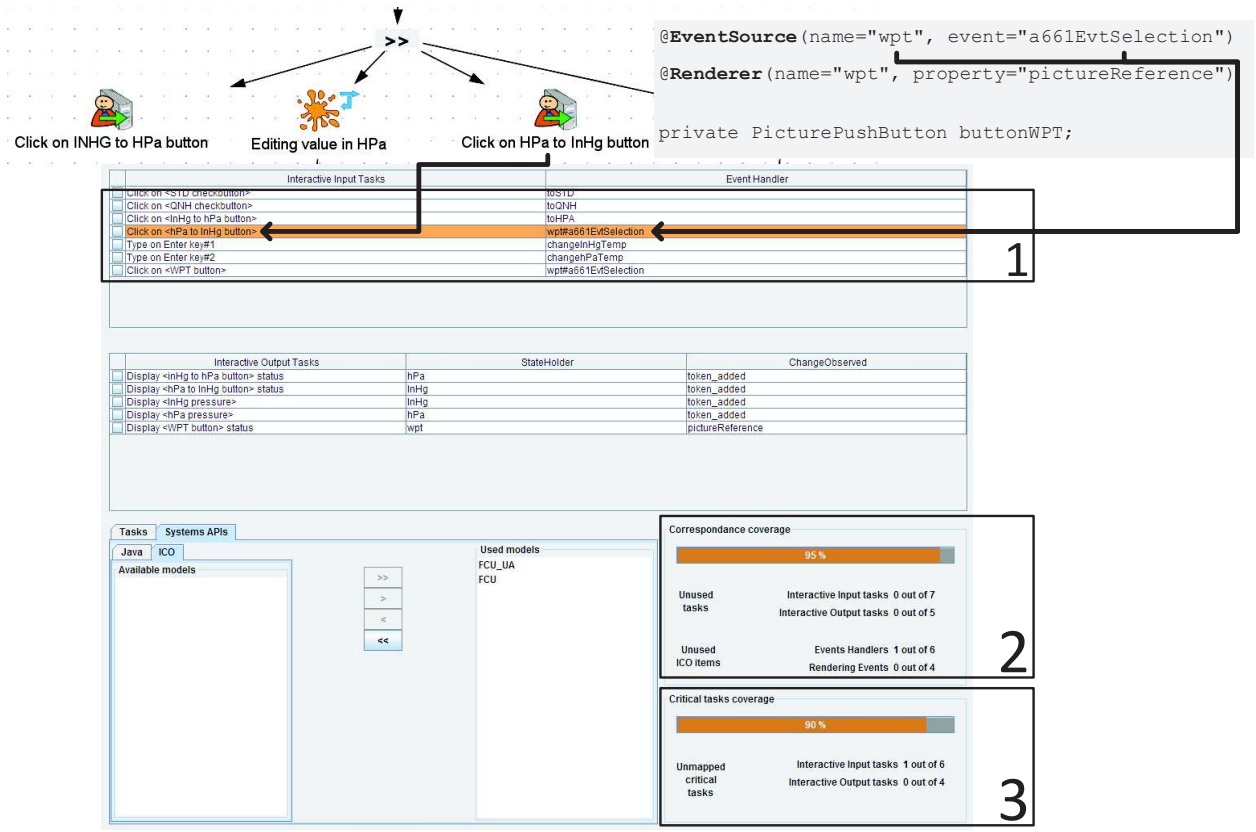

Figure 11. Correspondence editing with warnings: the critical task has been put into correspondence with a regular code element (not critical)

3. In the critical tasks coverage pane, it is displayed that one critical task has not been related to an ICO element.

The choice has been made here to only show warnings to the engineer instead of displaying errors as it supports iterative prototyping, making it possible to interact with the application even if correspondence editing has not been done in a complete manner. This section has presented: the architectural aspects of the CIRCUS tool applied to the mixed-criticality FCU Software, the artefacts produced during the process (task models, ICO formal models and regular code annotated), the correspondence editing step (taking as inputs automatically extracted interactive tasks, event sources, renderers, user services, formal model nodes meta-events).

CONCLUSION

Ensuring the compatibility between the description of task to be performed by operators and the systems on which those tasks are performed is necessary in order to assess the effectiveness of a system. Beyond that, such compatibility provides additional benefit such as construction of a training program based on the task descriptions, assessment of task complexity, and impact of systems evolutions on workload (among others). Previous work in the area of interactive systems engineering have provided tools and methods for ensuring such compatibility depending on how the underlying interactive system was built. [3] has presented a task-based framework using the ICO formal description technique while [21] has proposed a similar framework but only for systems developed using standard programming methods (extended with dedicated annotations). This paper has presented the CIRCUS framework capable of dealing with both approaches in a simultaneous way. Such a contribution is very relevant in the area of critical systems where different criticality levels coexist and thus where formal approaches and standard software engineering development methods are used. More precisely, in this article we have presented a task-model based approach for addressing multiple DALs for the interactive software. We have shown how this approach, composed of a process for the analysis of mixed-criticality interactive applications as well as a software modeling and development environment supporting that process, provide support for integration of and compatibility assessment between formal models, regular code and task models. However, we only have addressed systems offering WIMP interaction techniques and much more work has to be done to encompass more advanced interactive systems such as the one offering multimodal and multi-touch interactions.

ACKNOWLEDGMENTS

This work was partly funded by Airbus under the contract R&T Display System X31WD1107313.

REFERENCES

1. ARP 4754A, Guidelines for Development of Civil Aircraft and Systems, published by SAE International, 2010.
2. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., "Basic concepts and taxonomy of dependable and secure computing", 2004, *IEEE Trans. on Dependable and Secure Computing*, vol.1, no.1, pp. 11- 33.
3. Barboni E., Ladry J-F., Navarre D., Palanque P. and Winckler M. Beyond modeling: an integrated environment supporting co-execution of tasks and systems models. EICS'10, 165-174.
4. Bederson, B., Meyer, J., & Good, L. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters, 2(2), pp. 171-180.
5. Campos J.C., Harrison M.D. Model Checking Interactor Specifications. In Journal of Automated Software Engineering, vol.8, issue 3-4, August 2001, pp. 275-310.
6. Cuenca, F., Coninx, K., Vanacken, D., Luyten, K. Graphical Toolkits for Rapid Prototyping of Multimodal Systems: A Survey. Interacting with Computers 27(4): 470-488 (2015).
7. DO-333 Formal Methods Supplement to DO-178C and DO-278A, published by RTCA and EUROCAE December 13, 2011.
8. Forbrig, P., Martinie, C., Palanque, P., Winckler, M., Fahssi,R. Rapid Task-Models Development Using Sub-models, Sub-routines and Generic Components. Proc.of HCSE 2014, pp. 144-163.
9. Combéfis, S., Giannakopoulou, D., Pecheur, C., Feary, M. A Formal Framework for Design and Analysis of Human-Machine Interaction. Proceedings of IEEE System, Man and Cybernetics (SMC), Anchorage, USA, 2011, pp. 1801-1808.
10. CS-25 – Amendment 17 - Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes. EASA, 2015
11. DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification, published by RTCA and EUROCAE, 2012.
12. DO-254/ED-80. Design Assurance Guidance for Airborne Electronic Hardware, published by RTCA and EUROCAE, 2000.
13. Forbrig, P., Martinie, C., Palanque, P., Winckler, M., Fahssi,R. Rapid Task-Models Development Using Sub-models, Sub-routines and Generic Components. Proc.of HCSE 2014, pp. 144-163.
14. Fuchs, N.E. Specifications are (preferably) executable. Journal on Software Engineering, vol. 7, issue 5, September 1992, pp. 323-334.
15. Hayes, I., Jones, C.B. Specifications are not (necessarily) executable. Journal on Software Engineering, vol. 4, issue 6, Nov. 1989, pp. 330-338.
16. International Standard Organisation, ISO 8807:1989, Information processing systems -- Open Systems Interconnection -- LOTOS -- A formal description technique based on the temporal ordering of observational behaviour, 1989.
17. Jensen, K., Kristensen, L.M., and Wells, L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer 9*, 3-4 (2007), 213-254.
18. Kamel, N., Ait Ameur, Y. A Formal Model for CARE Usability Properties Verification in Multimodal HCI. In proceeding of IEEE International Conference on Pervasive Services, July 15-20, 2007, Istanbul, Turkey, pp. 341-348.
19. Kim, S.-K., Burger,D., Carrington,D. A. An MDA Approach Towards Integrating Formal and Informal Modeling Languages. In proc. of FM 2005: 448-464.
20. Manca, M., Paterno, F. Extending MARIA to support Distributed User Interfaces. In J.A. Gallud et al. (eds), Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, pages 33-40, 2011, Springer-Verlag.
21. Martinie, C., Navarre, D., Palanque, P., Fayollas, C. 2015. A generic tool-supported framework for coupling task models and interactive applications. 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '15). 244-253.
22. McMillan, K. L. Symbolic Model Checking, 1993, Kluwer Academic Publishers.
23. Myers, B. User interface software tools. ACM Trans. Comput.-Hum. Interact. 2, 1 (March 1995), 64-103.
24. 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. Transactions on Computer-Human Interaction, ACM SIGCHI, Vol. 16 N. 4, p. 1-56, 2009.
25. Oliveira, R., Dupuy-Chessa, S., Calvary, G. Formal verification of UI using the power of a recent tool suite. ACM SIGCHI symposium on Engineering interactive computing systems (EICS '14), 235-240.
26. Paternó, F., Santoro, C. Integrating model checking and HCI tools to help designers verify user interface properties. Palanque and Paternó (eds), DSV-IS 2000 Interactive Systems: Design, Specification and Verification. LNCS 1946, Springer 2001, pp. 135–150.

27. Salazar, E., Alonso, A., and Garrido, J. (2014). Mixed-criticality design of a satellite software system. In E. Boje and X. Xia (eds.), Proc. 19th IFAC World Congres1 12278-12283.

28. Silva, J.L., Campos, J.C., and Harrison, M.D. Formal Analysis of Ubiquitous Computing Environments through the APEX Framework. *EICS '12: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, ACM (2012), 131-140.

29. Soon-Kyeong Kim, Damian Burger, and David Carrington. 2005. An MDA approach towards integrating formal and informal modeling languages. In Proceedings of the 2005 international conference on Formal Methods (FM'05), John Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki (Eds.). Springer-Verlag, Berlin, Heidelberg, 448-464.